BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

CLUSTER ANALYSIS OF ENCRYPTED NETWORK TRAFFIC

SHLUKOVÁ ANALÝZA ŠÍFROVANÉHO PROVOZU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MYKOLA VORONTSOV**
AUTOR PRÁCE

SUPERVISOR **Ing. DANIEL POLIAKOV**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

165124

Institut:	Department of Information Systems (DIFS)
Student:	**Vorontsov Mykola**
Programme:	Information Technology
Title:	**Cluster Analysis of Encrypted Network Traffic**
Category:	Networking
Academic year:	2024/25

Assignment:

1. Get acquainted with statistical analysis methods at the level of network flows. Explore common characteristics used in the analysis. Study clustering methods and their evaluation possibilities.
2. Analyze the provided data and perform a basic statistical evaluation. Focus on the properties of network flows that may indicate the type of service, such as traffic patterns, temporal characteristics, and domain information. Identify relevant features for further analysis.
3. Design a method for identifying network services based on clustering. Consider suitable clustering algorithms and data preprocessing techniques.
4. Implement the proposed method and conduct measurements on a dataset of network services. Evaluate the results in terms of clustering quality and the method's ability to correctly distinguish between types of services.
5. Discuss the results, their limitations, and the potential applications of the method in practice, such as for detecting anomalies in network traffic.

Literature:

- Hofstede, R., Čeleda, P., et al. Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, 2014, pp. 2037–2064. doi:10.1109/COMST.2014.2321898.
- Moore, A. W., & Papagiannaki, K. (2005, March). Toward the accurate identification of network applications. In *International workshop on passive and active network measurement* (pp. 41-54). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Yamansavascilar, B., Guvensan, M. A., Yavuz, A. G., and Karsligil, M. E. Application Identification via Network Traffic Classification. In *2017 International Conference on Computing, Networking and Communications (ICNC)*, 2017, pp. 843–848. IEEE.
- Luxemburk, J., & Čejka, T. (2023). Fine-grained TLS services classification with reject option. *Computer Networks*, *220*, 109467.
- Erman, J., Arlitt, M., & Mahanti, A. (2006, September). Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data* (pp. 281-286).

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Poliakov Daniel, Ing.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2024
Submission deadline:	14.5.2025
Approval date:	11.10.2024

Abstract

The increasing use of encryption protocols like QUIC enhances Internet security but complicates network traffic analysis by removing access to payload data. This thesis explores unsupervised machine learning, focusing on clustering algorithms, to group encrypted network flows using basic flow-level features exported via IPFIX. K-Means and HDBSCAN are applied to the CESNET-QUIC22 dataset and evaluated using homogeneity and normalized mutual information. The Bhattacharyya distance is used to assess class separability. Results show that clustering can reveal meaningful traffic patterns and application groups, offering practical methods for encrypted traffic classification and anomaly detection.

Abstrakt

Zvyšující se využívání šifrovacích protokolů, jako je QUIC, zlepšuje bezpečnost, ale ztěžuje analýzu síťového provozu kvůli absenci datových náplní. Tato práce zkoumá využití metod učení bez učitele, zejména shlukovacích algoritmů, k analýze šifrovaných síťových toků pomocí základních atributů exportovaných přes IPFIX. Algoritmy K-Means a HDBSCAN jsou testovány na datasetu CESNET-QUIC22 a hodnoceny pomocí metrik homogenity a normalizované vzájemné informace. Bhattacharyyova vzdálenost slouží k posouzení oddělitelnosti tříd. Výsledky ukazují, že shlukování umožňuje identifikaci vzorců provozu i v šifrovaném prostředí a nabízí praktické využití pro klasifikaci a detekci anomálií.

Keywords

Encrypted network traffic, cluster analysis, unsupervised learning, flow-based traffic analysis, K-Means, HDBSCAN, QUIC protocol, ipfixprobe, CESNET-QUIC22 dataset, feature engineering, statistical analysis, Bhattacharyya distance, classification complexity, network flow clustering, anomaly detection, application fingerprinting, traffic classification.

Klíčová slova

Šifrovaný síťový provoz, shluková analýza, učení bez učitele, analýza síťového provozu na základě toků, K-Means, HDBSCAN, protokol QUIC, ipfixprobe, dataset CESNET-QUIC22, inženýrství atributů, statistická analýza, Bhattacharyyova vzdálenost, složitost klasifikace, shlukování síťových toků, detekce anomálií, identifikace aplikací, klasifikace provozu.

Reference

VORONTSOV, Mykola. *Cluster Analysis of Encrypted Network Traffic*. Brno, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Daniel Poliakov

Cluster Analysis of Encrypted Network Traffic

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Daniel Poliakov. The supplementary information was provided by Mr. Daniel Poliakov. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Mykola Vorontsov
May 13, 2025

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Ing. Daniel Poliakov, for his patience, guidance, and valuable insights throughout the preparation of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The increasing adoption of encryption across Internet services has significantly enhanced the security and privacy of network communication. However, this practice also introduced new challenges for network analysis because prior to universal encryption of most connections, classification often used inspection of packet payloads, which is not available. As a result, in a network community, there is a growing need for alternative methods of encrypted traffic analysis.

One of the possible alternative approaches is the use of machine learning techniques and, in particular, unsupervised methods such as clustering. These techniques are usually used to uncover hidden patterns in data without having knowledge of the samples' true labels. In the context of network traffic, clustering can be used to group flows that show similar statistical characteristics, potentially enabling the identification of applications or services even when the content of the communication is unavailable.

This thesis focuses on exploring the possibility of clustering encrypted network flows using a set of basic flow-level features. These features, such as packet counts, byte volumes, and flow durations, are readily available from most flow exporters and do not require access to packet payloads. The analysis is performed on the CESNET-QUIC22 dataset, which contains QUIC-encrypted traffic labeled by application, which can be useful for evaluating clustering performance.

The structure of the thesis is as follows. Chapter 2 presents the theoretical background, including clustering techniques and relevant evaluation metrics. Chapter 3 provides an exploratory analysis of the dataset, highlighting key statistical properties and traffic patterns that will be used during method design. Chapter 4 prepares the clustering method, including preprocessing steps, the choice of clustering algorithms, and parameter selection. Chapter 5 presents the results of the clustering experiments and discusses their implications. Chapter 6 concludes with a summary of the findings and suggestions for future research directions and practical usage.

Chapter 2

Theoretical background

For the purposes of this thesis, it is necessary to introduce and clarify several key concepts that form the foundation for the subsequent analyses. Familiarity with these concepts will facilitate a better understanding of the methods and experiments presented in the following chapters.

2.1 Network

As this thesis focuses on flow-based clustering of network services, it is important to clarify several fundamental terms related to networking. Understanding these concepts is essential for correctly interpreting the methodology and results presented in the following sections.

2.1.1 Flow

A **traffic flow**, or simply a **flow**, according to RFC 7011 [1], is defined as a set of packets or frames observed at a specific point in the network within a given time interval, sharing a common set of properties. A packet is considered part of a flow if it satisfies all defined criteria. This definition accommodates flows with zero or more packets and is compatible with packet sampling mechanisms, since sampling is considered a part of packet treatment.

Flow properties

The properties may be derived from:

- Packet headers (e.g., destination IP address, port number),
- Intrinsic characteristics of the packet (e.g., Multiprotocol Label Switching label count),
- Fields resulting from packet treatment (e.g., next-hop IP, output interface).

2.1.2 IPFIX

The **IP Flow Information Export (IPFIX) protocol**, standardized in RFC 5101 [4], provides a mechanism to export network flow information from routers, probes, or other devices to a collector for further processing and analysis.

Export Process

The IPFIX export process consists of three key components:

- **Exporter**: the device that observes and collects flow information (e.g., a router).
- **Collector**: the system that receives, stores and processes exported flow records.
- **Information Model**: a set of predefined data types and fields (Information Elements) used to describe the contents of flow records.

Template-Based Structure

IPFIX uses a template-based approach to describe the structure of flow records. Templates are transmitted periodically by the exporter and define which Information Elements appear in subsequent data records. This approach allows flexibility and efficiency in describing various types of flow data.

Transport and Encoding

IPFIX messages are typically transmitted over the Stream Control Transmission Protocol (SCTP), but support also exists for TCP and UDP. Each message includes a header, one or more sets of Template or Data Records, and is encoded in a binary format for compactness and performance.

Applications

IPFIX is widely used for traffic monitoring, network security, and performance analysis. It allows network operators and researchers to gain insights into network usage patterns, detect anomalies, and support decision-making based on empirical flow data.

2.1.3 ipfixprobe

ipfixprobe is a high-performance network flow monitoring probe developed by CESNET [3]. It is designed to analyze packet-level network traffic and export enriched flow information using the IPFIX protocol. The tool supports modular plugins that extract various features from traffic, such as statistical metrics, protocol-specific attributes, and security-relevant data.

ipfixprobe is optimized for high-throughput environments and is often used in conjunction with CESNET's data acquisition and monitoring infrastructure. It enables detailed flow-based analysis that is suitable for tasks such as anomaly detection, traffic classification, and network forensics.

2.1.4 QUIC

QUIC (Quick UDP Internet Connections) is a transport-layer protocol standardized in RFC 9000 [9]. It is designed to provide low-latency, reliable, and secure connections over the User Datagram Protocol (UDP). QUIC is designed as a replacement for TCP and TLS in a unified protocol, optimizing both connection setup and data transmission. QUIC is popular in modern web services, forming the transport layer for HTTP/3.

Comparison to TCP and TLS

QUIC addresses several inherent limitations of the traditional TCP and TLS stack:

- **Connection establishment latency**: TCP requires a three-way handshake and, when combined with TLS, an additional handshake is necessary before secure data transmission can begin. QUIC integrates the transport and cryptographic handshakes into a single process, often enabling data transfer within a single round-trip time (RTT), or even 0-RTT when resuming a session.
- **Head-of-Line blocking**: TCP enforces in-order delivery at the connection level, so when a packet is lost, all other packets in a sequence must be buffered until the missing packet is retransmitted and received. In contrast, QUIC supports multiple independent data streams within a single connection, allowing data on other streams to be transmitted without problem even if one stream experiences packet loss.
- **Integrated security**: QUIC requires the use of TLS 1.3 for all connections, ensuring end-to-end encryption, authentication, and integrity, unlike TCP, where encryption is layered on top via TLS.
- **Improved mobility support**: QUIC connections use Connection ID for identification instead of IP address and port pair, allowing seamless connection migration if a client's IP address changes. This cannot be natively supported by TCP due to its design.

2.2 Clustering

Clustering [18] is an unsupervised machine learning technique that focuses on grouping data objects based on the internal similarities found within the data itself, without the use of predefined class labels. The objective is to maximize within-cluster similarity while ensuring that outside-cluster differences are as distinct as possible. The difference is calculated using the chosen distance metric, which can differ among algorithms. Clustering differs from supervised classification, which relies on labeled data to build models for assigning class labels to new instances. Due to its exploratory nature, clustering is also referred to as *unsupervised classification*.

Distance metrics

A distance function $d(x, y)$ should satisfy the following equations [13]:

- **Law of indiscernability**: $D(x, y) \geq 0$ with equality iff. $x = y$,
- **Symmetry**: $D(x, y) = D(y, x)$,
- **Triangular inequality**: $D(x, y) \leq D(x, z) + D(z, y)$.

The most popular ones to choose from are:

1. **Euclidean distance** (Equation 2.1)

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.1}$$

2. **Manhattan distance** (Equation 2.2)

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{2.2}$$

3. **Cosine similarity** (Equation 2.3)

$$d(x, y) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \tag{2.3}$$

2.2.1 K-Means

K-Means [12] is a centroid-based clustering algorithm whose core idea is to minimize the within-cluster sum of squares between iterations. K-Means is one of the most popular clustering algorithms due to its simplicity and performance. However, it does not produce the same clustering for different initial points. As a consequence of this, some sets of initial points might result in more efficient clustering; therefore, there are some techniques to choose initial seeds that will produce better results.

Algorithm

1. Initialize n centroids $c_1, c_2, \ldots, c_n$, for example, using random points from the dataset or any other initialization method.

2. Assign every point $x \in X$, where X is the dataset, to a centroid c with a minimal value of the chosen distance function.

3. Recalculate centroids using their assigned points, as stated in 2.4.

$$c_{n+1} = \frac{\sum_{i=1}^{n} (c_n)}{|X|} \tag{2.4}$$

4. Return to Step 2 until any point did not change its cluster in this iteration.

2.2.2 DBSCAN

Ester & Krieger et al. proposed [6] a density-based clustering algorithm that can work efficiently with large databases. The algorithm takes two parameters: a minimal number of points (MinPts) required to form a cluster and an epsilon distance (Eps) threshold for a record p to be seen as a neighbor for another point q.

Definition 1 (Eps-neighborhood): Eps-neighborhood of a point $N_{Eps}(p)$ a set of the points whose $dist(p, q) <= Eps$.
Definition 2 (Directly density-reachable points): point q is *directly density-reachable from point* p if $q \in N_{Eps}(p)$ and $|N_{Eps}(q)| >= MinPts$.
Definition 3 (Density-reachable points): point q is *density-reachable* from point p if there is a chain of points p, p_1, p_2 ... p_n, q, where p_{i+1} is directly density-reachable from p_i.
Definition 4 (Density-connected points): point q is *density-connected* to point p when point r exists where both points p and q are density-reachable points from point r.
From the algorithm's point of view, cluster C is a set of mutually density-reachable and density-connected points.

Algorithm

1. Select any unvisited point P.
2. Find all points within a specified distance Eps around P — neighbors P_1... P_n.
3. If $|P_1 \dots P_n| < minPts$, then mark P as a noise. Otherwise, create a new cluster C with points P, $P_1 \dots P_n$.
4. For every point I in $P_1 \ \dots \ P_n$:
   (a) Mark the point I as visited.
   (b) Find all neighbors of I — $I_1 \dots I_n$. If $|I_1 \dots I_n| >= minPts$, add them to cluster C.
5. Repeat until there are no unvisited points in the dataset.

2.2.3 HDBSCAN

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise), which was proposed by Campello & Moulavi & Sander et al. [2], is an extension to the classic DBSCAN algorithm. DBSCAN is effective for flat cluster structures but struggles with varying density levels and nested clusters. HDBSCAN solves this problem by building a complete hierarchical clustering and extracting a flat clustering through a stability-based optimization.

HDBSCAN produces a hierarchy of clusters using density estimates and graph theory constructs and extracts the most significant clusters through an optimization process based on cluster stability. In the following, the theoretical definitions and algorithmic steps are outlined that form the core of HDBSCAN.

Definition 1 (Core distance): The distance from a point to its m_{pts}-nearest neighbor.
Definition 2 (Mutual reachability distance): defined as seen in Equation 2.5.

$$d_{\text{mreach}}(x_p, x_q) = \max\left(d_{\text{core}}(x_p), d_{\text{core}}(x_q), d(x_p, x_q)\right) \tag{2.5}$$

Definition 3 (Mutual reachability graph): A complete graph where edge weights represent mutual reachability distances.
Definition 4 (Minimum spanning tree): A subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles, and with the minimum possible total edge weight.

Algorithm

The HDBSCAN algorithm constructs a density-based cluster hierarchy and extracts the most significant clusters as follows:

1. Compute the core distances for all data points based on m_{pts}.
2. Build the **minimum spanning tree (MST)** of the mutual reachability graph.
3. Extend the MST by adding self-edges with weights equal to each point's core distance.
4. Process the MST:

(a) Set all objects to the same label for the root of the tree.

(b) With every iteration over the nodes, remove all edges of MST_{ext} in decreasing order of weights.

This approach allows HDBSCAN to effectively identify clusters of varying densities and shapes while minimizing sensitivity to parameter choices.

2.3 Clustering Evaluation

The clustering evaluation metrics, via Manning et al. [12], could be divided into two primary categories: those that require a gold standard, the correct partition that is known or that produced by human judges with a good level of inter-judge agreement; or the ones that don't rely on additional data to evaluate partitioning using its metrics, such as the variance ratio criterion or the silhouette coefficient.

Criteria

There are several basic requirements for a partition to be considered well-performed:

- High intra-cluster similarity: documents within a cluster are similar.
- Low inter-cluster similarity: documents from different clusters are dissimilar.

2.3.1 Homogeneity (purity) score

Purity score [12] is a simple and intuitive metric that calculates the percentage of correctly assigned records to appropriate clusters. Because the score represents a ratio, where in the best case, all items are associated with the corresponding group, the score can range between 0 and 1, where incorrect groupings have a score close to 0, and a grouping score with correct aggregation according to true labels is 1. However, the main drawback of this evaluation criterion is that there is no penalty for a large number of subsets because partitioning where every sample is nested in its cluster has a purity score of 1. In [15], Rosenberg and Hirschberg introduce a criterion referred to as **homogeneity**, which corresponds conceptually to the commonly used **purity** metric. Due to this equivalence, throughout this paper, the term *homogeneity* will be used to refer to the purity score, as both metrics capture the same aspect of cluster quality.

Calculation

1. Calculate the most frequently found true label for every cluster.
2. Divide the number of correctly assigned labels by the number of entries in the dataset.

2.3.2 Normalized Mutual Information (NMI)

Normalized Mutual Information [12] is an evaluation metric that uses the Mutual Information index to analyze partitioning. Mutual Information (MI) measures the extent to which a term affects the correct prediction of a label, and its calculation is shown on 2.7. The reason why MI is not used in its original form is the absence of punishment for the high number of clusters close to the number of records; therefore, NMI is freed from this

defect because it has a coefficient in the denominator that consists of the entropy of true and predicted labels, which grows larger with the number of clusters, consequently lowering the output value. Thus, NMI is always a number between 0 and 1, and is calculated as shown in Equation 2.8.

Definition 1. (Entropy): *Entropy* measures the uncertainty or randomness of a random variable. In clustering, entropy helps quantify how spread out or mixed the true labels or cluster assignments are. The formula for entropy is shown in Equation 2.6.

$$H(\Omega) = -\sum_{k=1}^{K} \frac{|\omega_k|}{N} \log\left(\frac{|\omega_k|}{N}\right) \tag{2.6}$$

$$I(\Omega; C) = \sum_{k=1}^{K}\sum_{j=1}^{J} \frac{|\omega_k \cap c_j|}{N} \log\left(\frac{N \cdot |\omega_k \cap c_j|}{|\omega_k| \cdot |c_j|}\right) \tag{2.7}$$

$$\mathrm{NMI}(\Omega, C) = \frac{I(\Omega; C)}{[H(\Omega) + H(C)]/2} \tag{2.8}$$

2.3.3 Contingency matrix

The **contingency matrix** is a commonly used tool to evaluate the agreement between two clusterings [16]. Given a set of ground truth labels and predicted cluster labels, the contingency matrix is a table where each element n_{ij} represents the number of samples that are in the true cluster i and the predicted cluster j.

Formally, let $U = \{U_1, U_2, \ldots, U_R\}$ be the set of ground truth clusters and $V = \{V_1, V_2, \ldots, V_C\}$ be the set of predicted clusters. The contingency matrix M is defined as 2.9.

$$M_{ij} = |U_i \cap V_j| \tag{2.9}$$

2.3.4 Pair confusion matrix

The **pair confusion matrix** [16] is a commonly used tool to evaluate the similarity between two groupings. It summarizes the pairwise agreement and disagreement between predicted and true cluster assignments by considering all possible pairs of samples.

The matrix consists of four components:

- **True Positives (TP)** C_{11}: Pairs that are in the same cluster in both the true and predicted clusterings.
- **False Positives (FP)** C_{01}: Pairs that are in the same cluster in the predicted clustering but not in the true clustering.
- **False Negatives (FN)** C_{10}: Pairs that are in the same cluster in the true clustering but not in the predicted clustering.
- **True Negatives (TN)** C_{00}: Pairs that are in different clusters in both the true and predicted clusterings.

The matrix is a useful basis for several pairwise clustering evaluation metrics, such as the Rand index, which will not be used, however, in this thesis.

2.4 Classification complexity

To select a smaller subset of applications, **class separation metrics** are used to identify a diverse set of applications. This approach ensures that the unsupervised machine learning techniques are evaluated across varying environments, progressively increasing the difficulty of the clustering task as more diverse applications are included. This methodology allows the assessment of the algorithms' ability to handle increasingly complex and heterogeneous application traffic.

2.4.1 Bhattacharyya distance

Bhattacharyya distance [17] is a parametric measure widely used to estimate class separability in classification problems.

The Bhattacharyya distance between two multivariate normal distributions with mean vectors μ_A, μ_B and covariance matrices Σ_A, Σ_B is defined as seen in Equation 2.10.

$$D_B = \frac{1}{8}(\mu_A - \mu_B)^T \left(\frac{\Sigma_A + \Sigma_B}{2}\right)^{-1} (\mu_A - \mu_B) + \frac{1}{2}\ln\left(\frac{\left|\frac{\Sigma_A+\Sigma_B}{2}\right|}{\sqrt{|\Sigma_A||\Sigma_B|}}\right) \tag{2.10}$$

Equation 2.10 consists of two terms: the first quantifies the separation due to the difference in class means, and the second captures the divergence due to covariance differences. The Bhattacharyya coefficient, on which this distance is based, measures the amount of overlap between two statistical distributions.

In empirical studies, Bhattacharyya distance has been shown to respond effectively to the elimination of irrelevant features, indicating its utility as a class separability measure in feature selection and dimensionality reduction tasks.

2.5 Development libraries and technologies

This thesis highly utilizes Python for implementation; however, standalone Python is not a convenient environment for data science. For this, scientists and developers usually use a wide range of libraries that implement the necessary instruments to be comfortable working with data and machine learning.

2.5.1 Scikit-learn

Scikit-learn is an open-source Python library that provides a wide range of state-of-the-art machine learning algorithms for both supervised and unsupervised tasks [14]. Developed with the goal of making machine learning accessible to non-specialists, Scikit-learn offers an easy-to-use interface built on top of the Python scientific ecosystem, particularly leveraging `NumPy`, `SciPy`, and `Cython`, which also implies high performance for end users. The library is distributed under the permissive BSD license, which enables its usage in both academic and commercial settings.

One of the core strengths of Scikit-learn is its consistent and minimalist design. The library exposes a simplistic API, which is defined by a standard interface with methods such as `fit`, `predict`, and `transform`, depending on the type of problem. This design simplifies the workflow for model development, evaluation, and deployment. Scikit-learn

also provides utilities for model selection, such as cross-validation and hyperparameter optimization through grid search, as well as pipelines that allow chaining of multiple processing steps.

2.5.2 Matplotlib

Matplotlib is an open-source 2D graphics library for Python that provides a comfortable environment for creating high-quality plots and visualizations [8]. Originally, it was developed to replicate the graphics capabilities of MATLAB within Python, but now Matplotlib enables users to generate publication-quality figures in various formats, including raster and vector outputs. It supports a wide range of plot types, such as line plots, scatter plots, bar charts, contour plots, and more.

Matplotlib is popular due to its ease of use, rich functionality, and compatibility with the Python scientific stack. Provides a Matlab-like procedural interface through the `PyLab`, which is currently being replaced by the `pyplot` module. At the same time, it offers a more Pythonic object-oriented API for advanced users and developers. Combined with its support for LaTeX integration and color mapping, Matplotlib has become an essential tool for scientific plotting and data visualization in Python.

2.6 Related work

Most current approaches to network traffic classification are based on supervised machine learning methods, where labeled training data are required to learn models that classify flows into specific application categories. Although these methods achieve high accuracy, they are impractical in situations where labeled data is unavailable or where new applications frequently appear. In contrast, this thesis focuses on unsupervised clustering methods, which allow classification and anomaly detection without the need for labeled data.

In [7], Gijón et al. proposed an unsupervised method for encrypted traffic classification in cellular radio access networks. Their method utilizes radio connection trace descriptors such as RRC connection time, uplink/downlink traffic volume, throughput, and activity ratios. Agglomerative hierarchical clustering was applied to segment traffic into broad service classes (e.g., messaging, browsing, streaming). Unlike the flow-based approach, which is approached in this thesis, their method operates on data collected from LTE base stations and is oriented toward coarse-grained classification for QoE management and planning. Their features are specifically tailored to the mobile radio interface, while the method uses per-packet flow-level features and applies to encrypted IP traffic in general-purpose networks.

In [5], Erman et al. proposed using clustering algorithms, specifically K-Means and DBSCAN, for traffic classification using only transport layer statistics such as connection duration, packet size, and byte count in each direction. They demonstrated that unsupervised methods could effectively cluster flows by application, even without payload inspection. DBSCAN, in particular, was able to detect noise points, highlighting its potential for anomaly detection. The method developed in this thesis is similar in spirit. However, their work uses flows' protocols as labels, whereas this paper aims to prepare a method that clusters flows based on their application.

In [19], Wang et al. introduced a novel unsupervised algorithm called Seed Expanding (SE) to cluster malicious traffic into different attack phases. They extracted simple flow-level features such as packet count, byte count, packet size statistics, and flag indicators,

which were discretized and binarized prior to clustering. The SE algorithm relies on similarity scores and weighted seeds to grow clusters iteratively. Their method is specifically designed for malicious traffic analysis, whereas the focus of the method presented in this paper is broader, covering general application classification in encrypted network flows.

Chapter 3

Exploratory statistical analysis

The experimental part of this thesis is based on the **CESNET-QUIC22** dataset [11], which is currently one of the largest publicly available datasets for QUIC traffic analysis. It consists of real-world encrypted traffic collected from high-speed backbone links within the CESNET2 network, which is the Czech national research and educational network that serves half a million people. The dataset spans a one-month period, capturing more than 153 million QUIC flows. The flows were exported using `ipfixprobe` and enriched with QUIC metadata.

Each record in the dataset represents a bidirectional QUIC flow and includes a comprehensive set of flow-level and packet-level features:

- **Basic flow statistics:** byte counts, packet counts, and flow duration in both directions.
- **Packet sequence metadata:** direction, payload sizes, and inter-packet times of up to the first 30 packets.
- **Histograms:** packet size and inter-packet time distributions in both directions using 8 logarithmic bins.
- **QUIC-specific fields:** Server Name Indication (SNI), QUIC version, and user-agent (where available).
- **Labels:** 102 service classes and 3 background classes were assigned based on SNI-domain mappings.

The services were manually mapped from the SNI fields using public documentation, Netify's Application Lookup Tool, and expert verification. Background traffic was further divided into Google, Facebook, and default classes to support open-world classification scenarios. Dynamic sampling was used to reduce class imbalance: high-volume services were downsampled up to a 1:15 ratio, while low-volume services were fully retained.

The dataset is designed to support tasks such as encrypted traffic classification, service fingerprinting, out-of-distribution detection, and behavior analysis. Given its scale and diversity, CESNET-QUIC22 allows robust evaluation of clustering and classification models under realistic traffic conditions. Ethical data handling was prioritized, with strong anonymization and no access to sensitive client identifiers.

For the experiments performed in this study, the **CESNET DataZoo** [10] is utilized, a framework for working with large network traffic datasets and to facilitate research in

the traffic classification area. The framework offers datasets in multiple predefined sizes: `XS` (10 million samples), `S` (25 million), `M` (50 million), `L` (100 million), as well as the full original dataset. For the purposes of this thesis, the `XS` version is chosen and a **subset of 100,000 flows** from **the W-2022-44 period name** is extracted to perform our clustering experiments.

3.1 Chosen features

In this study, our analysis is restricted to a set of **simple, precomputed features** that are directly available in the dataset, requiring no additional processing. The main idea behind this choice is to ensure the practicality and reproducibility of the designed clustering method in real-world scenarios. These features were selected to reflect characteristics that any standard flow exporter can reliably extract during normal operation. By focusing on universally accessible flow-level attributes, the method aims to maintain compatibility with typical network monitoring systems and promote the broader applicability of our method.

Simple flow features include:

- **DURATION**: Duration of the flow in seconds.
- **BYTES**: Number of transmitted bytes from client to server.
- **BYTES_REV**: Number of transmitted bytes from server to client.
- **PACKETS**: Number of packets transmitted from client to server.
- **PACKETS_REV**: Number of packets transmitted from server to client.

The per-packet information (PPI) sequence, which describes the first 30 packets of a flow. The features it offers are also included in the simple features that the method uses.

- **PPI_LEN**: Number of packets in the PPI sequence.
- **PPI_DURATION**: Duration of the PPI sequence in seconds.
- **PPI_ROUNDTRIPS**: Number of round trips in the PPI sequence.

3.2 Dataset study

It is essential to thoroughly examine this dataset to gain a deeper understanding of its structure, content, and statistical properties. This insight is crucial for informed decision-making when clustering and other types of data manipulation.

By analyzing the characteristics of the dataset, such as flow volume, application diversity, traffic directionality, and temporal indicators, it is possible to better prepare the preprocessing steps and interpret the results of the unsupervised learning methods. Furthermore, understanding the dataset's traffic patterns allows us to evaluate the predictability and flexibility of the clustering results more accurately.

Table 3.1: Main statistical characteristics of the dataset.

	PACKETS	PACKETS_REV	BYTES	BYTES_REV	DURATION
mean	44.7	150.6	16399.3	169256.5	9.5
std	626.0	2088.7	688135.5	2636014.8	29.6
50%	12.0	13.0	4178.5	5257.0	0.2
99%	541.0	2628.0	133281.5	3132120.6	136.0

Table 3.1 presents the default statistical characteristics of the most relevant features in the dataset. The data reveal that, on average, network flows are mostly download-oriented. This is evidenced by a significantly higher number of packets and bytes transmitted from the server to the client compared to those sent from the client to the server. This pattern is typical in modern Internet usage, where clients frequently request and consume large amounts of content hosted on remote servers.

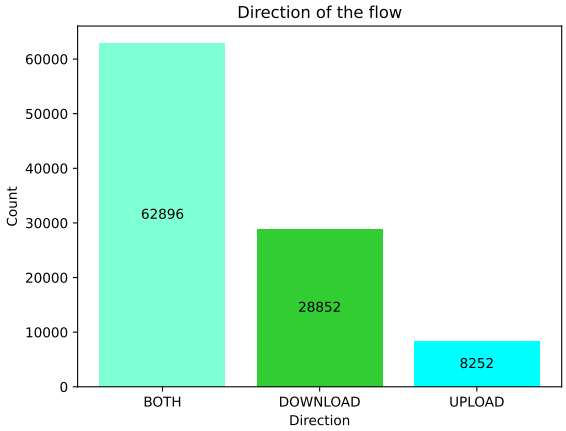

Figure 3.1: Number of flows with different data direction.

Figure 3.1 presents the number of flows grouped by their predominant data direction, either upload or download. A flow was classified as a download if the client sent at least 1.5 times more bytes than it received, indicating a larger volume of outbound data than inbound. In contrast, flows in which the client received significantly more data were marked as uploads. This heuristic helps differentiate between typical client-server interactions, such as web browsing or file transfers, and is particularly useful when explicit service labels are unavailable.

Figure 3.2 shows the distribution of the most common applications found in the dataset. As illustrated, most are API-based services and multimedia streaming platforms. These applications are characterized by frequent, lightweight exchanges (as seen with RESTful APIs or real-time updates) or sustained high-bandwidth flows (as with video or audio streaming). The prevalence of such services suggests that the dataset contains a mix of interactive and passive usage patterns, which can be valuable for identifying clusters based on traffic behavior rather than application protocol alone.

Figure 3.3 presents the distribution of traffic categories within the dataset. As observed, the most prevalent category is **Streaming Media**, indicating a significant proportion of flows associated with multimedia content delivery. This is followed by a large category la-

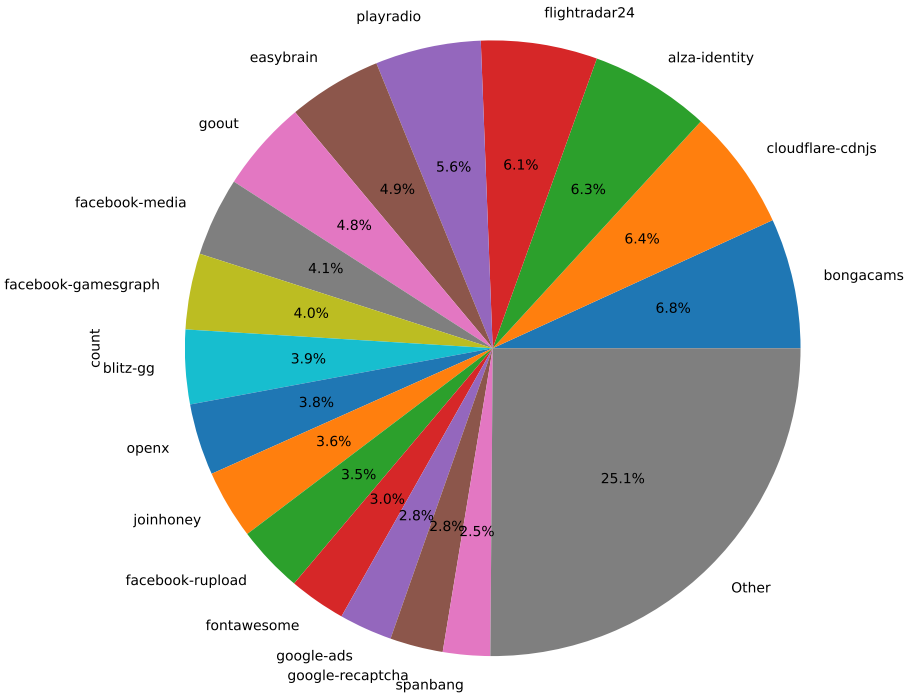

Figure 3.2: Distribution of apps in the dataset.

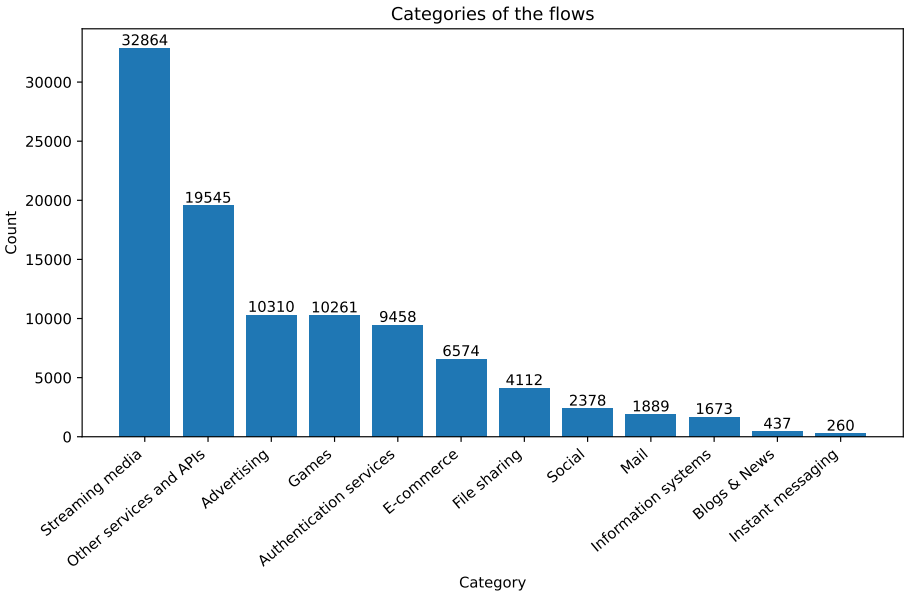

Figure 3.3: Distribution of categories in the dataset.

beled **Other Services and APIs**, suggesting that a substantial portion of the flows could not be accurately categorized, and they could not be a part of category-based classification. Additional prominent categories include **Advertising**, **Games**, and **Authentication Services**, each representing a considerable share of total traffic.

3.3 Concrete apps study

In this section, a selection of applications from the dataset is examined to characterize their traffic signatures using the defined set of features. Each application is compared against either the overall dataset or other individual applications in order to identify unique behavioral traits and determine the extent to which these traits differentiate them from others.

This analysis serves as a foundational step in evaluating the feasibility of application-level clustering based on the selected features. By identifying distinct patterns of behavior, this section aims to understand whether these features provide sufficient discriminatory power for unsupervised learning algorithms to reliably separate flows associated with different applications. Figures 3.4, 3.5, 3.6 show a comparison between percentiles of features of different apps involved and analyzed in the section.

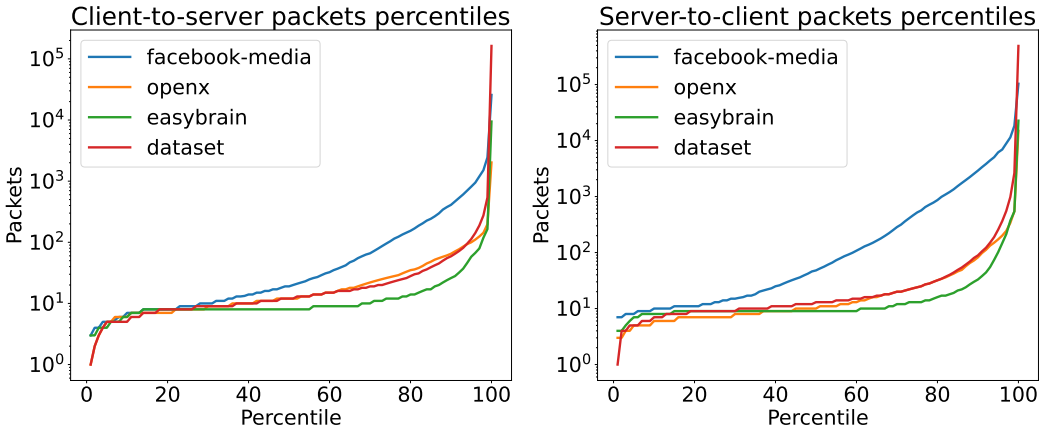

Figure 3.4: Packets percentiles distribution.

3.3.1 Facebook Media

Firstly, **Facebook Media** is in focus as a typical example of the most prevalent category: *Streaming Media*.

As expected, the data clearly reflects a dominant server-to-client communication pattern, which is typical for multimedia streaming applications. The median number of bytes sent from server to client is nearly ten times higher than that in the reverse direction, and this difference grows significantly at the upper percentiles, reaching up to a 40× difference at the 75th percentile. Although most of the data are sent to the client, there is still a measurable amount of client-to-server communication, which is likely associated with tasks such as control messages, metadata uploads, or low-resolution media synchronization, which are crucial for every media streaming app.

In particular, both the mean and median values for the volume of bytes from server to client exceed those of the overall dataset, indicating that a large volume of downloaded data

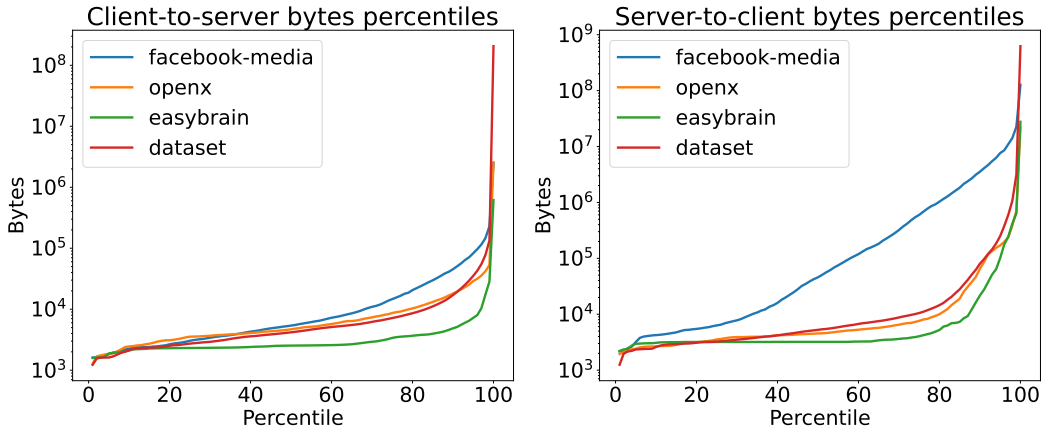

Figure 3.5: Bytes percentiles distribution.

may be used to distinguish Facebook Media traffic from other types of applications. The number of packets sent from server to client is also much higher than the dataset average, showing signs of a download-heavy flow pattern, which also supports the previously stated observation.

Regarding flow duration, Facebook Media sessions are relatively stable and moderately long. The median duration is approximately twice that of the overall dataset, and the mean is about 1.25 times higher. These data suggest that the connections are supported for a sufficient period to execute the transfer of large multimedia payloads, rather than for extended interactivity or session persistence.

With a median and mean of the PPI sequence length both approaching the upper boundary of 30, it is evident that most flows contain at least 30 packets, which is consistent with the assumption we made for large media file transfers or adaptive bitrate streaming.

Last but not least, the PPI sequence duration remains relatively low compared to the dataset's average, where the mean duration is 1.36 seconds. In comparison, the 90th percentile remains close at 1.11 seconds, suggesting limited variance and short-lived bursts of high-volume traffic. This also supports our theory about the transmission model, which is typical for media streaming, where data is sent in dense packets over brief intervals rather than in a steady, continuous stream.

3.3.2 OpenX

Secondly, we examine a representative application from the second most common category —*Advertising* — excluding the broad Other Services and APIs group. For this purpose, we selected **OpenX**, an advertising exchange platform.

The traffic pattern reveals a relatively balanced, bidirectional communication. Both the median and the 75th percentile values for bytes and packets transmitted in each direction are nearly identical, with slightly higher values observed in the client-to-server direction. However, certain edge cases demonstrate notable server-to-client dominance, indicating the presence of occasional unidirectional data bursts — possibly related to ad content delivery.

Compared to the overall dataset, OpenX displays median values for bytes and packets that are closely aligned with global statistics. However, its mean values are significantly higher in both directions, which may indicate the presence of outliers or skewed distributions. These findings suggest that OpenX — and as a result, other Advertising applica-

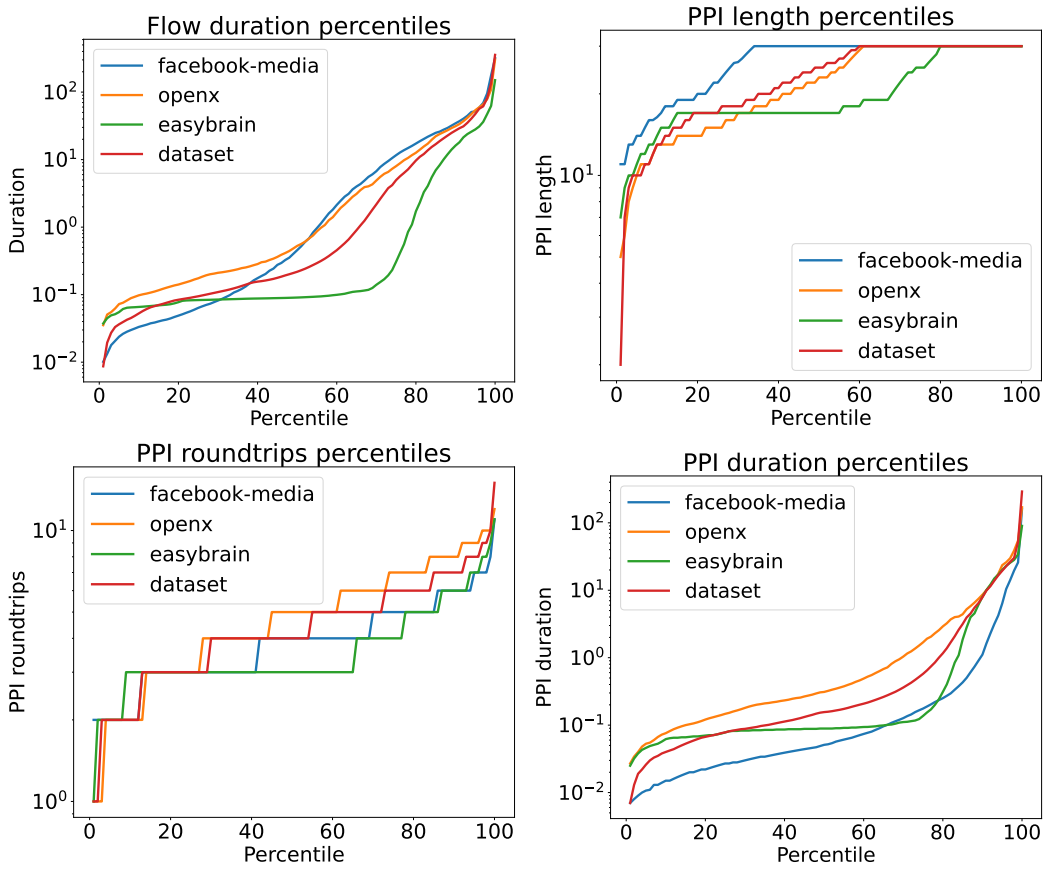

Figure 3.6: Duration and PPI sequence percentiles distribution.

tions — do not typically transmit large volumes of data. Instead, they expose moderate throughput similar to applications that serve lightweight assets such as banner ads, embedded trackers, and small media components, while also engaging in back-and-forth exchanges for user interaction logging or behavioral tracking.

The average flow duration for OpenX is close to 10 seconds, while the median is 0.53 seconds—slightly higher than Facebook Media's median duration, but with a 75th percentile value approximately 33 % lower. Nevertheless, the median duration remains around 50 % higher than the full dataset, suggesting that while OpenX connections are relatively brief, they often persist long enough to support the underlying interaction model typical of ad exchanges.

The PPI sequence length distribution is more closely aligned with the overall dataset than with Facebook Media. The median value of 23 indicates that at least half of the OpenX flows contain fewer than 30 packets, reinforcing the observation that advertising-related communication tends to be low in volume.

Interestingly, OpenX exhibits a higher number of PPI round trips than both the dataset average and Facebook Media. The median is elevated by approximately one unit, and the 75th percentile shows an even greater growth. This pattern could be an indicator for more interactive and two-way communication— which is likely caused because of negotiation, tracking, or verification mechanisms that require consistent client-server exchanges.

Finally, PPI duration for OpenX is considerably longer than that observed in both the overall dataset and Facebook Media. This extended duration may reflect the behavior of Advertising applications, which often do not transfer large payloads but instead maintain ongoing flows. At the same time, client-side data is being collected or computed.

3.3.3 Easybrain

Finally, we analyze a representative application from the last major category — *Games*. The application with the highest number of flows in this category is **Easybrain**, a mobile gaming platform.

The data reveals a generally bidirectional communication pattern between client and server, as indicated by the close similarity in median and 75th percentile values for packet counts in both directions. While this behavior mirrors the communication model seen in OpenX, the overall number of packets in Easybrain flows is approximately ten packets lower on average, and notably below the dataset mean, making packet volume a potentially valuable feature for distinguishing game-related traffic from other categories.

With both median and 75th percentile values for BYTES_REV surpassing their client-to-server counterparts, Easybrain shows a tiny advantage of server-based communication regarding byte-level statistics. Nevertheless, the difference is rather small, implying a rather equal data exchange. Easybrain manages lightweight data exchanges — probably related to state synchronization, matchmaking, or event reporting in online games — as most client-to-server flows remain under 7 kilobytes and 75 % of server-to-client flows fall below 4 kilobytes.

The flow duration for Easybrain is significantly shorter than that for OpenX and the whole dataset averages. The median duration is more than 50 % lower than the same from OpenX, while the 75th percentile is nearly 900 % shorter. This significant reduction in flow lifetime reinforces the hypothesis that Easybrain performs brief, high-frequency interactions rather than long-lived sessions, consistent with the behavior of casual or mobile games that perform periodic updates rather than continuous data streams.

The PPI sequence length, which reflects the number of packets within the initial flow segment (up to 30), further illustrates the lightweight nature of Easybrain's traffic. The mean value is approximately 20 packets, and the 75th percentile reaches only 25, suggesting that a substantial portion of flows do not fully utilize the 30-packet window. This again points toward a usage pattern dominated by short, bursty transmissions—typically for game state updates or server synchronization, rather than sustained data transfer.

3.4 Category apps study

Next, we perform a statistical analysis of the aggregated app categories to compare their average feature values with those of their corresponding representative applications. The goal of this comparison is dual: first, to verify the internal consistency of each category by checking whether the representative app exhibits typical behavior for its group; and second, to evaluate the potential for clustering at the category level rather than at the individual application level.

Such an approach can offer several advantages. If representative apps align well with their respective categories in terms of statistical characteristics, such as flow duration, packet size, or number of bytes sent in either direction, it would suggest that the current

categorization is meaningful. This, in turn, can prove that we can treat app categories as clusterable units in scenarios where app-level labeling is unavailable or incomplete.

On the other hand, if significant deviations are observed between a representative app and its broader category, this may indicate misclassification or highlight outlier applications with atypical traffic behavior, at least according to a given category. Identifying these anomalies can be valuable for improving the dataset's labeling accuracy, refining clustering methods, or even detecting applications with unique network signatures that merit separate analysis.

Overall, this analysis serves as an intermediate validation step, bridging low-level flow statistics with higher-level semantic groupings and providing insight into the structure and integrity of the dataset.

3.4.1 Streaming media

We begin our analysis with the most populated flow category in the dataset: **Streaming Media.** and its comparison with the app we discussed earlier — *Facebook Media*

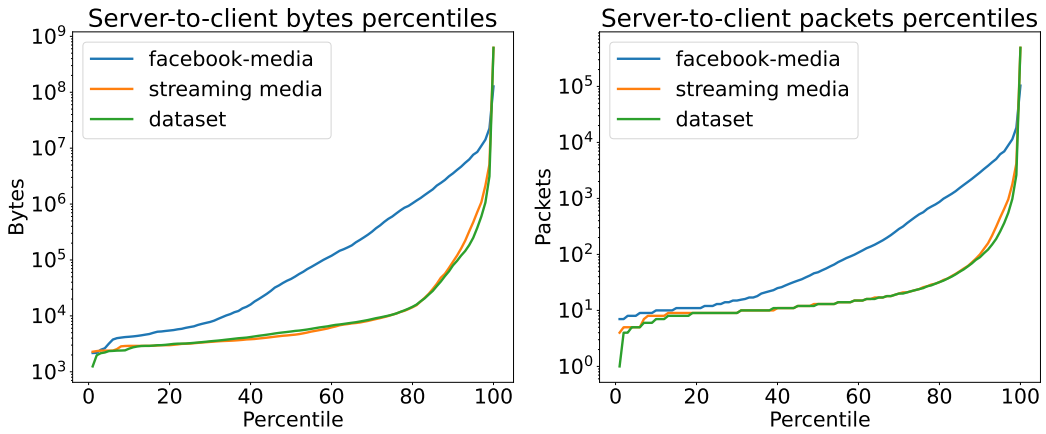

Figure 3.7: Comparison of key features of Streaming media.

Figure 3.7 depicts a percentile distribution of the most different features between the category and its according app. As expected, the communication pattern within this category is predominantly server-to-client, reflecting the nature of media delivery. The volume of data sent from the client to the server is closely aligned with the dataset's average. In contrast, the volume from the server to the client is significantly higher, which is typical for content consumption scenarios.

This category shares several characteristics with its representative application, Facebook Media, which was analyzed earlier. However, there are also significant differences, as, for instance, the average number of bytes sent from servers to clients in the overall category is approximately 10 times lower than that of Facebook Media. Similarly, there is almost a five-fold difference in the number of packets exchanged in both directions. These disparities suggest that Facebook Media represents a more data-intensive or „volumetric“ example within the category. Flow duration in the Streaming Media category is shorter than the dataset average, whilst being slightly longer than Facebook Media's. This indicates that many streaming applications may establish shorter-lived connections optimized for rapid media content delivery. Additionally, the PPI sequence length is close to the dataset mean but remains significantly lower than that of Facebook Media, further confirming the idea

that typical streaming flows are not sustained for long periods. The average number of PPI round trips is also near the dataset mean and lower than Facebook Media's, which implies that while some bidirectional communication or synchronization from the client is required, it is generally minimal. Interestingly, the PPI sequence duration for the category is 2.57 seconds—approximately one second shorter than the dataset mean, yet one second longer than that of Facebook Media. This may indicate burst-like traffic behavior, which suggests large volumes of data are exchanged quickly within short intervals. Overall, while the representative application and its category share most core patterns, the difference between several key indicators raises an important observation: the Streaming Media category may include a range of applications with varying behaviors. This variability creates challenges when attempting to cluster flows only based on category labels and suggests the possibility of encountering problems with such a way of partitioning.

3.4.2 Advertising

Next, we examine the **Advertising** category and compare it with its representative from the previous chapter: *OpenX*.

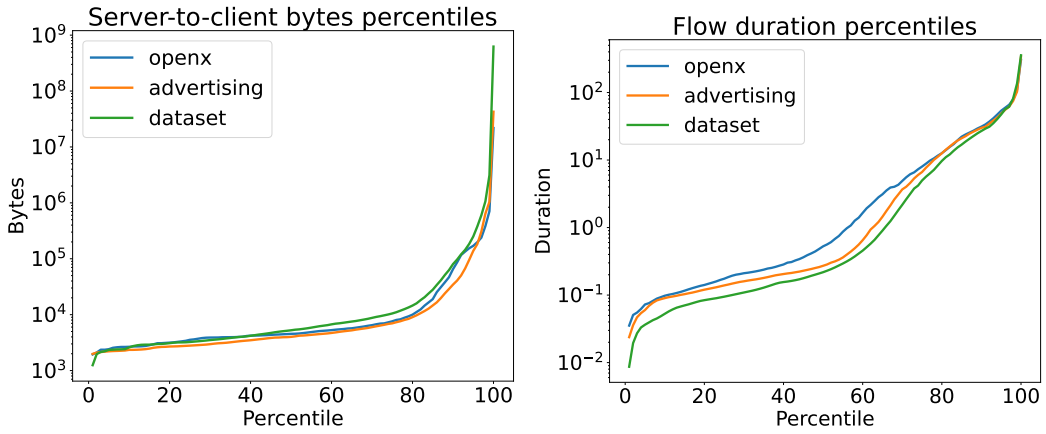

Figure 3.8: Comparison of key features of Advertising.

Figure 3.8 depicts a percentile distribution of the most different features between the category and its according app.

The 50th and 75th percentiles for packet counts in the Advertising category are quite similar to those of the OpenX application, as well as to the corresponding percentiles in the overall dataset. However, the server-to-client packet mean in the category is significantly lower than the mean observed in the full dataset, which indicates relatively balanced, bidirectional communication—a behavior typical of Advertising applications, which often deliver content and simultaneously collect analytical data. Interestingly, the category's mean is 33 % higher than OpenX's, suggesting that other applications within the category may transmit more data downstream than the OpenX application.

The byte indicators closely resemble those of OpenX, with only minor deviations across percentiles. However, the server-to-client byte mean is again 33 % higher than OpenX's, reflecting a slightly increased downstream traffic load within the category. When compared to the dataset's statistics, the percentiles are relatively close. Still, the client-to-server byte mean of the dataset is 60 % higher than the category's, and the server-to-client byte mean

is twice as high, indicating that the Advertising category typically occupies the lower to mid-range in terms of data volume.

Flow duration statistics between the category and OpenX reveal some differences. The 50th percentile for duration in the category is 0.27 seconds, which is almost half of OpenX's. The 75th percentile of the category is approximately 15 % lower than OpenX's, although their mean durations are relatively close. The data suggests that apps from the Advertising category establish relatively short-lived connections, with some of the applications potentially requiring even less time to complete data transfer. This is also supported by the slightly higher server-to-client mean in the category, indicating that some apps may focus more on ad delivery than telemetry collection. While the mean values for PPI duration are comparable between the categories and OpenX, the percentiles vary.

The PPI sequence length in the Advertising category is lower than that of both OpenX and the dataset in terms of both median and mean. This implies that many flows in the category do not reach the maximum 30-packet threshold, supporting the hypothesis of brief, low-volume connections in this type of traffic.

The PPI roundtrip count in the Advertising category is higher than the dataset's but lower than OpenX's, suggesting that while bidirectional communication is a common trait among applications in this category, it may be less frequent than what is observed in OpenX. This also supports the theory that certain applications in the category place greater emphasis on server-to-client communication.

In summary, while OpenX provides a reasonable representation of the Advertising category, there are measurable differences that point to variability in traffic patterns among the applications within the category.

3.4.3 Games

We now turn to the final category under consideration: Games, and compare its aggregated behavior to that of its representative application, Easybrain, analyzed previously.

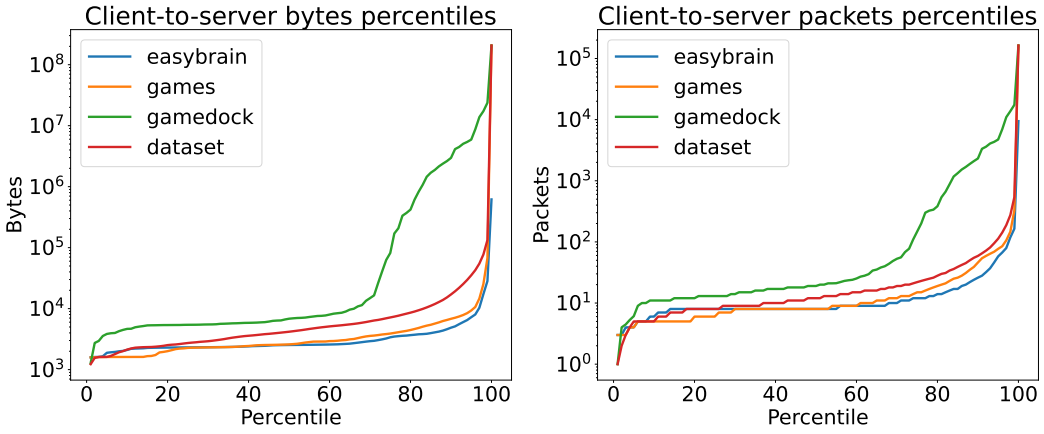

Figure 3.9: Comparison of key features of Games.

Figure 3.9 depicts a percentile distribution of the most different features between the category and its according app.

The median number of packets in the Games category is comparable to that of Easybrain; however, differences emerge at higher percentiles and in the mean values. This is particularly evident in the client-to-server direction, where the mean value for the cate-

gory is three times higher than Easybrain's. And also, the mean number of packets in the client-to-server direction for the Games category is approximately 45 % higher than entire dataset's.

Flow duration metrics also show divergence between the category and Easybrain. For example, the 75th percentile of the Games category is 2.5 times higher than the Easybrain application indicator, while the mean duration is 0.4 seconds lower, indicating that although many applications within the category establish longer-lived connections than Easybrain, others may exhibit significantly shorter sessions.

The volume of bytes from client to server has a huge difference with the app. While the mean and median values for server-to-client bytes in the category are relatively close to Easybrain's, the mean client-to-server byte volume in the category is nearly 15 times greater, suggesting the presence of an outlier application with a substantially different traffic signature.

Upon closer inspection, the Gamedock application was identified as this outlier. It displays a markedly different traffic pattern from both the category and Easybrain. The medians for packets sent in both directions are roughly twice as high as those in the category and in Easybrain. Furthermore, the mean number of packets sent from client to server and from server to client are approximately 24 times and 5 times higher than Easybrain's, respectively.

The byte-level analysis supports this distinction. While Gamedock's mean server-to-client byte volume is comparable to Easybrain's, its median is three times higher. Most notably, Gamedock's mean client-to-server byte volume is approximately 2 MB, which is about 500 times greater than Easybrain's. These characteristics suggest that Easybrain exhibits a notably distinct traffic profile, which is highly focused on high-volume client-to-server with a bit of server-to-client interactions.

Flow duration for Gamedock also exceeds that of Easybrain, with a mean of 7.88 seconds compared to Easybrain's 4.38 seconds. This increased duration can be logically explained as larger volume of data transferred, which requires longer connection times.

The PPI-level features further support this distinction. Gamedock's PPI sequence length has a mean of 27.48, compared to Easybrain's 20, indicating that most flows contain the maximum of 30 packets. Additionally, the PPI round-trip count is 50 % higher than that of Easybrain.

These findings reinforce the conclusion that category-based classification or clustering is not suitable, as categories often contain highly heterogeneous applications with significantly different traffic signatures. Since categorization is based solely on SNI information, this paper will focus instead on per-application clustering, avoiding category-level generalizations.

Chapter 4

Method design

In this chapter, we develop a clustering methodology that will be used for flow-based identification of network services. The design of this approach is strongly influenced by the structure and ideas derived from the dataset analysis presented in Chapter 3, and although certain methodological choices are made with dataset-specific characteristics in mind, the overall framework remains broadly applicable and would require minimal adaptation when applied to other datasets.

A critical aspect of this process is the selection of an appropriate suite of clustering algorithms capable of effectively partitioning network traffic. An equally important part is the design of a preprocessing pipeline that will transform the dataset into a representation suitable for use with the selected algorithms.

4.1 Preproccessing

The dataset is preprocessed using a **standard scaler**, which normalizes each feature by subtracting the mean and scaling to unit variance. This standardization step is essential to ensure that all features contribute equally to the distance metrics used by the clustering algorithms, thereby preventing any single feature from disproportionately influencing the clustering outcome.

4.2 Feature engineering

For this experiment, we introduce two additional features: **BYTES_PER_PACKET** and **BYTES_PER_PACKET_REV**, which are calculated as the ratio of `BYTES` to `PACKETS`, and `BYTES_REV` to `PACKETS_REV`, respectively. These new features represent the average size of transmitted packets in both communication directions. They are particularly useful in distinguishing between application types: data-centric applications, such as streaming media or download-oriented services, tend to show higher values for these features, whereas applications characterized by frequent, but small exchanges, as signaling or telemetry, etc., typically have lower values. Including these features enhances the method's ability to capture differences in traffic behavior beyond simple volume-based metrics.

4.3 Clustering algorithms

In this thesis, we select several clustering algorithms to evaluate their effectiveness in flow-based network traffic classification. The selection is motivated both by the distinct characteristics of the algorithms, which offer different clustering paradigms, and by the need to compare their performance in the task. Some choices are also informed by prior work, such as the study presented in [5], which demonstrated the practical utility of certain methods in this domain. The first algorithm selected is **K-Means**, which serves as a baseline due to its simplicity, broad usage, and established effectiveness in numerous unsupervised learning tasks, including network traffic analysis. Lastly, we include **HDBSCAN**, a state-of-the-art clustering technique available in the Scikit-Learn ecosystem. Although HDBSCAN has shown strong performance on complex and heterogeneous datasets, it has not yet been specifically validated in the context of network traffic clustering, and including it in this evaluation enables us to measure its potential applicability in the domain.

4.4 Parameter tuning

Parameter tuning is a critical aspect of clustering performance, and careful selection of these parameters is essential to ensure the validity and reliability of the experimental outcomes. In this study, we take deliberate steps to configure algorithm parameters appropriately in order to prevent experimental contamination and bias.

For **K-Means**, the primary parameter is the number of clusters, K. Given that the dataset includes labeled instances, we can directly bind K to the number of known classes. However, to gain deeper insights into how clustering performance scales with increased model complexity, we perform several K-Means runs using different multiples of the true number of labels. This allows us to analyze the sensitivity of the clustering metrics to the number of clusters.

Finally, **HDBSCAN** requires only one main parameter: the minimum cluster size. The minimum cluster size parameter is set to 7, which is based on the smallest number of samples associated with any single label in the dataset.

4.5 Evaluation metrics

The quality of the clustering results is evaluated using two principal metrics: **homogeneity score** and **Normalized Mutual Information** (NMI), both of which were introduced earlier. These metrics evaluate the extent to which the resulting clusters correspond to the ground truth labels, either application-specific or category-based.

The **homogeneity score** measures whether each group contains only members of a single class. Although it is an intuitive and interpretable metric, it has a known tendency to increase with the number of clusters, potentially overestimating clustering quality in over-partitioned scenarios. Despite this limitation, it remains useful due to its simplicity and transparency.

As an addition, **NMI** is used as a secondary and more reliable metric. NMI quantifies the mutual dependence between the predicted cluster assignments and the true labels, normalized to ensure comparability across different clustering configurations. Importantly for the method and the studies, it provides a penalty for large numbers of clusters, helping to evade the problems seen in homogeneity and providing a more balanced view of clustering

performance. Therefore, while homogeneity offers initial insights, NMI is considered a more reliable indicator for comparing clustering algorithms under varying conditions.

In addition to homogeneity and NMI, we employ two additional evaluation methods: the **contingency matrix** and the **pair confusion matrix**. The contingency matrix is primarily used for clustering scenarios involving a small number of applications, in the experiment up to five, as it becomes increasingly difficult to interpret with a larger number of labels or clusters. This matrix provides valuable information on which applications are misclustered with others, allowing for a detailed analysis of clustering behavior at the application level.

In reverse, the pair confusion matrix is much more useful for experiments involving a larger number of applications, where the contingency matrix loses interpretability. Although the contingency matrix provides the same information and more, the pair confusion matrix offers a more scalable way to quantify clustering performance. Specifically, it reveals the proportion of correctly clustered flow pairs and the types of misclassification errors (false positives and false negatives), which makes it particularly useful for comparing clustering algorithms across more complex scenarios.

4.6 Visualization

The results of clustering are projected using t-SNE on a two-dimensional plane, which is available in the Scikit-Learn library.

4.7 Iterative refinement process

The primary approach used in this study is a progressive increase in the number of clustered applications. Initially, we tried to cluster all applications simultaneously. If this approach is classified as ineffective using the discussed metrics and due to high heterogeneity or overlapping traffic patterns, we perform a stepwise strategy, starting with just two applications and incrementally increasing their number. This gradual expansion is also performed with parameter tuning at each stage, allowing us to adapt the clustering configuration and evaluate its effectiveness under varying levels of complexity.

Subsets of applications are selected using **class separation metrics**, specifically the *Bhattacharyya distance*. The selection process follows a greedy strategy: initially, the two applications with the highest pairwise separation score are selected. Subsequently, additional applications are evaluated based on their minimum separation score, which is based on each app in the set already selected. At each step, the application that maximizes this minimum score is added to the subset. This approach guarantees that the chosen applications have the highest possible degree of mutual separability, thereby facilitating more meaningful clustering evaluations.

4.8 Practical implementation

All technologies, algorithms, and evaluation metrics discussed in this chapter are implemented using the Scikit-Learn library for Python, which serves as the primary tool for clustering algorithm development and evaluation. The CESNET DataZoo provides the dataset in the form of a pandas[1] DataFrame, enabling efficient data manipulation and sta-

[1] https://pandas.pydata.org/

tistical analysis. For visualization purposes, the matplotlib library is used, specifically its pyplot submodule, which is popular to generate high-quality plots and graphical summaries.

4.9 Summary

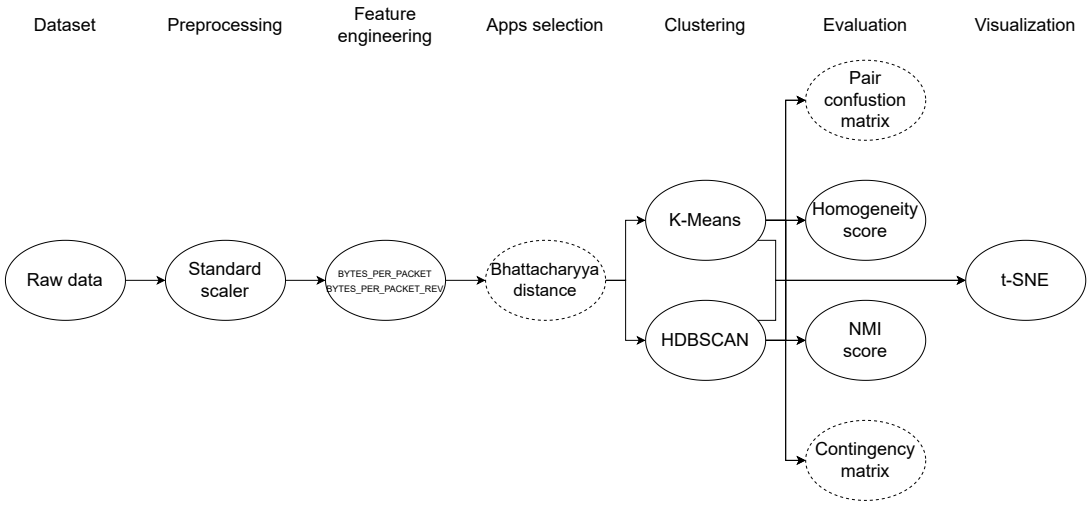

Figure 4.1: Final method design.

Figure 4.1 shows the diagram of the final method design, steps marked with a dotted line are optional and not featured in every experiment in this thesis. In summary, the preprocessing stage includes a Standard Scaler to normalize feature distributions, ensuring equal contribution of each feature during clustering. The selected clustering algorithms include K-Means and HDBSCAN, chosen for their proven effectiveness in partitioning. For evaluation, we utilize two well-established metrics: the Homogeneity Score, which measures the consistency of labels within clusters, and Normalized Mutual Information (NMI), which provides a balanced assessment accounting for both cluster quality and complexity. All preprocessing, clustering, and evaluation tasks are implemented using Python, primarily powered by the Scikit-Learn library. As a potential direction for future work, including additional clustering algorithms, such as hierarchical or spectral clustering, could further enhance the flexibility and effectiveness of the method.

Chapter 5

Simple features clustering

In this experiment, we focus on clustering network flows using only the 'simple' features that were introduced in the previous chapter. The objective is to evaluate whether it is possible to meaningfully partition the flows based only on these selected characteristics, or at least to determine whether the resulting groupings have a degree of structure sufficient for practical applications.

5.1 Initial clustering

We begin the clustering experiments by applying the **K-Means** algorithm to the whole dataset, which contains 100,000 network flows corresponding to 101 unique application labels, which were analyzed in the previous chapter. As K-Means requires only specification, that is, the number of clusters K, we initially set this parameter to 101, corresponding to the number of known labels, and subsequently explore its multiples.

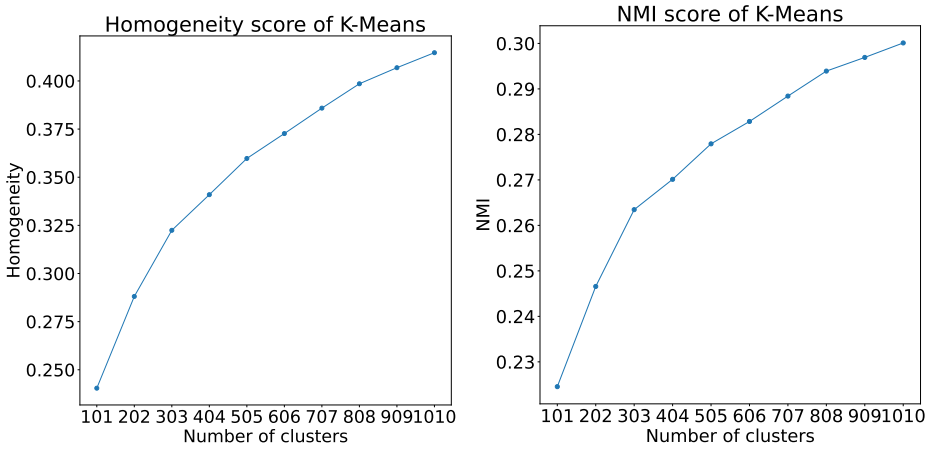

Figure 5.1: Homogeneity and NMI scores of whole dataset clustering using K-Means.

Figure 5.1 depicts the homogeneity and NMI scores achieved by K-Means using different initial configurations. Although the homogeneity score improves as the number of clusters increases, this rule works due to the nature of the metric itself. Even at its highest point, the homogeneity score only slightly exceeds 40 %, indicating that the clustering quality remains relatively low. A general upward trend is observed on the NMI graph, suggesting that adding more clusters improves the clustering accuracy rather than decreasing the score.

This is particularly notable because NMI penalizes large numbers of clusters. Nevertheless, the overall NMI values remain unsatisfactory for practical use.

Table 5.1: Normalized pair confusion matrix of whole dataset K-Means and 101 K.

	Predicted not paired	Predicted paired
True not paired	0	0.33
True paired	0.62	0.05

The normalized confusion matrix for K-Means clustering with 101 clusters is shown in Table 5.1. The matrix reveals that only approximately 5 % of the flow pairs are correctly clustered. In contrast, 33 % of pairs represent false positives, where flows from different applications are assigned to the same cluster, and 62 % represent false negatives, where flows belonging to the same application are split between different clusters.

HDBSCAN identified a total of 2,063 clusters when applied to the full dataset. Without removing noise-labeled points, the algorithm achieved a homogeneity score of 0.324 and an NMI score of 0.307. After excluding noise points, the homogeneity score and the NMI score improved significantly to 0.67 and 0.455, respectively. However, the algorithm labeled 51,786 of 100,000 flows, more than half of the dataset, as noise. Although HDBSCAN outperforms K-Means in terms of cluster purity, its effectiveness comes at the cost of drastically reducing the usable portion of the dataset. Consequently, although the remaining flows are grouped with relatively high precision, this result severely limits the model's applicability for comprehensive classification, as a substantial portion of the traffic remains unclassified.

Table 5.2: Normalized pair confusion matrix of whole dataset HDBSCAN without noise.

	Predicted not paired	Predicted paired
True not paired	0	0.13
True paired	0.85	0.02

Table 5.2 presents the normalized confusion matrix for the HDBSCAN clustering results. The matrix indicates that only 2 % of flow pairs are correctly clustered. This suggests that despite HDBSCAN achieving a higher homogeneity score than K-Means, its overall clustering performance is not necessarily better, especially when considering the broader classification objectives. The distribution of incorrect pairings shows that 13 % of flow pairs are false positives and 85 % are false negatives. These findings highlight that HDBSCAN may be more suitable in scenarios where minimizing inter-application overlap within clusters is prioritized, even at the cost of fragmenting flows from the same application or reducing the overall dataset size due to noise exclusion.

In conclusion, it is possible to state that it is hard to cluster the whole dataset using only „simple“ features with our initial configuration. However, we can try to start with successfully clustering two apps, adding new apps one by one, and if it is hard even with a few apps, then we can state that it is impossible to cluster flows using only these features.

5.2 Two apps

Due to the inability to effectively partition all application labels at once, we begin with a smaller subset of applications and increase their number step by step. We started by selecting the two most distinct applications, as indicated by the *Bhattacharyya distance* distance metric: **Alza Web API** (ID 40) and **MDPI** (ID 81).

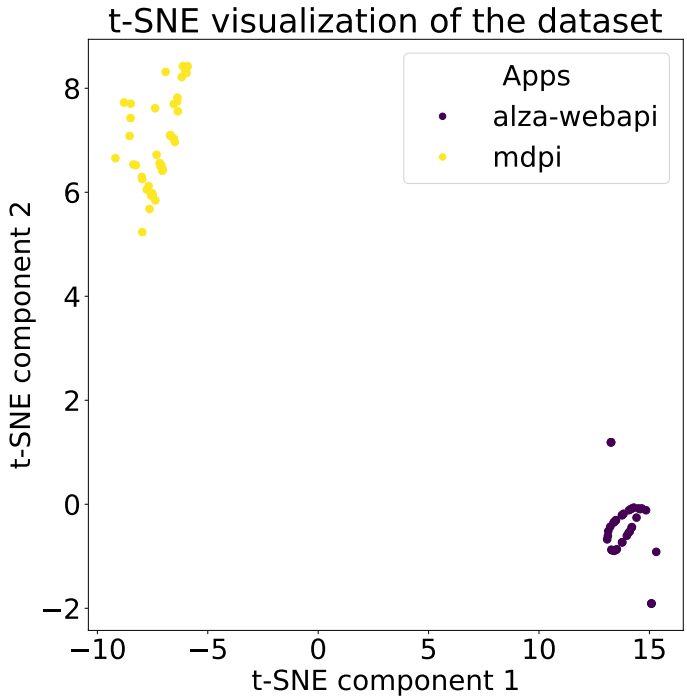

Figure 5.2: t-SNE projection of 2 apps true labels.

Figure 5.2 presents the t-SNE projection of the ground-truth labels into two dimensions. As observed, the applications form well-separated clusters, suggesting that the clustering algorithms should be able to distinguish between them without difficulty.

Applying **K-Means** with $K = 2$ resulted in perfect clustering, resulting in homogeneity and NMI scores of 1.0, indicating flawless alignment with the ground truth. Therefore, no additional evaluation metrics are necessary.

In addition, **HDBSCAN**, with predefined parameters, produced perfect clustering results similar to K-Means, achieving both homogeneity and an NMI score of 1.0.

Given that most algorithms successfully clustered the two applications, we proceed to the next step: clustering three applications.

5.3 Three apps

Now, we proceed to clustering three distinct applications. Once again, the indicator of dissimilarity — the *Bhattacharyya distance* — identified the same set of applications. In addition to **Alza Web API** and **MDPI**, the third application is **Instagram** (ID 34).

Figure 5.3 illustrates the t-SNE projection of the ground-truth labels in two dimensions. As shown, Instagram flows form two separate clusters: one is located near MDPI and the other near the Alza Web API. This fragmented location introduces complications, as clus-

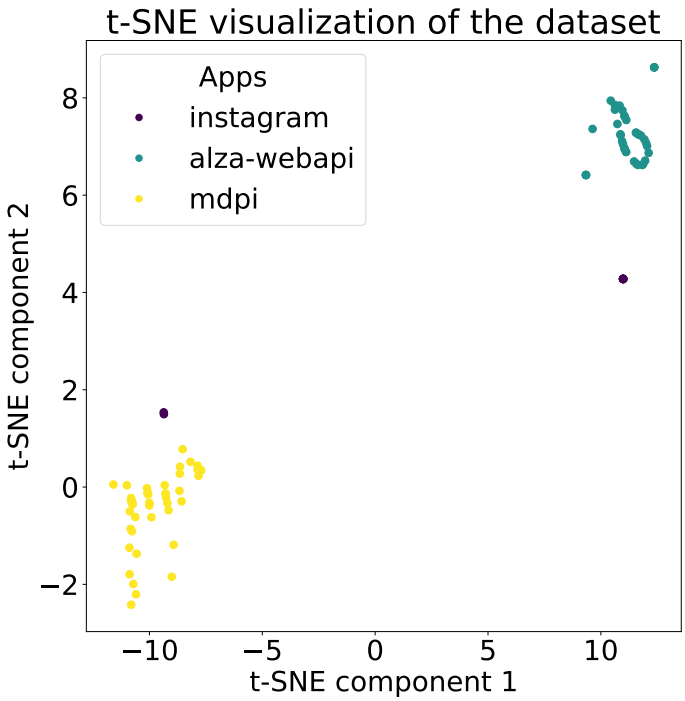

Figure 5.3: t-SNE projection of 3 apps true labels.

tering algorithms may struggle to accurately separate flows. The existence of two distinct Instagram clusters suggests either the existence of outliers or dual behavioral patterns in its network signature, potentially corresponding to flows with significantly different data volumes and durations.

Figure 5.4 presents the homogeneity and NMI scores for various **K-Means** runs using different values of K. These results confirm our concerns regarding the clustering of Instagram flows. The homogeneity score only reaches 1 when $K = 9$, which is notably high given that only three applications are being clustered. Meanwhile, the NMI score decreases with

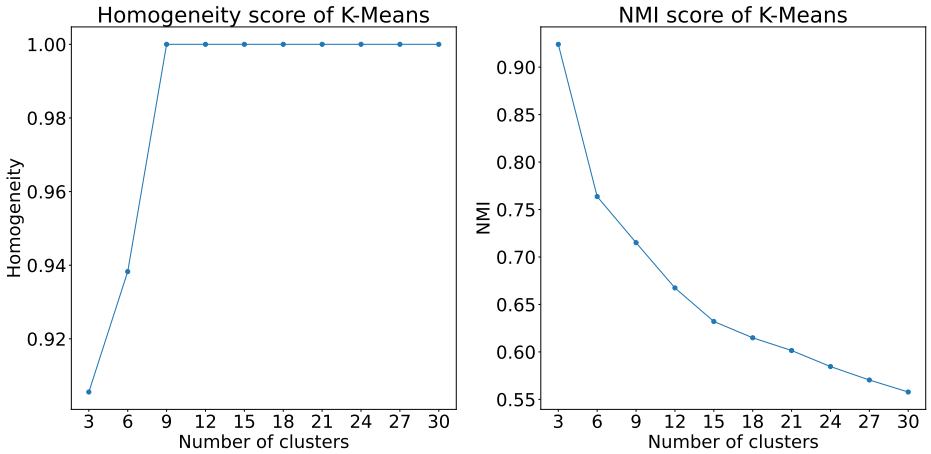

Figure 5.4: Homogeneity and NMI scores of K-Means of 3 apps.

larger K values, indicating that the initial clustering results are better in terms of mutual information.

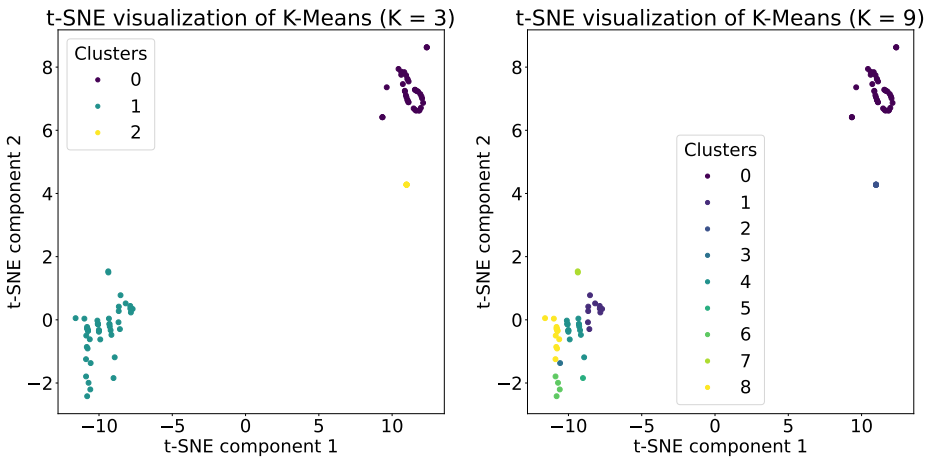

Figure 5.5: t-SNE projection of K-Means clustering of 3 apps with $K = 3$ and $K = 9$.

The clustering results for K-Means with $K = 3$ and $K = 9$ are visualized in Figure 5.5. For $K = 3$, the algorithm successfully separates MDPI and a subset of Instagram flows. However, it groups Alza Web API with the remaining Instagram flows, resulting in a clustering outcome that approximates the ground truth but does not achieve perfect separation. For $K = 9$, the algorithm achieves a homogeneity score of 1 by assigning the two distinct Instagram flow groups into separate clusters, while also sub-clustering flows from the MDPI. Notably, it still groups all the flows of the Alza Web API together.

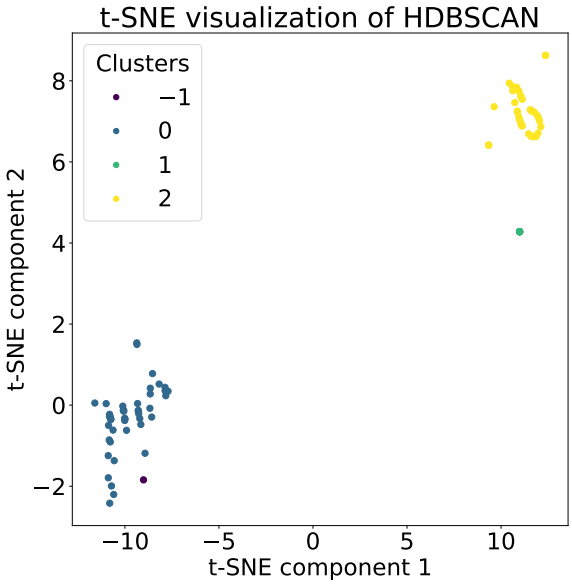

Figure 5.6: t-SNE projection of HDBSCAN clustering of 3 apps.

HDBSCAN achieves a homogeneity score of approximately 0.9 and an NMI score of around 0.92. Labeling one flow as noise does not significantly affect the homogeneity score. Figure 5.6 shows the t-SNE projection of the HDBSCAN clustering results. As is evident,

the algorithm correctly clusters the Alza Web API and one subset of Instagram flows, but merges the remaining Instagram flows with MDPI. Additionally, one MDPI flow is marked as noise.

5.4 Five apps

Up to this point, the clustering results have remained reasonably acceptable. In the next step, we increase the complexity by clustering five applications simultaneously.

At this stage, two more applications are added, besides **Alza Web API**, **MDPI** and **Instagram**: these are **YouTube** (ID 15) and **Google Autofill** (ID 23).

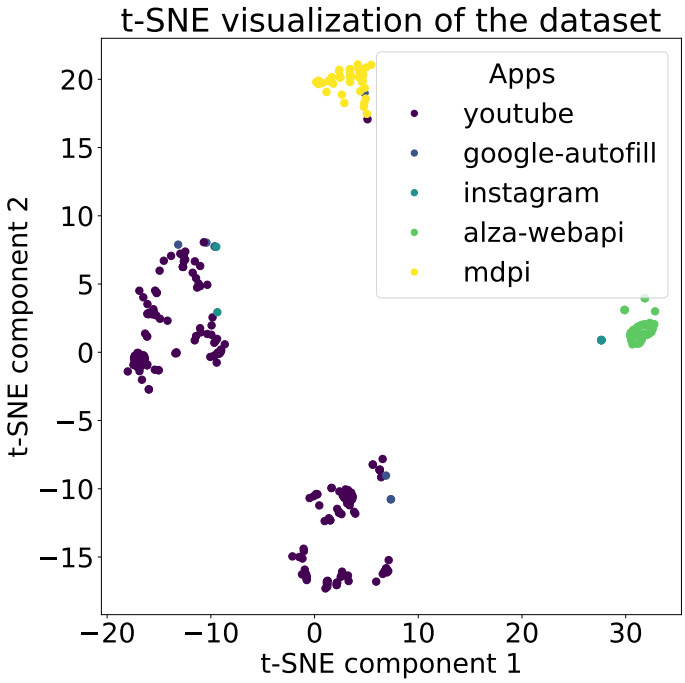

Figure 5.7: t-SNE projection of 5 apps true labels.

Figure 5.7 displays the t-SNE projection of the flows with ground-truth labels. As illustrated, the data becomes increasingly difficult to visually distinguish: for example, one subset of YouTube flows is intermixed with Google Autofill, while another overlaps with both Google Autofill and Instagram. The MDPI cluster remains relatively consistent but contains a few misclassified flows, including some from YouTube and Google Autofill.

We first apply the **K-Means** algorithm to this dataset. Figure 5.8 shows the homogeneity and NMI scores as the number of clusters K increases. As observed, the homogeneity score only reaches 1 when $K = 55$, which is eleven times the number of actual application labels. The NMI score, on the other hand, is highest for lower K values and decreases with increasing K, reflecting the metric's penalization of over-partitioning despite acceptable initial accuracy.

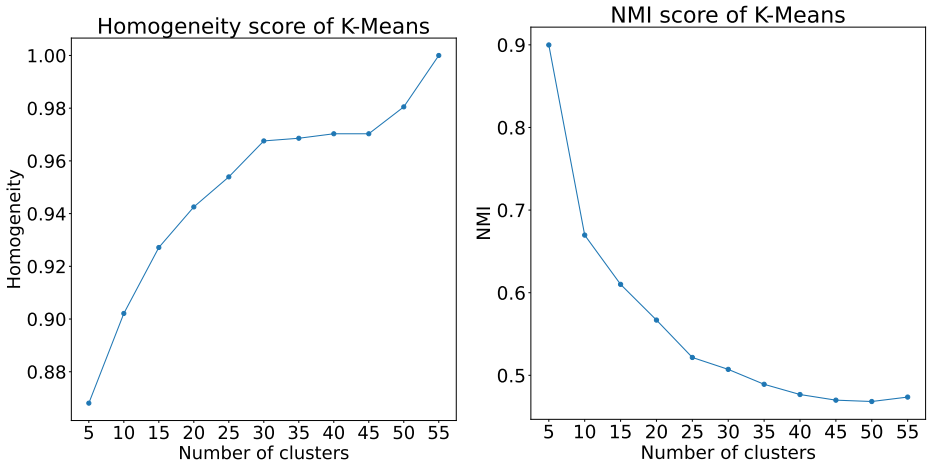

Figure 5.8: Homogeneity and NMI scores of K-Means of 5 apps.

Table 5.3: Contingency matrix of K-Means with 5 apps chosen.

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
YouTube	0	199	0	1	0
Google Autofill	3	4	0	1	0
Instagram	0	2	0	0	7
Alza Web API	0	0	39	0	0
MDPI	0	0	0	45	0

Table 5.3 presents the contingency matrix for K-Means clustering with $K = 5$. The results indicate that all Alza Web API flows are perfectly clustered, with no inclusion of other applications and no Alza Web API flows found in other clusters. The main YouTube cluster (Cluster 2) contains nearly all YouTube flows, with the exception of a single flow assigned to the MDPI cluster (Cluster 4). However, Cluster 2 also includes four Google Autofill and two Instagram flows. The Google Autofill flows are highly dispersed across three clusters. One of these, Cluster 1, consists only of Autofill flows, but it is not the most populated cluster for this application. The majority of Google Autofill flows are found in a cluster shared with YouTube flows, while one flow is assigned to the MDPI cluster. Most Instagram flows are grouped into Cluster 5 without contamination, though some are included in the YouTube cluster (Cluster 2). All MDPI flows are assigned to Cluster 4, but this cluster also includes one YouTube flow and one Google Autofill flow.

These results suggest that while K-Means achieves mostly correct groupings, it still has limitations in multi-class clustering with overlapping flows represented by simple features.

Next, we examine the results of clustering using **HDBSCAN**. Without preprocessing, the algorithm achieved a homogeneity score of 0.84 and a Normalized Mutual Information (NMI) score of 0.52. HDBSCAN identified 11 clusters in addition to a noise cluster. Notably, the algorithm labeled 65 of 301 flows as noise, which corresponds to approximately 20 % of the data. These noisy points belong mainly to YouTube and Google Autofill.

After filtering out the noise-labeled flows, the performance improved significantly, showing a homogeneity score of 0.98 and an NMI score of 0.6. Among the remaining flows, only one misclassification was observed: a single Google Autofill flow clustered with MDPI.

Despite the relatively high proportion of discarded flows and the detection of 11 distinct clusters, HDBSCAN demonstrated strong clustering quality in the retained data. Therefore, it may be considered a viable approach for application-level clustering in scenarios where a high degree of cluster homogeneity is prioritized over complete data utilization.

As observed, despite certain limitations, clustering algorithms demonstrate satisfactory performance on the subset of five applications, generating results that are both useful and meaningful. Consequently, in the following section, the experimental setup will be extended to include a significantly larger number of applications to evaluate whether the clustering results remain acceptable under increased complexity.

Chapter 6

Conclusion

This thesis explored the practicality of applying unsupervised clustering methods for flow-based classification of encrypted network traffic. While initial experiments with the full CESNET-QUIC22 dataset showed mediocre performance, with clustering metrics such as Homogeneity and Normalized Mutual Information (NMI) remaining relatively low, further experimentation on selected subsets of applications revealed better results, which reveals the main purpose of using such techniques.

The limitation of clustering the entire dataset exists due to its diversity and high class imbalance, making it difficult for unsupervised methods like K-Means or HDBSCAN with such a subset of features to produce clusters aligned properly with application labels. Nevertheless, when the scope was narrowed to smaller subsets of highly distinct applications, clustering performance improved significantly. For example, clustering just two or three applications resulted in near-perfect or fully accurate groupings, demonstrating that meaningful structure does exist in the selected flow-level features.

Among the algorithms tested, HDBSCAN showed particular promise due to its ability to identify noise points. While it performed poorly on the complete dataset, its performance on reduced subsets was much stronger, especially after excluding flows marked as noise. This characteristic makes HDBSCAN a suitable tool not only for unsupervised classification in labelless environments but also for anomaly detection, as flows designated as noise may correspond to rare outliers or suspicious activity.

In practical scenarios where the labeled data is not complete or available, clustering methods can provide basic knowledge by grouping similar traffic patterns and marking records as anomalous behavior. This matches practical needs such as network monitoring, service fingerprinting, or early detection of malicious traffic. However, the effectiveness of such techniques strongly depends on the homogeneity of the analyzed traffic subset and the availability of discriminative flow-level characteristics.

In summary, while clustering is not a perfect solution for flow classification at a large scale, it is still a viable and useful tool in targeted applications, especially when applied to small, well-defined subsets or in environments lacking ground-truth labels. Future work, based on unsupervised learning algorithms, could explore hybrid methods combining unsupervised clustering with lightweight supervised refinement or anomaly detection pipelines to extend the practicality and flexibility of this approach.

Bibliography

[1] Aitken, P.; Claise, B. and Trammell, B. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information* RFC 7011. RFC Editor, september 2013. Available at: https://doi.org/10.17487/RFC7011.

[2] Campello, R. J. G. B.; Moulavi, D. and Sander, J. Density-Based Clustering Based on Hierarchical Density Estimates. In: Pei, J.; Tseng, V. S.; Cao, L.; Motoda, H. and Xu, G., ed. *Advances in Knowledge Discovery and Data Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, p. 160–172. ISBN 978-3-642-37456-2.

[3] CESNET. *Ipfixprobe is a high-performance flow exporter.* 2021. Available at: https://github.com/CESNET/ipfixprobe.

[4] Claise, B. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information* RFC 5101. RFC Editor, january 2008. Available at: https://doi.org/10.17487/RFC5101.

[5] Erman, J.; Arlitt, M. and Mahanti, A. Traffic classification using clustering algorithms. In: *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*. New York, NY, USA: Association for Computing Machinery, 2006, p. 281–286. MineNet '06. ISBN 159593569X. Available at: https://doi.org/10.1145/1162678.1162679.

[6] Ester, M.; Kriegel, H.-P.; Sander, J. and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996, p. 226–231. KDD'96. Available at: https://dl.acm.org/doi/10.5555/3001460.3001507.

[7] Gijón, C.; Toril, M.; Solera, M.; Luna Ramírez, S. and Jiménez, L. R. Encrypted Traffic Classification Based on Unsupervised Learning in Cellular Radio Access Networks. *IEEE Access*, 2020, vol. 8, p. 167252–167263.

[8] Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. IEEE COMPUTER SOC, 2007, vol. 9, no. 3, p. 90–95.

[9] Iyengar, J. and Thomson, M. *QUIC: A UDP-Based Multiplexed and Secure Transport* RFC 9000. RFC Editor, may 2021. Available at: https://doi.org/10.17487/RFC9000.

[10] Luxemburk, J. and Hynek, K. DataZoo: Streamlining Traffic Classification Experiments. In: *Proceedings of the 2023 on Explainable and Safety Bounded,*

Fidelitous, Machine Learning for Networking. New York, NY, USA: Association for Computing Machinery, 2023, p. 3–7. SAFE '23. ISBN 9798400704499. Available at: https://doi.org/10.1145/3630050.3630176.

[11] Luxemburk, J.; Hynek, K.; Čejka, T.; Lukačovič, A. and Šiška, P. CESNET-QUIC22: A large one-month QUIC network traffic dataset from backbone lines. *Data in Brief*, 2023, vol. 46, p. 108888. ISSN 2352-3409. Available at: https://www.sciencedirect.com/science/article/pii/S2352340923000069.

[12] Manning, C. D.; Raghavan, P. and Schütze, H. *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008. 356-360 p. ISBN 0521865719.

[13] Nielsen, F. *Introduction to HPC with MPI for Data Science*. Springer International Publishing, 2016. Undergraduate Topics in Computer Science. ISBN 9783319219035. Available at: https://books.google.cz/books?id=eDiFCwAAQBAJ.

[14] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 2011, vol. 12, p. 2825–2830.

[15] Rosenberg, A. and Hirschberg, J. V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure. In: Eisner, J., ed. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, June 2007, p. 410–420. Available at: https://aclanthology.org/D07-1043/.

[16] Scikit Learn. *Clustering Guide*. 2025. Available at: https://scikit-learn.org/stable/modules/clustering.html.

[17] Skrypnyk, I. Irrelevant Features, Class Separability, and Complexity of Classification Problems. In: *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*. 2011, p. 998–1003.

[18] Tan, P.-N.; Steinbach, M.; Karpatne, A. and Kumar, V. *Introduction to Data Mining (2nd Edition)*. 2ndth ed. Pearson, 2018. ISBN 0133128903.

[19] Wang, J.; Yang, L.; Wu, J. and Abawajy, J. H. Clustering analysis for malicious network traffic. In: *2017 IEEE International Conference on Communications (ICC)*. 2017, p. 1–6.