Received ; | Revised ; | Accepted

DOI: xxx/xxxx

ORIGINAL ARTICLE

Computation of Persistent Homology on Streaming Data using Topological Data Summaries

Anindya Moitra* | Nicholas O. Malott | Philip A. Wilsey

Dept. of Electrical and Computer Engineering, University of Cincinnati, Cincinnati, Ohio, USA

Correspondence
*Anindya Moitra. Email: id.anindya@gmail.com

Summary

Persistent homology is a computationally intensive and yet extremely powerful tool for Topological Data Analysis. Applying the tool on potentially infinite sequence of data objects is a challenging task. For this reason, persistent homology and data stream mining have long been two important but disjoint areas of data science. The first computational model, that was recently introduced to bridge the gap between the two areas, is useful for detecting steady or gradual changes in data streams, such as certain genomic modifications during the evolution of species. However, that model is not suitable for applications that encounter abrupt changes of extremely short duration. This paper presents another model for computing persistent homology on streaming data that addresses the shortcoming of the previous work. The model is validated on the important real-world application of network anomaly detection. It is shown that in addition to detecting the occurrence of anomalies or attacks in computer networks, the proposed model is able to visually identify several types of traffic. Moreover, the model can accurately detect abrupt changes of extremely short as well as longer duration in the network traffic. These capabilities are not achievable by the previous model or by traditional data mining techniques.

KEYWORDS:
Persistent homology, Topological Data Analysis, data stream mining, discovery science, data mining, network anomaly detection.

1 | INTRODUCTION

Persistent homology is a powerful tool for Topological Data Analysis that is based on a robust theoretical foundation and is capable of discovering insights from data that are not discernible by traditional data mining methodologies [1,2,3,4,5,6]. However, persistent homology incurs an exponential space complexity due to its use of a combinatorial object, called the *complex* [7], as the foundational data structure used in its computation. The exponential memory requirement of persistent homology (due to the size of the complex associated to the input set of data points) poses a significant challenge for its application to data stream mining.

Data streams are generated by applications such as stock market and other financial transactions, computer network traffic, meteorological analysis, audio and video streaming services, satellite imagery, sensor networks, and so on [8]. Since a data stream is a potentially unbounded sequence of continuously arriving data objects, the entire stream cannot be stored in the memory typically available to a computer. As random access to the data is unavailable, algorithms dealing with data streams must make only one (or very few) pass(es) through the data [9]. Moreover, the data generation process can be non-stationary, resulting in a

data stream that *evolves* over time. A change in the data generation process or in its underlying probability distribution is called *concept drift*. Such unique challenges of data stream mining coupled with the high computational cost of persistent homology are the main reasons why they have long been two disjoint areas of data science.

The first computational model or framework for applying persistent homology to streaming data was recently introduced[10] where a bounded summary of an unbounded data stream was maintained by utilizing the concept of *feature vector* of microclusters[8]. A feature vector is a data structure that consists of statistics related to the weighted sum of a small group of data points called the *microcluster*. It was shown that the data summary maintained by the feature vector of microclusters captures the topological structure of the stream and can detect the concept drifts the stream may exhibit with time.

While the computational model used in the previous work[10] is capable of identifying steady or gradual concept drifts, it is not suitable for detecting changes that occur for very short durations. This is because the size and the update sensitivity of the data summarization model[10] are not precisely configurable by the user or the application. Thus, the model is not suitable for applications that can involve 'sharp' temporal changes (or, 'spikes') in a metric that is being monitored. Examples of such applications include monitoring of sensor and computer networks, manufacturing equipment, and surveillance systems.

In order to address these shortcomings of the previous work and to develop a model that is suitable for a wide variety of stream applications, this paper proposes a computational framework, henceforth referred to as the *sliding-window model*. The sliding-window model is based on a new type of data summarization model that maintains a summary of the stream by a dynamically updated complex. Consistent with the standard computational paradigm for processing data streams, the model comprises two components, namely: (i) online and (ii) offline.

The online component maintains a summary of the stream that is designed to preserve the topological structure of the streaming data. The summary of the stream maintained by the sliding-window model is the *complex*. The complex is constructed on a set of *representatives* of the most recent data points from the stream. A representative is a point that has properties similar to a large number of other points in the stream. As new points arrive from the stream, the representatives and the complex built on them are continuously updated based on certain conditions. When a new data point qualifies to be inserted into the set of representatives, new simplices are added to the complex. At the same time, the *least relevant* representative and its corresponding simplices are deleted from the set of representatives and the complex. Thus, the complex, that represents a topological summary of the stream, is incrementally maintained during the online component of the sliding-window model.

The offline component comprises the computation of *persistence intervals* (see Section 3), the final output of persistent homology, from the complex at fixed time intervals. The persistence intervals are displayed in one of the standard output formats such as the barcodes or persistence diagrams (Figure 1). The output from the sliding-window model can also be represented in terms of quantitative metrics, such as the Wasserstein distance[11] between pairs of consecutive persistence diagrams or between a reference persistence diagram and subsequent diagrams. By monitoring the sequence of barcodes, persistence diagrams, or distance values in real-time, one can visualize and detect any changes with the progress of the stream.

The sliding-window model is applied to the detection of anomalous TCP connections (or, *attacks*) in large streams of network traffic. It is shown that by monitoring the Wasserstein distances between pairs of persistence diagrams, one can identify the occurrence of short *and* long anomalies in computer networks. Moreover, it is possible to distinguish several types of traffic from one another by visualizing the topological structures of those traffic through barcodes or persistence diagrams. These capabilities are not achievable by the model described in[10] or by traditional data mining methods.

The remainder of this paper is organized as follows. Section 2 briefly reviews the existing literature on the application of persistent homology to streaming data. Section 3 provides an overview of the basic concepts and terminology related to persistent homology. Section 4 describes the data structures and algorithms that constitute the sliding-window model. Section 5 reviews the input parameters of the sliding-window model. Section 6 provides a detailed evaluation of the sliding-window model on the detection of anomalous traffic in computer networks. Finally, Section 7 concludes the paper and provides suggestions for future work that may lead to advancements of this research.

2 | RELATED WORK

To the best of our knowledge, there is only one existing computational framework for applying persistent homology to streaming data[10]. It is based on the concept of feature vector of microclusters[8]. As indicated in Section 1, the size or number of microclusters in the existing model[10] can not be directly and precisely configured by the user or the application. As a result, it is not straightforward to summarize a stream with a small number of microclusters and accurately detect any 'spikes' of changes

in the stream. The technical details of this shortcoming of the previous work are discussed in Section 4. The sliding-window model proposed in this paper is suitable for detecting both gradual and sharp changes in a stream.

It is worth mentioning that the challenges of data stream mining received adequate attention over the last two decades. Several approaches were introduced to address those challenges by adapting the batch processing algorithms to data streams for both supervised and unsupervised learning[8,12]. In addition, there are several studies on time series data (which are akin to streaming data) using persistent homology[13,14,15,16] that "batch process" either the entire time series or predefined partitions thereof. In other words, existing studies on time series data using persistent homology generally assume the entire data are available *a priori*. However, there is no existing work that accomplishes the following at the same time:

- Applies persistent homology on *potentially infinite* streaming data, where at any given time we cannot store all the previous data objects and do not have any information on future data objects;
- Detects the occurrences of concept drifts and also visually identifies the various types of segments or clusters that cause the concept drifts in a data stream; and
- Detects both gradual changes and sharp 'spikes' in a stream.

Being able to achieve the above objectives in a single computational model is the primary contribution of the current work.

3 | BACKGROUND

This section briefly explains the introductory ideas of persistent homology; an intuitive visual presentation of the basic concepts is available at[17]. The reader may refer to[11,18,19] for a formal exposition of the subject.

Homology is a way of counting the topological features of a space, such as connected components, holes or loops, voids, and their higher dimensional analogs[11]. *Persistent homology* extends homology by computing the lifespans of topological features through increasing spatial resolutions.

3.1 | Simplicial Complex

A set of data points equipped with a distance function is called a *point cloud*, which is assumed to be sampled from an underlying topological space $\mathbb{S}$. For most practical applications, the probability distribution of $\mathbb{S}$ is unknown. Computing the homology of such arbitrary topological spaces is difficult. To overcome this obstacle, the topology of $\mathbb{S}$ is approximated by constructing a structure, called a *complex*, on the given set of data points that homology can be computed[20]. Simplicial, cubical, and CW complexes[20,21] are some of the commonly used complexes. Since the simplicial complex is the most widely used with a richer theoretical foundation than others[20], persistent homology computed from simplicial complexes is examined in this paper.

A simplicial complex K is a generalization of triangular geometric structures. In particular, a simplicial complex is a set of points, edges, and triangular objects such triangles, tetrahedrons and so on. K comprises all possible subsets that can be constructed from the distinct points in K. More precisely, a simplicial complex is a set K of finite sets such that if $\sigma \in K$ and $\tau \subseteq \sigma$, then $\tau \in K$[7]. In other words, every subset of the constituent sets of K is also a subset of K. For example, a geometric $\bigtriangleup abc$, formed by the points a, b and c, constitutes a simplicial complex $K = \left\{ \emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{b,c\}, \{c,a\}, \{a,b,c\} \right\}$.

3.2 | Vietoris–Rips Complex

One of the most widely used approaches to building a simplicial complex on a point cloud is the construction of a *Vietoris–Rips* complex[11] (also called a *Rips* complex). If the simplicial complex is built by connecting any two points within a distance $\varepsilon \geq 0$ by an edge, a Vietoris–Rips complex V is obtained. In other words, in a Vietoris–Rips complex any two points within a distance $\varepsilon \geq 0$ are connected by an edge. Formally, a Vietoris–Rips complex $V(P, \varepsilon)$ at scale ε is defined as:

$$V(P,\varepsilon) = \left\{ \sigma \subset P \mid dist\,(\mathbf{x},\mathbf{y}) \leq \varepsilon \text{ for all } \mathbf{x},\mathbf{y} \in \sigma \right\}.$$

The concepts, algorithms, and implementations described in the remainder of this paper are based on the Vietoris–Rips complex, the most widely used complex due to its fast construction on higher dimensional data[7]. However, the computation of persistent homology on streaming data using the sliding-window model described in this paper is not tied to the Vietoris–Rips complex and is generalizable to any type of complex.

3.3 | Filtration

A subset $K_i \subseteq K$ is called a *subcomplex* if K_i itself is a simplicial complex. A *filtration* of a complex K is a sequence of nested subcomplexes $\varnothing = K_0 \subseteq K_1 \subseteq K_2 \subseteq \ldots \subseteq K_n = K$. A complex with a filtration is called a *filtered complex*.

3.4 | Homology Groups

Each topological feature of a simplicial complex is assumed to have a *dimension p*. For example, each connected component has a dimension $p = 0$, each loop or cycle has $p = 1$, each void has $p = 2$, and so on. The maximal dimension that the topological features are examined is a required input for the computation of persistent homology. The set of p-dimensional features forms a *group*, called the p-th *homology group* H_p. The *rank* of H_p is the p-th *Betti number* β_p that represents the count of the p-dimensional topological features. For example, β_0, β_1 and β_2 denote, respectively, the number of connected components, the number of holes, and the number of voids in a simplicial complex.

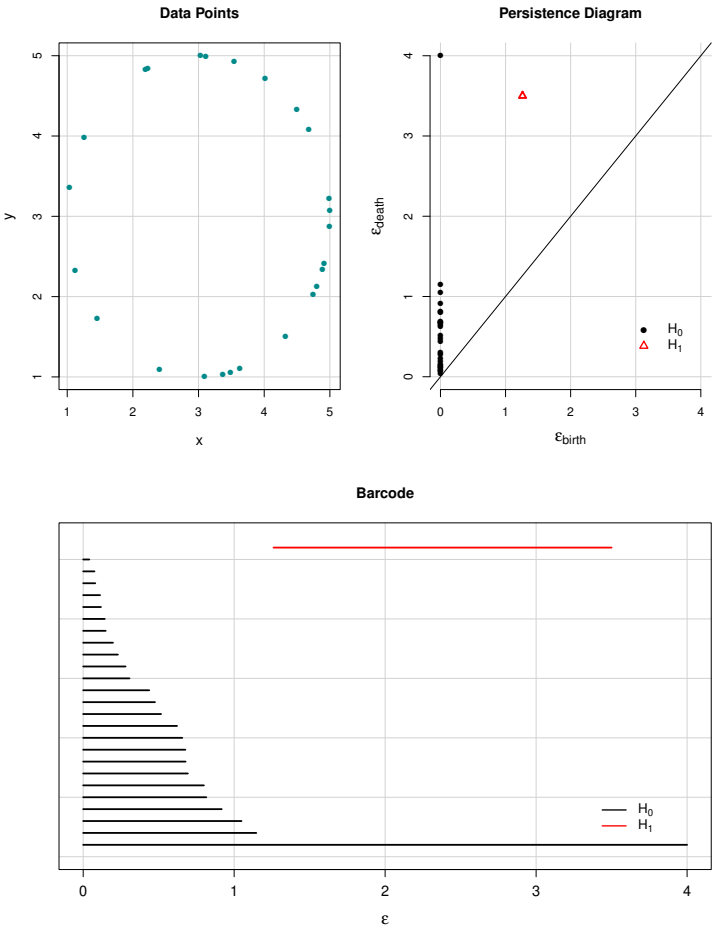

FIGURE 1 Illustration of the input and output of persistent homology

3.5 | Persistent Homology

The homology of a simplicial complex, through Betti numbers, provides the count of p-dimensional topological features for each dimension p specified by the user. Persistent homology is a mechanism of computing the homology on a sequence of simplicial complexes. In particular, persistent homology computes the homology of each subcomplex in the filtration of a simplicial complex K. Since the subcomplexes in the filtration represent a sequence of increasingly connected and nested topological spaces, constructing those subcomplexes can be considered as the process of increasing the spatial resolution of the underlying space $\mathbb{S}$. By computing the homology of each subcomplex in the filtration, persistent homology tracks the lifespan of each topological feature of $\mathbb{S}$ as it appears and subsequently disappears while the space is increasingly 'magnified'.

For most practical applications, the input for persistent homology is a set of data points $X = \{x_1, x_2, x_3, \ldots\}$, $x_i \in \mathbb{R}^d$. The computation of persistent homology begins by constructing a simplicial complex on X. If two different Rips complexes, V_1 and V_2, are constructed on the same set of points with different scale parameters, ε_1 and ε_2, respectively, then for $\varepsilon_1 \leq \varepsilon_2$, we have $V_1 \subseteq V_2$. Therefore, constructing a sequence of Rips complexes with increasing values of the scale parameter ε generates a filtration. This property of the complex is central to the idea of computing persistent homology with increasing spatial resolutions.

In practice, instead of building multiple simplicial complexes by varying the scale parameter ε between 0 and some user specified maximum ε_{max}, only one maximal Vietoris–Rips complex K is constructed at $\varepsilon = \varepsilon_{max}$. Then, a nested sequence of subcomplexes, or the filtration of K, is extracted from the complex K at certain values of ε. In order to do so, every simplex σ

of K is defined to have a weight ω that is the maximum of the lengths of all the edges in σ. A 0-simplex $\{\mathbf{x}\}$ has $\omega = 0$, and a 1-simplex $\{\mathbf{x}, \mathbf{y}\}$ has $\omega = dist(\mathbf{x}, \mathbf{y})$, the distance between the points $\mathbf{x}$ and $\mathbf{y}$. The weight ω of a simplex σ can be interpreted as the minimum ε at which σ is *born* in the Vietoris–Rips complex K. Thus, by sorting the simplices of K by their weights, a nested sequence of simplicial complexes is obtained, that would otherwise result from varying ε in $[0, \varepsilon_{max}]$. In other words, by assigning weights to the simplices of K and sorting them by their weights, the filtration of K is extracted. The simplicial complex K is then called a *weight-filtered complex*[7].

Each subcomplex in the filtration is associated to a distinct value of ε. Since the topological structure of each subcomplex is usually different from those of other complexes in the filtration, it is said that the topology of a simplicial complex changes with the scale parameter ε. For example, at $\varepsilon = 0$, the simplicial complex is a set of disconnected points. As ε increases, the points start becoming connected to one another by edges, and the corresponding simplicial complexes contain sets of points, edges, triangular faces, tetrahedrons, and so on. As additional points are connected with increasing ε: the connected components become longer; existing connected components are merged into one another; holes and voids appear (or, *are born*) and eventually get filled (or, *die*). Persistent homology provides a mechanism to track these changes in the topological properties of the simplicial complex as ε increases from 0 to the user-specified threshold ε_{max}. Persistent homology computes the birth and death times of the topological features, such as connected components, holes, and voids, as they appear and disappear with increasing ε. The output of persistent homology is a set of pairs of real numbers $(\varepsilon_{birth}, \varepsilon_{death})$. A topological feature is born at $\varepsilon = \varepsilon_{birth}$, and dies at $\varepsilon = \varepsilon_{death}$. The difference between the death and birth times is called the *lifespan*, or *persistence* of a topological feature. The lifespans of significant topological features are much longer than those of trivial or insignificant features[22,19]. Thus, one can segregate important topological features from the trivial ones by the lifespans computed by persistent homology.

TABLE 1 Persistence intervals or lifespans of topological features shown as pairs of real numbers

Dimension p	Birth time ε_{birth}	Death time ε_{death}
0	0.0000	4.0000
0	0.0000	1.1468
0	0.0000	1.0488
0	0.0000	0.9176
0	0.0000	0.8156
0	0.0000	0.7996
0	0.0000	0.6932
0	0.0000	0.6783
0	0.0000	0.6770
0	0.0000	0.6564
0	0.0000	0.6217
0	0.0000	0.5158
0	0.0000	0.4756
0	0.0000	0.4374
0	0.0000	0.3067
0	0.0000	0.2808
0	0.0000	0.2301
0	0.0000	0.1973
0	0.0000	0.1491
0	0.0000	0.1430
0	0.0000	0.1178
0	0.0000	0.1118
0	0.0000	0.0799
0	0.0000	0.0749
0	0.0000	0.0403
1	1.2609	3.5001

The set of lifespans $(\varepsilon_{birth}, \varepsilon_{death})$ is displayed in a variety of visual formats such as the barcodes, persistence diagrams, or persistence landscapes[23]. In a barcode, the lifespans or persistence intervals are shown as bars or lines, where the length of a bar or line represents the lifespan of the corresponding topological feature (bottom graphic of Figure 1). A persistence diagram is a 2-dimensional scatter plot of the set of lifespans, with the birth and death times plotted along the horizontal and vertical axes, respectively (top right graphic of Figure 1). Since significant features have longer lifespans, their death times are substantially greater than birth times. Therefore, the points that represent significant features lie far away from the 45° line that passes through the origin of the persistence diagram. Noise points, on the other hand, reside close to or on the 45° line.

A demonstration of the output of persistent homology computed on a point cloud of 25 data points sampled from a circle in a plane is shown in Figure 1 and Table 1. Since the data points form a 1-dimensional loop, persistence intervals are computed for dimensions $p = 0$ and $p = 1$, and are shown as pairs of real numbers in Table 1. In Figure 1, those persistence intervals are displayed as the barcode and as persistence diagram. Since the death time ε_{death} of the 1-dimensional loop is substantially greater than its birth time ε_{birth}, the loop is a significant topological feature in the given point cloud. This topological feature is displayed as a long red line in the barcode, and as a red triangle lying far away from the diagonal in the persistence diagram.

4 | SLIDING-WINDOW MODEL: DATA SUMMARIZATION BY COMPLEX

4.1 | Shortcomings of the Previous Work

In the previously developed microcluster-based model[10], the data summarization is performed using feature vectors of the microclusters[24,25,26,8]. There are two input parameters, r_{max} and w_{min}, that control the size and update frequency of the summary of the stream. These two input parameters are discussed below.

4.1.1 | Maximum Radius of the Microcluster r_{max}

Each microcluster is assumed to have a maximum radius r_{max} that determines whether a new data point $\mathbf{x}$ from the stream can be added to its nearest microcluster c. If $\mathbf{x}$ cannot be added to c, a new microcluster is created with only $\mathbf{x}$ in it. A larger value of r_{max} increases the probability of $\mathbf{x}$ being added to c. In other words, r_{max} controls the creation of new microclusters. Thus, it is one of the parameters that impacts the number of microclusters in the memory. A large value of r_{max} results in fewer microclusters, each of which is large in size. On the other hand, a small value of r_{max} results in a large number of small-sized microclusters.

4.1.2 | Minimum Weight of the Microcluster w_{min}

Each microcluster is assumed to have a minimum threshold of weight w_{min}. If the weight of a microcluster falls below w_{min}, it is considered outdated and is deleted from memory. The parameter w_{min} affects the number of microclusters stored in the memory. A larger value of w_{min} leads to fewer microclusters, and vice versa. For a fixed value of r_{max}, the number of microclusters controlled by w_{min} (together with a decay parameter λ) impacts the length of 'history' (or, the number of topological features) retained in the memory. A higher value of w_{min} shortens the history, whereas a smaller value of w_{min} expands the history.

For rapidly evolving streams, one should choose a small r_{max} and large w_{min}, which facilitate faster creation and removal of the microclusters. The opposites are preferred for streams with more gradual concept drifts. The microcluster-based framework[10] only provides *indirect* control over the size and the update frequency of the data summary through the input parameters r_{max} and w_{min}. As a result, it is difficult to configure the summary to accurately represent a small number of data objects from the stream. This, in turn, renders the microcluster model inaccurate for applications that can involve 'sharp' temporal changes (or, 'spikes') in the metric that is being monitored.

The choice of values for r_{max} and w_{min} depends on the problem and the nature of the stream, and so does the choice of values for the sliding-window model. However, the sliding-window model provides *direct* control over the size and the update frequency of the data summary (as will be seen in Sections 4.3.2, 6 and 5) and is highly accurate for detecting 'spikes' in the stream. This is the primary drawback of the microcluster-based framework that the model of this paper addresses.

4.2 | Overview of the Sliding-Window Model

The sliding-window model advances the online component of the microcluster-based framework[10] by incorporating the construction of the complex within the data summarization phase. Thus, the data summary maintained by the online component of the sliding-window model is the *complex* built on a set of data points that represent the current 'state' of the stream. The offline component in the sliding-window model then consists only of the computation of persistence intervals from the complex maintained during the online component. The two principal components of the sliding-window model are summarized below.

1. *Online:* Continuous summarization of the data stream into a complex of bounded size.
2. *Offline:* Computation of persistence intervals from the complex.

The *online* component of the sliding-window model maintains a set of representatives (henceforth referred to as the *window*) $\mathcal{W}$ of the most recent data points from the stream. The representative points in the window are ordered according to when they were inserted into the window. The maximum size of the window, m, is specified by the user in terms of the maximum number of representative data points the window may contain. When a new data point $\mathbf{x}$ arrives, several conditions are checked to determine if $\mathbf{x}$ should be inserted into $\mathcal{W}$. If $\mathbf{x}$ is inserted into $\mathcal{W}$, and if as a result of this insertion the size of $\mathcal{W}$ exceeds that specified by the user, the *least relevant representative* is removed from $\mathcal{W}$. A simplicial complex K, built on the current set of data points in $\mathcal{W}$, is incrementally maintained during the online component. Whenever a data point is inserted into and/or

removed from $\mathcal{W}$, the corresponding simplices are added to and/or deleted from K. Thus, the simplicial complex K is the data summary maintained by the online component of the sliding-window model.

From the above description, the online component of the sliding-window model can be divided into two stages, namely:

1. the incremental maintenance of the window, and
2. the incremental maintenance of the complex.

These two stages are described separately in the subsections below.

The *offline* component involves the computation of persistence intervals from the complex K at fixed intervals. The persistence intervals are displayed in one of the standard output formats such as the barcodes or persistence diagrams. Thus, by monitoring the sequence of barcodes or persistence diagrams in real-time, one can visualize the current state of the stream and detect any changes. The changes in the output of persistent homology with the progress of the stream can also be represented in terms of distance values (such as the Wasserstein distance) between pairs of consecutive persistence diagrams or between a reference persistence diagram and subsequent diagrams. The sliding-window model is suitable for developing change detection mechanisms in continuously evolving data streams. Examples of such applications include real-time surveillance systems, continuous and real-time monitoring of network traffic, telecommunication systems, sensor networks, and other dynamic environments.

In Section 6, the sliding-window model is applied to detect anomalous TCP connections in large streams of network traffic. It is shown that by monitoring the Wasserstein distances between pairs of persistence diagrams, one can identify the occurrence of anomalies or attacks in computer networks. Moreover, it is possible to distinguish several types of traffic from one another by visualizing the topological structures of the traffic through barcodes or persistence diagrams.

4.3 | Incremental Maintenance of the Window

This section describes the first stage of the online component that involves maintaining a set of representatives of the most recent data points from the stream. The second stage in the online component, that incrementally updates a complex constructed on the representatives, is explained in Section 4.4.

4.3.1 | Basic Principles

In order to enable faster processing of the stream, the sliding-window model aims to minimize the number of addition and deletion operations on the complex K. Addition and deletion of simplices are associated with insertion and removal of data points in a window $\mathcal{W}$. Hence, in order to minimize the number of updates to the complex K, the window employs a selection mechanism that only accepts certain new data points from the stream for insertion into $\mathcal{W}$ if specific conditions are met. In other words, the window uses a set of filtering criteria that attempts to reject as many new points as possible under certain circumstances.

The data points in the window are partitioned into groups such that the points within the same group or partition are *similar*[1] to one another while points in the different partitions are distant from one another. The partition membership of each representative data point is marked by a partition label associated to the point.

The basic design principle of the sliding-window model is that it applies a light-weight clustering mechanism (Section 4.3.2) to the incoming data points that are accepted for insertion into the window. In other words, the window maintains a set of partitions or clusters of data points within it. When the window contains only one partition $\mathcal{P}_1$ (Figure 2.a) during a steady state of the stream, or in the absence of a concept drift, it rejects any incoming data point that is similar to the existing points

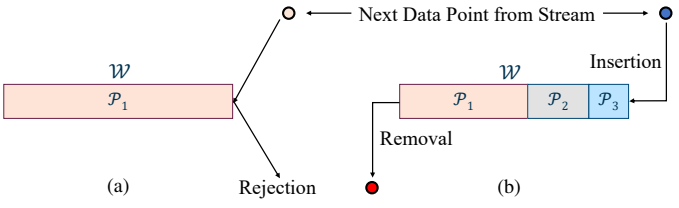

FIGURE 2 The basic design principle of the window

[1]The similarity is commonly measured in terms of some distance function, such as the Euclidean distance.

in the window. On the other hand, when the window contains multiple partitions (Figure 2.b) due to a change or concept drift in the stream, it accepts all incoming data points until the window has only one partition (*i.e.*, until the stream reaches a steady state again).

One of the differences between the maintenance of the window and that of the microclusters [10] is that the set of microclusters is updated with the arrival of every new data point from the stream. In contrast the window is updated only in the event of a concept drift (*i.e.*, when the window contains multiple partitions or when a new data point is significantly dissimilar to the existing ones in the window). Another difference is that each microcluster is the mean or cluster center of a small group of similar data points, whereas each partition in the window is a cluster of actual data points. The remainder of this section details the conditions and procedures for the incremental maintenance of the window.

4.3.2 | Detailed Description

In the sliding-window model, the topological properties of the complex K constructed on the data points in $\mathcal{W}$ are monitored with the progress of the stream. Any changes to the underlying distribution of the stream (called the *concept drift*) triggers changes in the topological structure of K. To help visualize the current state of the stream and promptly detect any concept drift, the sliding-window model aims to maintain only one partition in the window as long as possible. If there is more than one partition in the window, it admits every new data point and removes the *least relevant* existing point until the window has only one partition. Once the window consists of only one partition, it begins rejecting any new data point that is *similar* to the existing set of representatives in the window.

At all times, the window maintains the average nearest neighbor distance d_{avg} of each partition, which is the mean of the nearest neighbor distances of all points within the partition. When a new point $\mathbf{x}$ arrives, the distance δ from $\mathbf{x}$ to its nearest data point $\mathbf{y}$ in the window is computed. If the window consists of only one partition and if $\delta/d_{avg} > f$ (where f is a user specified input parameter), there is an indication that the new data point $\mathbf{x}$ has properties not represented by the set of points in the existing partition. Therefore, $\mathbf{x}$ qualifies to be inserted into the window. In this case, a new partition is created with only $\mathbf{x}$ in it. The average nearest neighbor distance of the new partition is set to -1 to indicate that d_{avg} for the new partition is undefined. The smaller the value of f, the higher the probability of $\mathbf{x}$ being added to the window. In other words, f represents the update sensitivity (or equivalently, the sliding speed) of the window that has only one partition in it, with smaller values of f corresponding to higher frequency of updates (or, higher speed of sliding). If the addition of $\mathbf{x}$ results in a size of the window that exceeds its maximum size, the first added (*i.e.*, the oldest) representative point is removed from the window. On the other hand, if $\delta/d_{avg} \leq f$, and if the window consists of only one partition, the new data point $\mathbf{x}$ is discarded, implying $\mathbf{x}$ already has a representative in the window.

If the window $\mathcal{W}$ has more than one existing partition, the partition $\mathcal{P}$ nearest to the new data point $\mathbf{x}$ is determined. The partition nearest to $\mathbf{x}$ is defined as the partition that $\mathbf{y}$ belongs to, where $\mathbf{y} \in \mathcal{W}$ is the point nearest to $\mathbf{x}$ in the window. Let δ be the distance between $\mathbf{x}$ and $\mathbf{y}$, and d_{avg} be the average nearest neighbor distance of the partition $\mathcal{P}$ that is nearest to $\mathbf{x}$. If $\delta/d_{avg} \leq f$, the new point $\mathbf{x}$ is assigned to $\mathcal{P}$. Otherwise, if $\delta/d_{avg} > f$, a new partition is created with only $\mathbf{x}$ in it. As before, the average nearest neighbor distance of this new partition is set to -1. The insertion of $\mathbf{x}$ is accompanied by the removal of the *least relevant point* from $\mathcal{W}$. Below are the sequence of steps used to determine the least relevant data point in the window.

1. Check if the window has any *outdated* partition. A partition is considered outdated if no new point was assigned to it during the last n_o insertions into the window. For the experiments described in Section 6, n_o is set to 10; in general, however, this can be a user controlled value.

2. If the window has only one outdated partition, remove the oldest data point from that partition. If the window has more than one outdated partition, find the smallest one (*i.e.*, the one with the fewest data points in it) among them, and remove the oldest point from the smallest outdated partition. The reason behind using the *size* of the outdated partitions in the selection of the point to be removed is that a smaller outdated partition is more probable to consist of outliers than a larger partition.

3. If the window has no outdated partition, the partition that the new point $\mathbf{x}$ was assigned to is taken under consideration. If $\mathbf{x}$ was assigned a new partition (*i.e.*, if a new partition was created with only $\mathbf{x}$ in it), the oldest point in the window is removed. If $\mathbf{x}$ was assigned to one of the existing partitions $\mathcal{P}$, the oldest point from any other partition $\mathcal{P}_o \neq \mathcal{P}$ is removed.

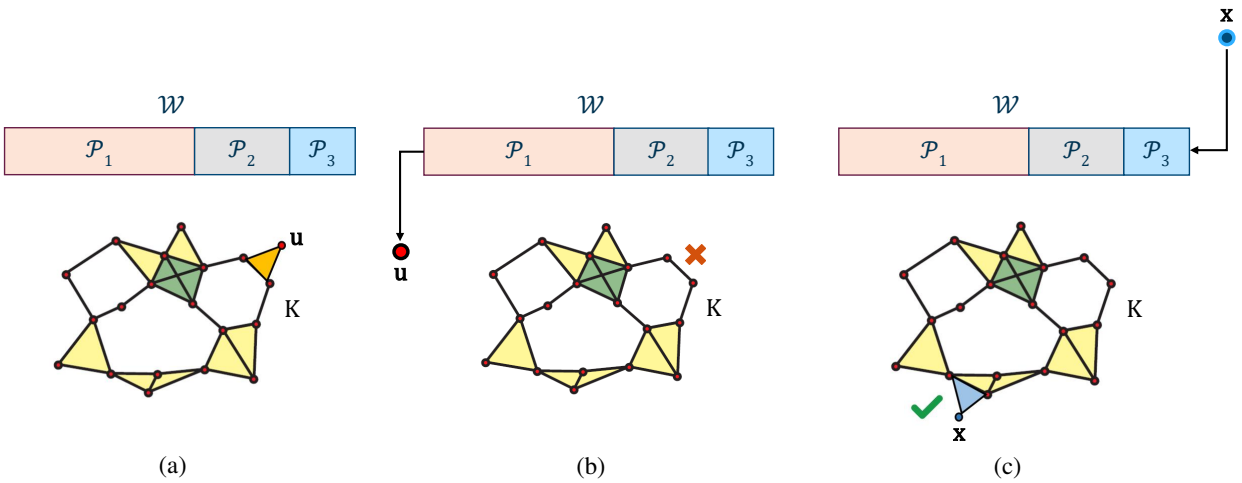

FIGURE 3 The incremental maintenance of the complex

At the beginning of the stream, the sliding-window model is initialized by inserting the first m data points into $\mathcal{W}$, and by adding the corresponding simplices to K. During the initialization phase, no data points or simplices are removed from $\mathcal{W}$ or K, respectively. The window $\mathcal{W}$ is implemented by a container in which the data points are ordered according to when they were inserted into $\mathcal{W}$. The function INSERT($\mathbf{x}$) inserts a point $\mathbf{x}$ into the container. The function REMOVE($\mathbf{u}$) removes and returns a point $\mathbf{u}$ from the container.

Figure 3 illustrates the incremental maintenance of the window and the associated complex. Figure 3.a shows a typical instance of the window $\mathcal{W}$ and the complex K during the progress of the stream. It is assumed that $\mathcal{W}$ contains m data points in this instance. The next data point from the stream, $\mathbf{x}$, needs to be inserted into the window. As $\mathcal{W}$ is already at its maximum capacity, the least relevant representative $\mathbf{u}$ is removed from $\mathcal{W}$, and the simplices corresponding to $\mathbf{u}$ are deleted from K (Figure 3.b). After that, $\mathbf{x}$ is inserted into the window $\mathcal{W}$, and the simplices corresponding to $\mathbf{x}$ are added to the complex K (Figure 3.c).

The computation of persistent homology on streaming data using the sliding-window model is summarized in Algorithm 1. In the online component of Algorithm 1, the incremental maintenance of the window is accompanied by additions and deletions of simplices on the simplicial complex K. When a new data point $\mathbf{x}$ is inserted into the window $\mathcal{W}$, the algorithm ADDSIMPLICES($\mathcal{W}, K, \mathbf{x}, p, \epsilon$) (Algorithm 2) adds the corresponding simplices of up to dimension p to the simplicial complex K associated to $\mathcal{W}$. When a representative point $\mathbf{u}$ is removed from $\mathcal{W}$, the algorithm DELETESIMPLICES($K, \mathbf{u}$) (Algorithm 3) deletes the simplices corresponding to $\mathbf{u}$ from K. The algorithms ADDSIMPLICES($\mathcal{W}, K, \mathbf{x}, p, \epsilon$) and DELETESIMPLICES($K, \mathbf{u}$) are detailed in Section 4.4.

The function COMPUTEPERSISTENCE(K), described in Section 4.5, computes the persistence intervals from the simplicial complex K during the offline component. The offline component is invoked at fixed intervals of Δn data points, *i.e.*, with the arrival of every new Δn points from the stream. It is worth noting that the offline computation could also be invoked by other triggers such as the requests from the user. The variable *pointCounter* of Algorithm 1 records the total number of data points that have arrived and been processed from the stream at any given time.

Since the window $\mathcal{W}$ maintains a maximum of m points in memory, the computation of persistent homology of a Vietoris–Rips complex using the sliding-window model can take up to $2^{\mathcal{O}(m)}$ space in the worst case. Although not shown explicitly in Algorithm 1, an $m \times m$ distance matrix D_m is incrementally maintained in memory that contains all the pairwise distances for the points in $\mathcal{W}$. The matrix D_m is used to look up the points in the ϵ-neighborhood of any given data point $\mathbf{x}$, and thus generates the set of simplices corresponding to $\mathbf{x}$ (see Section 4.4 for details). Additionally, D_m is utilized to update the average nearest neighbor distances of the partitions whenever a point is inserted into or removed from $\mathcal{W}$. Thus, the insertion of a new point $\mathbf{x}$ and the removal of an existing point $\mathbf{u}$ takes $\mathcal{O}(m^2)$ time, the time required to update the distance matrix D_m. On the other hand, it takes $\mathcal{O}(m)$ time, the maximum time required to find the distance δ, when a new point from the stream is not accepted for insertion into $\mathcal{W}$. Thus, the run time and the memory usage of the sliding-window model largely depend on the maximum size of the window, m. In Section 6, it is shown that a small to moderate size (typically up to $m = 100$) of the window is often preferred for practical applications, a fact that makes the sliding-window model fast and memory efficient.

Algorithm 1 Streaming Persistent Homology using Sliding-Window Model

Given: $stream, m, f, p, \varepsilon, \Delta n$

Initialize $\mathcal{W}$ as an empty container
$K \leftarrow \emptyset$
$pointCounter \leftarrow 0$
while new data points arrive from *stream* **do**
  Read the next point $\mathbf{x}$ from *stream*
  $pointCounter \leftarrow pointCounter + 1$
  if $\mathcal{W}$.SIZE( ) $< m$ **then**
    $\mathcal{W}$.INSERT($\mathbf{x}$)
    $K \leftarrow$ ADDSIMPLICES($\mathcal{W}, K, \mathbf{x}, p, \varepsilon$)
  else
    $\mathbf{y} \leftarrow$ NEARESTNEIGHBOR($\mathbf{x}, \mathcal{W}$) ▷ Return the data point nearest to $\mathbf{x}$ in $\mathcal{W}$
    $\delta \leftarrow dist(\mathbf{x}, \mathbf{y})$
    if $\mathcal{W}$ has only one partition **then**
      if $\frac{\delta}{d_{avg}} > f$ **then** ▷ d_{avg} is the average nearest neighbor distance of the partition in $\mathcal{W}$
        $\mathbf{u}_1 \leftarrow \mathcal{W}$.REMOVE($\mathbf{u}_1$) ▷ Remove and return the oldest point $\mathbf{u}_1$ from $\mathcal{W}$
        $K \leftarrow$ DELETESIMPLICES($K, \mathbf{u}_1$)
        Assign $\mathbf{x}$ to a new partition $\mathcal{P}_{new}$
        $\mathcal{W}$.INSERT($\mathbf{x}$)
        $K \leftarrow$ ADDSIMPLICES($\mathcal{W}, K, \mathbf{x}, p, \varepsilon$)
      end if
    else ▷ $\mathcal{W}$ has more than one partition
      Find the partition $\mathcal{P}$ nearest to $\mathbf{x}$ such that $\mathbf{y} \in \mathcal{P}$
      if $\frac{\delta}{d_{avg}} \leq f$ **then** ▷ d_{avg} is the average nearest neighbor distance of $\mathcal{P}$
        Assign $\mathbf{x}$ to $\mathcal{P}$
      else
        Assign $\mathbf{x}$ to a new partition $\mathcal{P}_{new}$
      end if
      if there are one or more outdated partitions in $\mathcal{W}$ **then**
        Find the oldest point $\mathbf{u}_{so}$ in the smallest outdated partition
        $\mathbf{u}_{so} \leftarrow \mathcal{W}$.REMOVE($\mathbf{u}_{so}$)
        $K \leftarrow$ DELETESIMPLICES($K, \mathbf{u}_{so}$)
      else ▷ There is no outdated partition in $\mathcal{W}$
        if $\mathbf{x}$ was assigned to a new partition $\mathcal{P}_{new}$ **then**
          $\mathbf{u}_1 \leftarrow \mathcal{W}$.REMOVE($\mathbf{u}_1$) ▷ $\mathbf{u}_1$ is the oldest point in $\mathcal{W}$
          $K \leftarrow$ DELETESIMPLICES($K, \mathbf{u}_1$)
        else ▷ $\mathbf{x}$ was assigned to the existing partition $\mathcal{P}$
          Find the oldest point $\mathbf{u}_o \in \mathcal{W}$ from any partition $\mathcal{P}_o \neq \mathcal{P}$
          $\mathbf{u}_o \leftarrow \mathcal{W}$.REMOVE($\mathbf{u}_o$)
          $K \leftarrow$ DELETESIMPLICES($K, \mathbf{u}_o$)
        end if
      end if
      $\mathcal{W}$.INSERT($\mathbf{x}$)
      $K \leftarrow$ ADDSIMPLICES($\mathcal{W}, K, \mathbf{x}, p, \varepsilon$)
    end if
  end if
  if ($pointCounter$ modulo Δn) $= 0$ **then**
    COMPUTEPERSISTENCE(K)
  end if
end while

4.4 | Incremental Maintenance of the Complex

Along with the incremental maintenance of the window $\mathcal{W}$, a weight-filtered simplicial complex K is maintained in memory, whose vertices or 0-simplices correspond to the set of data points in $\mathcal{W}$. The algorithm ADDSIMPLICES($\mathcal{W}, K, \mathbf{x}, p, \varepsilon$) is based on the incremental Vietoris–Rips expansion[2] procedure given by Zomorodian[7]. It adds the simplices (corresponding to a new point $\mathbf{x}$) of up to dimension p to K. The algorithm requires an arbitrary but total ordering of the data points on which the simplicial complex K is constructed. Since points are inserted sequentially into $\mathcal{W}$ and are then maintained in the order in which they were inserted, such a total ordering is satisfied by the points in $\mathcal{W}$. It is assumed that the later a point is inserted into $\mathcal{W}$, the higher the point is in the ordering. The algorithm is given in Algorithm 2.

[2]The construction of 2- (and higher-dimensional) simplices from a graph is called the *expansion*.

Algorithm 2 The addition of new simplices during the online component

procedure ADDSIMPLICES($(\mathcal{W}, K, \mathbf{x}, p, \varepsilon)$)
  $L \leftarrow$ FINDNEIGHBORS($\mathbf{x}, \mathcal{W}, \varepsilon$)
  ADDCOFACES($\mathcal{W}, K, \{\mathbf{x}\}, L, p$)
  return K
end procedure
procedure FINDNEIGHBORS ($\mathbf{x}, \mathcal{W}, \varepsilon$)
  return $\left\{\mathbf{y} \in \mathcal{W} \mid \mathbf{x} > \mathbf{y},\ dist(\mathbf{x}, \mathbf{y}) \leq \varepsilon\right\}$
end procedure
procedure ADDCOFACES ($\mathcal{W}, K, \tau, L, p$)
  $K \leftarrow K \cup \tau$
  if $dim(\tau) \geq p$ **then**
    return
  else
    for each $\mathbf{y} \in L$ **do**
      $\sigma \leftarrow \tau \cup \{\mathbf{y}\}$
      $M \leftarrow L \cap$ FINDNEIGHBORS($\mathbf{y}, \mathcal{W}, \varepsilon$)
      ADDCOFACES($\mathcal{W}, K, \sigma, M, p$)
    end for
  end if
end procedure

Algorithm 3 The removal of simplices during the online component

procedure DELETESIMPLICES($(K, \mathbf{u})$)
  $R \leftarrow \left\{\sigma_R \in K \mid \{\mathbf{u}\} \cap \sigma \neq \emptyset\right\}$
  $K \leftarrow K \setminus R$
  return K
end procedure

Algorithm 4 The standard algorithm

for $j = 1$ to n_K **do**
  while there exists $j_0 < j$ with $low(j_0) = low(j)$ **do**
    add column j_0 to column j
  end while
end for

When a new point $\mathbf{x}$ is inserted into $\mathcal{W}$, the procedure FINDNEIGHBORS($\mathbf{x}, \mathcal{W}, \varepsilon$) returns the points in the ε-neighborhood of $\mathbf{x}$ by looking up the distances from $\mathbf{x}$ to other points in $\mathcal{W}$ from the distance matrix D_m. ADDCOFACES($\mathcal{W}, K, \{\mathbf{x}\}, L, p$) recursively adds all the simplices whose maximal vertex is $\mathbf{x}$ according to the total ordering of the data points in $\mathcal{W}$. Since $\mathbf{x}$ is a simplex in K, every σ computed by ADDCOFACES will also be a simplex in K. Therefore, M is the set of points shared by the ε neighborhoods of all the vertices of σ whose cofaces are recursively added to K. The computation of the weights of simplices, though not shown explicitly, is done along with the creation of simplices by the procedure ADDCOFACES.

When a point $\mathbf{u}$ is removed from $\mathcal{W}$, the algorithm for deletion of the corresponding simplices from K is given in Algorithm 3. R is a set of simplices such that $\{\mathbf{u}\}$ is a face of any simplex $\sigma_R \in R$. Then the simplices in R are removed from K by the set difference $K \setminus R = \left\{\sigma \in K \mid \sigma \notin R\right\}$.

4.5 | Offline Component: Computation of Persistence Intervals

When the function COMPUTEPERSISTENCE(K) is called in Algorithm 1, a total ordering is imposed on the simplices of the weight-filtered complex K such that:

- the simplices are sorted according to their weights, and
- a face of a simplex precedes the simplex.

Let n_K be the total number of simplices in the weight-filtered complex K. The simplices with the total ordering imposed on them are denoted by $\sigma_1, \sigma_2, ..., \sigma_{n_K}$. A square matrix ∂, called the *boundary matrix*, of order n_K is constructed as below:

$$\partial[i,j] = \begin{cases} 1, & \text{if } \sigma_i \text{ is a co-dimension one face of } \sigma_j \\ 0, & \text{otherwise.} \end{cases}$$

The columns and rows of ∂ represent the simplices of the filtration arranged according to the total order. The boundary (or, co-dimension one face) of a simplex is recorded in its column (by a 1 in the corresponding row).

The boundary matrix ∂ is reduced to another $0-1$ matrix ∂_R by Algorithm 4, called the *standard algorithm*[22,19]. Let $low(j)$ be the row index of the lowest 1 (*i.e.*, the highest row index of a 1) in column j. If the entire column is zero, then $low(j)$ is undefined. The columns of ∂ are scanned from left to right, and when a column j is reached such that there is another column $j_0 < j$ with $low(j_0) = low(j)$, the column j_0 is added to j. The boundary matrix is *reduced* when $low(j_0) \neq low(j)$ for any two non-zero columns $j_0 \neq j$. After the boundary matrix ∂ is reduced to ∂_R, the birth and death times of topological features are recorded from the reduced matrix ∂_R.

The worst case run time of the standard algorithm is cubic in the number of simplices n_K. In practice, the standard algorithm has displayed a quasi-linear behavior on real-world data[27]. A number of solutions have been designed to improve the worst case run time[20,27].

Once the birth and death times ($\varepsilon_{birth}, \varepsilon_{death}$), or the persistence intervals, of topological features are recorded from the reduced matrix ∂_R, they can be plotted as barcodes or persistence diagrams. The sequence of barcodes, persistence diagrams or their distance values, displayed at fixed time intervals during the progress of the stream, is the final outcome of the sliding-window model.

It is worth clarifying that even though simplices are incrementally added to and deleted from K during the online component, the computation of zigzag persistent homology[28] is not required during the offline component. This is because when the function COMPUTEPERSISTENCE(K) is invoked, the simplicial complex K that exists in memory at that time, is used for the construction of the boundary matrix ∂. Thus, the persistence intervals can be computed from the filtration of K itself by utilizing the *standard algorithm*.

5 | INPUT PARAMETERS

This section provides a detailed discussion of the input parameters of the sliding-window model. The maximum dimension p for the computation of homology, and the maximum value of the scale parameter ε are the input parameters for persistent homology itself, and are described in Section 3.

5.1 | The Maximum Size of the Window: m

The maximum size of the window, m, is the maximum number of data points the window may contain. The higher the value of m, the larger the simplicial complex K maintained in memory. And the larger the complex K, the higher the run time and memory requirements of the model.

The choice of the value of m should depend on the requirements of the application. If the user is concerned only about the detection of changes (*i.e.*, the binary categorization of data objects) in a stream, the user can choose a small value of m that enables faster processing, lower memory usage and the ability to detect changes of both shorter and longer duration. On the other hand, if the user is primarily concerned about changes of longer duration and wants the additional capability of visually identifying the different types of data objects in the stream, the user should choose a window size m that is sufficiently large (typically between 50 and 100 data points) to form visually identifiable topological structures.

5.2 | Update Sensitivity of the Window: f

The parameter f determines the rate of acceptance of the new data points for insertion into the window. The smaller the value of f, the higher the probability of a new data point $\mathbf{x}$ being inserted into the window. In other words, f represents the update sensitivity (or equivalently, the sliding speed) of the window (that has only one partition in it), with smaller values of f corresponding to higher frequency of updates (or, higher speed of sliding).

For a given data stream, f represents a trade-off between the run time and the guarantee of change detection. A sufficiently small value of f forces the window to accept every new data point from the stream, and hence, guarantees that every change in the stream is detected (for appropriate values of the window size m and offline interval Δn). However, such a guarantee comes at the cost of run time that can increase significantly due to frequent updates to the simplicial complex. On the other hand, a high value of f reduces the run time of the model (due to reduced acceptance of new data points and consequently infrequent updates to the complex), but can lead to missing concept drifts in the stream.

5.3 | Frequency of Offline Computation: Δn

The parameter Δn determines how often the offline component is invoked. The value of Δn represents a trade-off between the run time and the guarantee of change detection. If $\Delta n = 1$, *i.e.*, if the offline computation is done with the arrival of every new data point from the stream, the sliding-window model is guaranteed to detect every change in the stream (for appropriate values of m and f). However, invoking the offline component too often increases the overall run time (*i.e.*, reduces the speed of processing the stream). On the other hand, performing the offline computation too infrequently decreases the overall run time, but may lead to some of the changes in the stream getting undetected.

5.4 | The Rate at which Existing Partitions in the Window Are Outdated: n_o

A partition is considered outdated if no new data point was assigned to it during the last n_o insertions into the window. A smaller value of n_o quickly outdates existing partitions that, in turn, are removed from the window. For rapidly evolving streams, a small value of n_o should be chosen.

In general, the optimal values of m, f, Δn and n_o depend on the application. If the user expects frequent changes of short duration in the stream, then small values of m and Δn commensurate with the duration of changes should be chosen. If the user has no prior knowledge about the properties of the stream, then, as a rule of thumb, the user should begin with small values of m, f, Δn and n_o, and then configure those values to obtain an optimal trade off between the run time, memory usage, and parameter values.

6 | APPLICATION

This section describes the application of the sliding-window model in the detection of anomalous connections in large streams of network traffic. In an usual real-world scenario, most of the TCP connections in a local area network of computers are *normal connections*. However, there can be occasional bursts of attacks at certain times. It is shown that whenever an attack or a set of anomalous connections occurs, the topological structures of the output from the sliding-window model change to indicate the occurrence of the event. In addition, the user can identify the types of several attacks by visualizing their topological properties through barcodes or persistence diagrams. In the following subsections, the sliding-window model is evaluated on two of the data sets most widely used to benchmark the performance of network intrusion detection systems: (i) KDD Cup 1999[29], and (ii) NSL-KDD[30] data sets.

The sliding-window model is implemented in `C++` as part of the *Lightweight Homology Framework (LHF)*[31] that significantly speeds up the computation of persistent homology. The execution times reported in the following subsections are captured on a computer with an Intel(R) Core(TM) i5-4210U CPU @ 1.70 GHz and 8 GB of memory, and are averaged over 5 runs. The data sets used for the experiments were stored in the hard disk of the computer, and were read as file streams (*i.e.*, one data point at a time). The accuracy and effectiveness of the model are demonstrated by its ability to promptly and correctly identify anomalous connections in the streams of TCP traffic in computer networks.

6.1 | KDD Cup 1999 Data Set

The KDD Cup 1999 data set is one of the most widely used real data for the evaluation of network intrusion detection systems and stream clustering algorithms[32,8]. The data set represents the important problem of automatic and real-time detection of cyber attacks, and consists of a series of TCP connection records from two weeks of network traffic managed within a military local area network[24,33]. Each record corresponds to either a normal connection, or an intrusion/attack. Most of the connections in this data set are normal, but occasionally there could be a burst of attacks at certain times.

In the data set, there are a total of 22 types of intrusions that fall into four main categories[29]:

- Denial of Service (DoS): For example, syn flood ('neptune', 'smurf');
- Probing: Surveillance and other probing (*e.g.*, 'ipsweep', 'satan');
- Remote to Local (R2L): Unauthorized access from a remote machine (*e.g.*, guessing password, 'warezmaster'); and
- User to Root (U2R): Unauthorized access to local superuser (root) privileges (*e.g.*, various buffer overflow attacks).

The data is available both as a complete set that contains approximately 4.9 million records and as a 10% sub-sampled set containing 494, 021 points. Each connection record consists of 41 features plus a class ID.

As in almost all the previous papers using the data set, the evaluations performed in this section consider the sub-sampled set and retain 34 continuous valued features, pruning out the remaining discrete or categorical attributes[34,24,33,35,36,37]. The resulting data is normalized by dividing each value of a data vector by the standard deviation of the corresponding attribute[33,37].

Each of the four main categories of intrusions in the KDD Cup 1999 data set belongs to one of the three types of anomalies[32]:

1. *Collective anomaly:* When a group of similar data objects behave anomalously with respect to the entire data set, but a single data object from that group is not anomalous by itself, the group is called a collective anomaly. DoS attacks are categorized as the collective anomaly. For example, a DoS attack consisting of numerous connection requests to a web service within a very short period of time is a collective anomaly, but a single request can be legitimate.
2. *Contextual anomaly:* When data objects behave anomalously in a particular context, they are called a contextual anomaly. A probing attack that is launched to gather information about a targeted network or host is an example of a contextual anomaly.
3. *Point anomaly:* When a certain data object or a small set of data objects deviates from the normal pattern of other data objects, it is called a point anomaly. R2L and U2R attacks aimed at gaining unauthorized user or administrative access to a machine or an account are considered point anomalies.

Thus, the intrusions or attacks present in the KDD Cup 1999 data set vary widely in their properties, purposes and mechanisms of launching the attacks. Accordingly, the topological structures of the connections corresponding to those attacks are different from one another. In order to demonstrate and visualize the different capabilities of the sliding-window model and its effectiveness in identifying different types of traffics, two separate subsets, called Subset 1 and Subset 2, are constructed from the 10% sub-sampled data set. Subset 1 consists of those intrusions that are launched using large number (typically from hundreds to thousands) of connections, namely, DoS and probing attacks. It is worth noting that a DoS attack requires no prior access to the target, and hence, is relatively easier to launch than other attacks. For these reasons, the DoS attack is often considered the most dreaded attack and receives the highest priority for attack detection in intrusion detection systems[32]. On the other hand, Subset 2 contains those attacks that are carried out using small number (typically from a few to tens) of connections, namely R2L and U2R attacks.

6.1.1 | Subset 1: Attack Types with Large Number of Connections

Subset 1 contains of a total of 25, 800 connection records. The occurrence of different types of traffic with the progress of the stream is shown in Table 2. Figures 4 and 5 show the detection of attacks with the help of the sliding-window model for the Subset 1 data stream. Figure 4 displays the *distances* between pairs of consecutive persistence diagrams generated by the sliding-window model with the progress of the stream. On the other hand, Figure 5 shows the *distances* from the persistence diagram of a reference normal traffic to all other persistence diagrams generated by the model. The *distance* between two

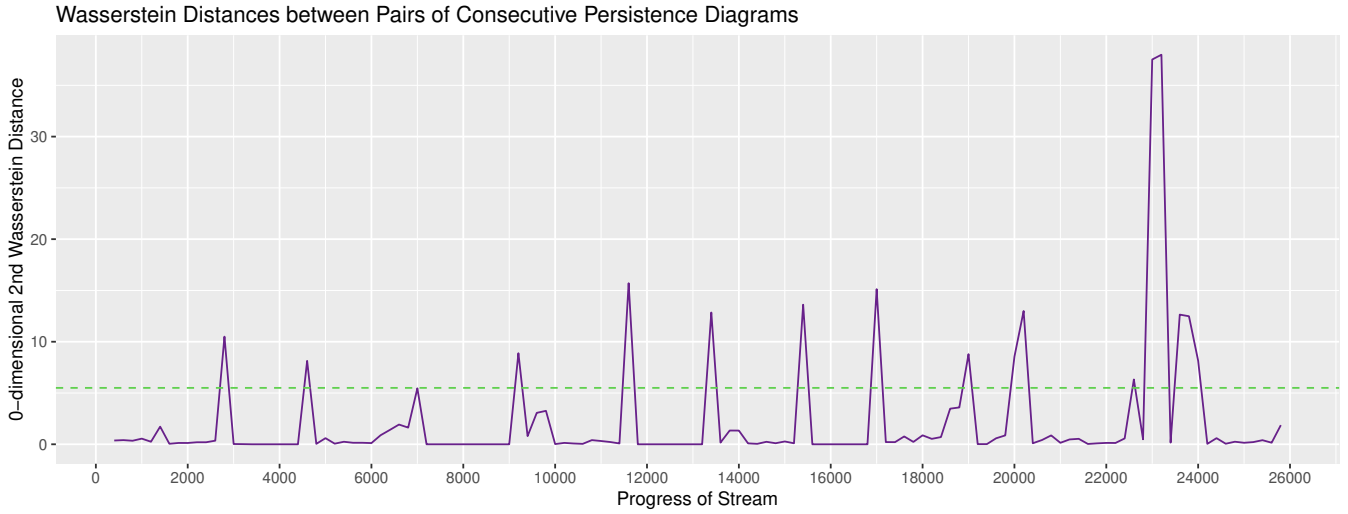

FIGURE 4 Detection of attacks using Wasserstein distances between pairs of consecutive persistence diagrams for Subset 1 data stream. The dashed horizontal line at $W_2 = 5.5$ represents the threshold for 'spike' detection.

persistence diagrams is measured using a metric called the 0-dimensional 2^{nd} Wasserstein distance, or W_2 for short. The 2^{nd} Wasserstein distance is commonly used to quantify the difference between two persistence diagrams. The 0-dimensional distance is used due to the lack of consistent 1-dimensional features across different types of connections in the KDD Cup 1999 data set. The interpretations of these figures are more thoroughly discussed below.

Figure 4 shows the plot of W_2 between pairs of consecutive persistence diagrams with the progress of the stream. For Subset 1, the sliding-window model is configured with $m = 100$, $f = 6$, and $\Delta n = 200$. Each type of traffic has its own topological structure that is usually distinct from other types. Therefore, the distance W_2 between a pair of persistence diagrams obtained from the same type of traffic is much smaller than that obtained from different types of traffics. For example, W_2 between two persistence diagrams obtained from the normal connection is smaller than that obtained from the normal connection and 'neptune' attack. Thus, whenever the type of traffic changes in the data stream (for example, an attack is launched or ended), W_2 between the current and previous persistence diagrams increases substantially. A 'spike' in Figure 4, therefore, represents a transition from one type of traffic to another, or more specifically, either the launch or the end of an attack.

TABLE 2 The occurrence of different types of connections with the progress of Subset 1 data stream

Type	No. of Connections	Progress of Stream
Normal	2600	2600
smurf (DoS)	1800	4400
Normal	2400	6800
neptune (DoS)	2200	9000
Normal	2403	11403
neptune (DoS)	1802	13205
Normal	2000	15205
smurf (DoS)	1604	16809
Normal	2041	18850
ipsweep (Probing)	1153	20003
Normal	2402	22405
satan (Probing)	1404	23809
Normal	1991	25800

For Figure 5, the persistence diagram obtained after the first 200 normal connections from the stream is considered as a *reference*. The figure shows the plot of W_2 between the reference persistence diagram and that obtained from each subsequent computation of persistence intervals. Since the normal connections have similar topological properties, the distance W_2 between the reference persistence diagram and the diagram obtained from each set of subsequent normal connections is small. Thus, the 'troughs' in Figure 5 represent normal connections. On the other hand, the attacks are topologically dissimilar to the normal connections. The distance W_2 between the reference persistence diagram and the diagram obtained from a set of anomalous connections is large. Therefore, the 'plateaus' in Figure 5 represent attacks. Thus, by continuously monitoring

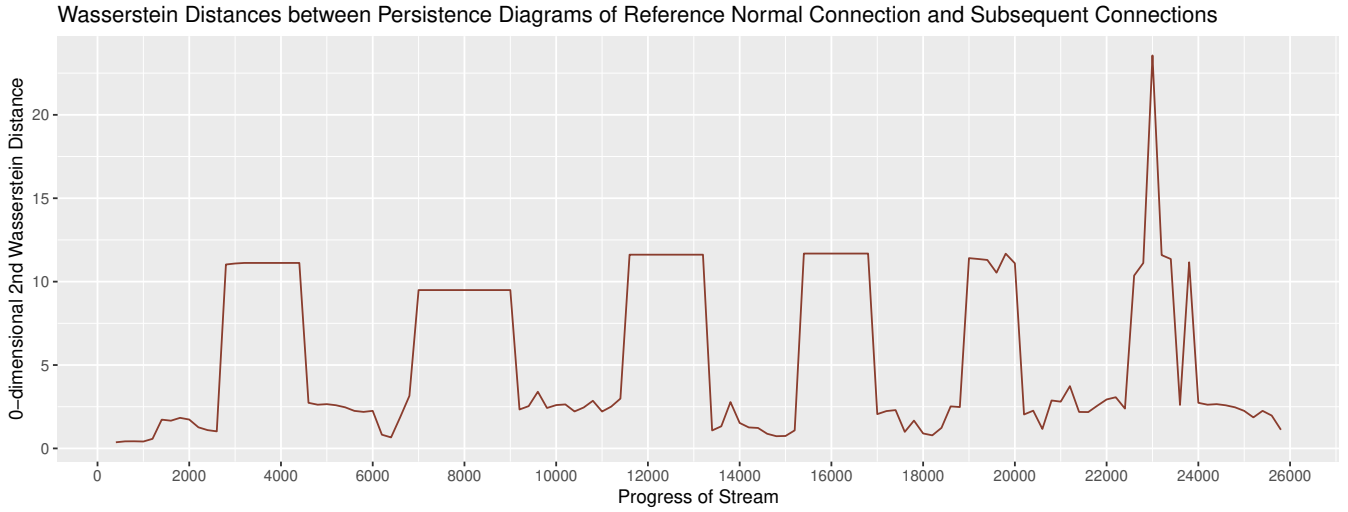

FIGURE 5 Detection of attacks using Wasserstein distances between persistence diagrams of reference normal connection and subsequent connections for Subset 1 data stream

the distance between persistence diagrams generated by the sliding-window model, one can identify attacks in a data stream of network traffic.

In addition to the detection of attacks, the persistence intervals generated by the sliding-window model can help identify the *types* of several attacks. Figure 6 shows a visual representation (using barcodes) of the topological structures of different types of traffics in Subset 1 data stream. The figure consists of two instances of the barcodes for each type of traffic that occurs at different times during the progress of the stream. In Appendix A, the topological structures of the same instances of traffic as in Figure 6 are shown using persistence diagrams.

It is evident that the topological structure of each type of traffic is distinct from that of other types of traffics. For example, the H_0 barcodes have similar shapes and structures for the two instances of normal traffic. The H_1 features, while not significant, appear sporadically on both examples of normal connections. The smurf attacks show mostly insignificant H_0 features, and they do not form any H_1 loops at all. On the other hand, neptune attacks display dense but insignificant H_0 and H_1 features on both the instances examined in Figure 6. Thus, in addition to the detection of attacks, it is possible to visually identify the types of several traffics from the output of the sliding-window model.

It can be noted that most other network anomaly detection systems based on traditional data mining techniques such as classification, clustering or information theory can categorize the connections either as attacks or as normal traffic[32]. Even when multiple (*i.e*, more than two) clusters or groups of connections are discovered, classification or clustering based methods cannot directly identify the *types* of those groups. In contrast, however, persistent homology provides the capability for the identification of several types of connections in addition to the binary categorization of network traffic into attacks and normal connections. Moreover, due to some variations within an individual type of traffic, clustering based anomaly detection systems often create multiple groups for the same type of traffic. However, since each individual type of traffic maintains a consistent topological structure through the stream, analyzing the topology of different types of connections results in more accurate identification of the number of clusters or groups of traffic in the stream[38].

TABLE 3 Confusion matrix for Subset 1 data stream

		Actual	
		Positive	**Negative**
Detected	**Positive**	$TP = 12$	$FP = 5$
	Negative	$FN = 0$	$TN = 111$

In the following, a detailed analysis of the accuracy and abilities of the sliding-window model to correctly profile different types of network traffic is presented. Based on the distribution of values of W_2 between pairs of consecutive persistence diagrams, a 'spike' in Figure 4 is empirically defined as $W_2 \geq 5.5$. The components related to the accuracy of the sliding-window model are computed based on the numbers described below:

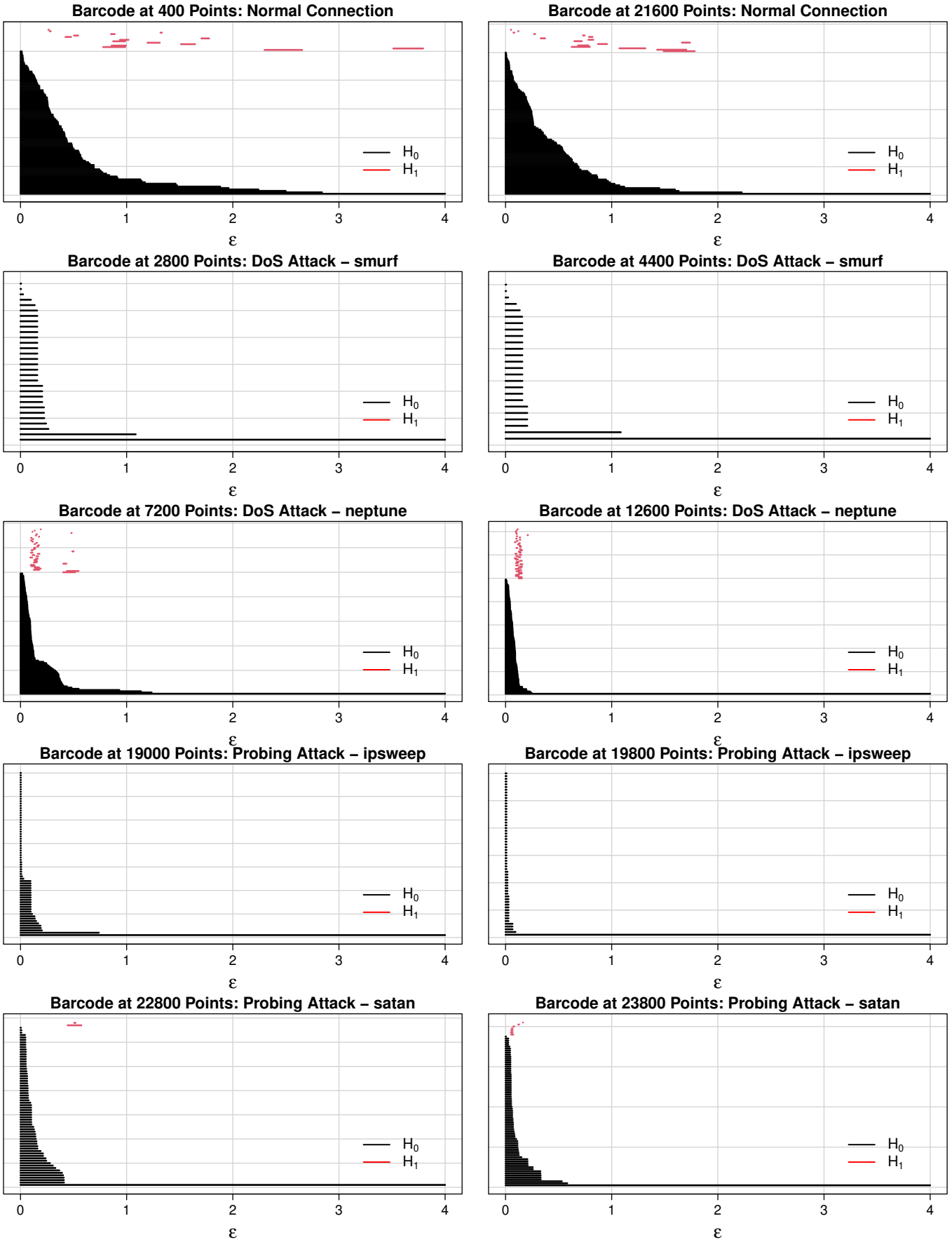

FIGURE 6 Identification of types of traffic in Subset 1 data stream using barcodes

TABLE 4 Performance of the sliding-window model on Subset 1 and Subset 2 data streams

Metric	Subset 1	Subset 2
m	100	10
Accuracy	0.961	0.890
Precision	0.706	0.474
TPR	1.000	0.900
FPR	0.043	0.111
Avg Runtime: of Online Component (sec/1,000 pts)	3.831	0.208
Avg Runtime: of Offline Component (sec/matrix reduction)	0.496	0.0004
Avg # of Simplices	124,277	109
Avg Memory Used (MB)	16.119	0.014

TABLE 5 The occurrence of different types of connections with the progress of Subset 2 data stream

Type	No. of Connections	Stream Progress
Normal	200	200
guess password (R2L)	20	220
Normal	120	340
buffer overflow (U2R)	10	350
Normal	150	500
imap (R2L)	10	510
Normal	100	610
rootkit (U2R)	10	620
Normal	180	800
warezmaster (R2L)	10	810
Normal	200	1010

- **True Positives** (TP): The number of times the changes in traffic type are *correctly* and *timely* detected. In other words, TP is the number of times the changes in traffic are detected at the offline computations immediately following the respective changes.
- **True Negatives** (TN): The number of offline computations that correctly identify intervals of *stable* traffic. In other words, TN is the number of offline computations that correctly identify the intervals where the traffic type does not change.
- **False Positives** (FP): The number of offline computations that incorrectly detect changes in traffic type during *stable* intervals.
- **False Negatives** (FN): The number of offline computations that fail to detect changes in network traffic.

Based on the four numbers described above and the number of 'spikes' formed in Figure 4, the confusion matrix obtained for the Subset 1 data stream is shown in Table 3. From the results in this Table, four components to highlight the accuracy of the sliding-window model are computed, namely:

- Accuracy = $\dfrac{TP+TN}{TP+FP+FN+TN}$
- Precision = $\dfrac{TP}{TP+FP}$
- Recall or True Positive Rate (TPR) = $\dfrac{TP}{TP+FN}$
- False Positive Rate (FPR) = $\dfrac{FP}{FP+TN}$

The resulting values of these components of accuracy are given in Table 4. In addition, Table 4 shows the run times for the online and offline components of the sliding-window model. The run time for the online component depends on the parameter m and the proportion of new data points that are discarded or filtered out (*i.e.* not chosen to be inserted into the window) by the model. The bigger the size of the window (*i.e.*, m), the larger the simplicial complex. And the larger the complex is, the more time it takes for the simplices to be added to and deleted from the simplicial complex. The run time for the online component is reported as the average time required to summarize 1,000 data points from the Subset 1 stream. The run time for the offline component depends on the parameter m. A bigger window size and consequently a larger simplicial complex result in a larger boundary matrix that takes longer to be *reduced* than a smaller one. The run time for the offline component is given as the average time required to reduce the boundary matrix generated from the complex constructed on $m = 100$ data points. In addition, Table 4 shows the average memory usage of the sliding-window model in megabytes (MB) and in terms of the average number of simplices in the complex through the progress of the stream.

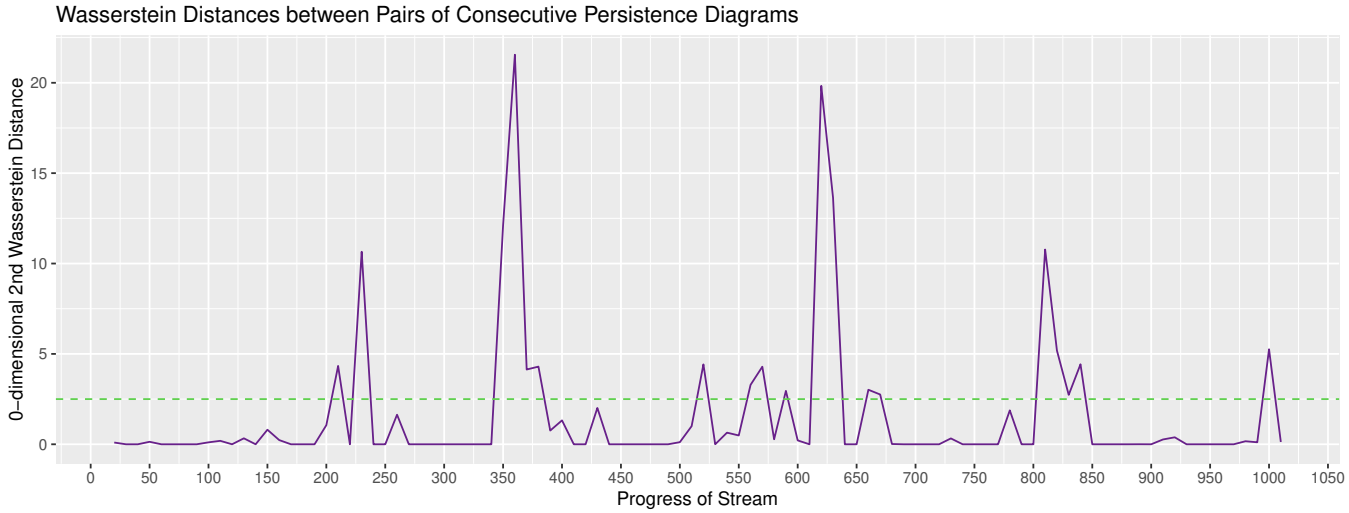

FIGURE 7 Detection of attacks using Wasserstein distances between pairs of consecutive persistence diagrams for Subset 2 data stream. The dashed horizontal line at $W_2 = 2.5$ represents the threshold for 'spike' detection.

6.1.2 | Subset 2: Attack Types with Few Connections

In the previous subsection, the sliding-window model is used to detect attack types that are carried out using large number of connections, such as the DoS and probing attacks. However, the model is also capable of detecting intrusion events that are performed with small number of connections, such as the R2L and U2R attacks. The Subset 2 traffic in this study consists of R2L and U2R attacks (as well as normal traffic). It contains a total of $1,010$ connection records. The occurrence of different types of traffic with the progress of the stream is shown in Table 5.

In order to detect intrusions carried out by such small number of connections, the size of the window (*i.e.*, m) is made extremely small. For example, the window is scaled down to the size of the smallest chunk of attacks in Subset 2 by setting $m = 10$. In addition, the offline computation is performed frequently enough to detect the rapid changes in the traffic type caused by the R2L and U2R attacks. For instance, Δn is set to 10 for detecting the attacks in Subset 2 data stream. Decreasing the values of the parameters m and Δn enables the sliding-window model to scan the stream through a 'lens' of higher magnification that helps analyze the data at a finer granularity.

Figures 7 shows the W_2 distances comparing adjacent persistence diagrams, and Figure 8 shows the W_2 distances comparing the persistence diagrams to that of the reference normal traffic. In Figure 8, the persistence diagram obtained after the first 10 normal connection records from the stream is considered as a *reference*. Unlike in Figure 5, the attacks do not form 'plateaus' in Figure 8. Since the durations of R2L and U2R attacks are small, they are displayed as 'spikes' in both Figures 7 and 8. A 'spike' in Figure 7 is empirically defined as $W_2 \geq 2.5$ based on the distribution of values of W_2 between pairs of consecutive persistence diagrams. The confusion matrix obtained from the detection of attacks in Subset 2 data stream is given in Table 6. The accuracy, run time and memory usage of the sliding-window model for Subset 2 data stream is shown in Table 4. For this data stream, the update sensitivity factor f is set to 4.

Table 4 shows that the sliding-window model generates high values of $Accuracy$ for both Subset 1 and Subset 2 data streams. The high values of true negatives contribute to the high $Accuracy$ of the model. While a large number of false positives results in a moderately low value of $Precision$ for Subset 2 stream, the extremely low values of false negatives produce high $Recall$ for both the streams. In almost all practical applications, having low false negatives is more important than having low false positives. The low values of false negatives demonstrate that the sliding-window model seldom fails to detect changes in the traffic types. In addition, the higher values of false positives are offset by high true negatives, producing small $False\ Positive\ Rates$ for both Subset 1 and Subset 2 data streams.

From Table 4 it is observed that the run time and the memory usage of the sliding-window model decrease substantially with the size of the window. The ability to adjust the time and memory requirements of the model creates a unique advantage for the sliding-window over other techniques for network anomaly detection. Intrusion detection methods based on traditional

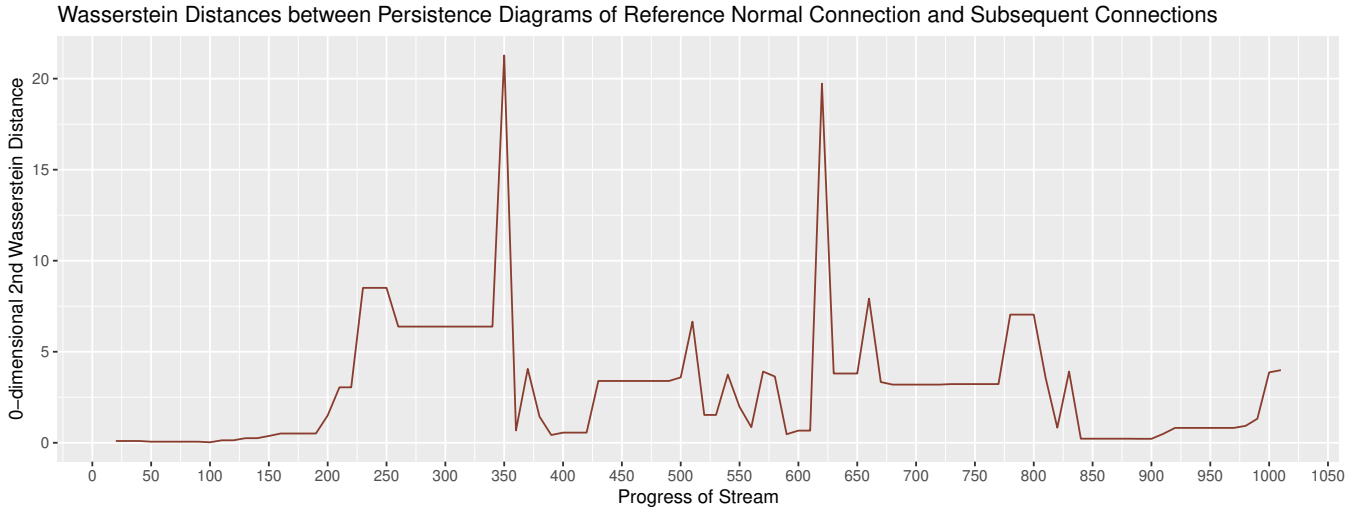

FIGURE 8 Detection of attacks using Wasserstein distances between persistence diagrams of reference normal connection and subsequent connections for Subset 2 data stream

TABLE 6 Confusion matrix obtained for the sliding-window model on Subset 2 data stream

		Actual	
		Positive	**Negative**
Detected	**Positive**	$TP = 9$	$FP = 10$
	Negative	$FN = 1$	$TN = 80$

TABLE 7 Confusion matrix obtained for the microcluster-based framework on Subset 2 data stream

		Actual	
		Positive	**Negative**
Detected	**Positive**	$TP = 7$	$FP = 33$
	Negative	$FN = 3$	$TN = 57$

data mining do not have the ability to substantially alter their run times and memory usages without affecting the accuracy[3]. The capability of significantly reducing the time and space requirements of the sliding-window model makes it suitable for fast-moving streams with rapidly changing traffic types.

The advantages of faster stream processing and lower memory usage of the sliding-window model come at the cost of its diminished capability of visually recognizing different types of traffic. While the model configured with an extremely small window is capable of the binary categorization of traffic into attacks and normal connections, the complex constructed on the data points in the window is not large enough to form visually recognizable topological structures. As a result, the visual identification of different types of traffic may not be possible with an extremely small window size.

In practice, the choice of the size of the window should depend on the requirements of the application. For example, if a user is concerned only about the detection of attacks (*i.e.*, the binary categorization of traffic) in a computer network, then the user should choose a small window size that enables faster processing, lower memory usage and the ability to detect all types (DoS, probing, R2L and U2R) of attacks. However, if the user is primarily concerned about DoS and probing attacks, and wants the additional capability of visually identifying the different subtypes among those attacks, the user should choose a window size that is sufficiently large (typically between 50 and 100 data points) to form visually identifiable topological structures.

It is evident that the mechanisms and capabilities of the sliding-window model for the detection of network anomalies are vastly different than those of existing techniques. For instance, classification and clustering based anomaly detection systems

[3]The difference in accuracy of the sliding-window model between Subset 1 and Subset 2 is due to the difference in traffic types in the two data streams.

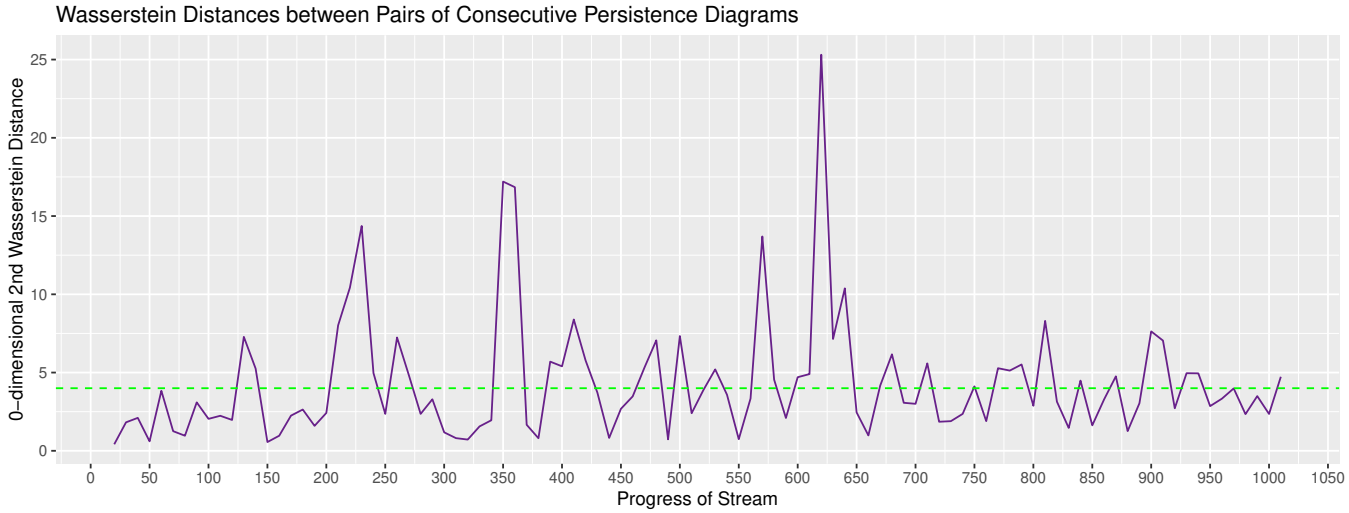

FIGURE 9 Detection of attacks by the microcluster-based framework using Wasserstein distances between pairs of consecutive persistence diagrams for Subset 2 data stream

categorize *each individual connection* either as normal or as anomalous. None of those techniques provides the option to significantly alter the processing speed or the memory usage (without affecting the accuracy). On the contrary, the sliding-window model does not attempt to determine the category of each individual connection. Instead, the model computes the topological properties of a set of connections, and looks for changes to those topological properties as the stream progresses. For these reasons, a fair comparison of the performance between the sliding-window model and existing methods is not possible. However, the comparison of the output of the model to the *baseline* (or *ground truth*) provides an accurate representation of the performance of the model.

6.1.3 | Microcluster-based Framework Applied on Subset 2

In this subsection, the microcluster-based framework[10] is applied on Subset 2 data stream to demonstrate its inability to accurately categorize the traffic when the attacks are carried out with small number of connections. We implemented the framework using the same libraries and interfaces (*i.e.*, the `R` interface[39] to ClusTree[26] from Massive Online Analysis[40], and the `R` interface[41] to persistent homology from the GUDHI library[42,43]) as those used in[10]. In the following, we report the accuracy of the microcluster-based framework using $maxHeight = 10$ (for the maximum height of the tree), $horizon = 11$ (for the range of the time window), and $\Delta n = 10$ (for the interval of offline computations), the set of parameter values that yielded the best result in our experiments.

TABLE 8 Accuracy of the microcluster-based framework on Subset 2 data stream

Metric	Value
$maxHeight$	10
Accuracy	0.640
Precision	0.175
TPR	0.700
FPR	0.367

Figure 9 shows an attempt of the detection of attacks with the help of the microcluster-based framework for the Subset 2 data stream. Based on the distribution of values of W_2 between pairs of consecutive persistence diagrams, a 'spike' in Figure 9 is defined as $W_2 \geq 4.0$. The confusion matrix obtained from the detection of attacks by the microcluster-based framework in Subset 2 data stream is given in Table 7. The accuracy of the framework for Subset 2 data stream is shown in Table 8.

Tables 7 and 8 show that the microcluster-based framework is unable to accurately categorize the traffic in Subset 2 data stream. The framework fails to detect the changes in traffic type in three instances ($FN = 3$). Moreover, a large number of false positives results in a poor precision for the framework. These results demonstrate that the microcluster-based framework is not suitable for detecting changes of very short duration.

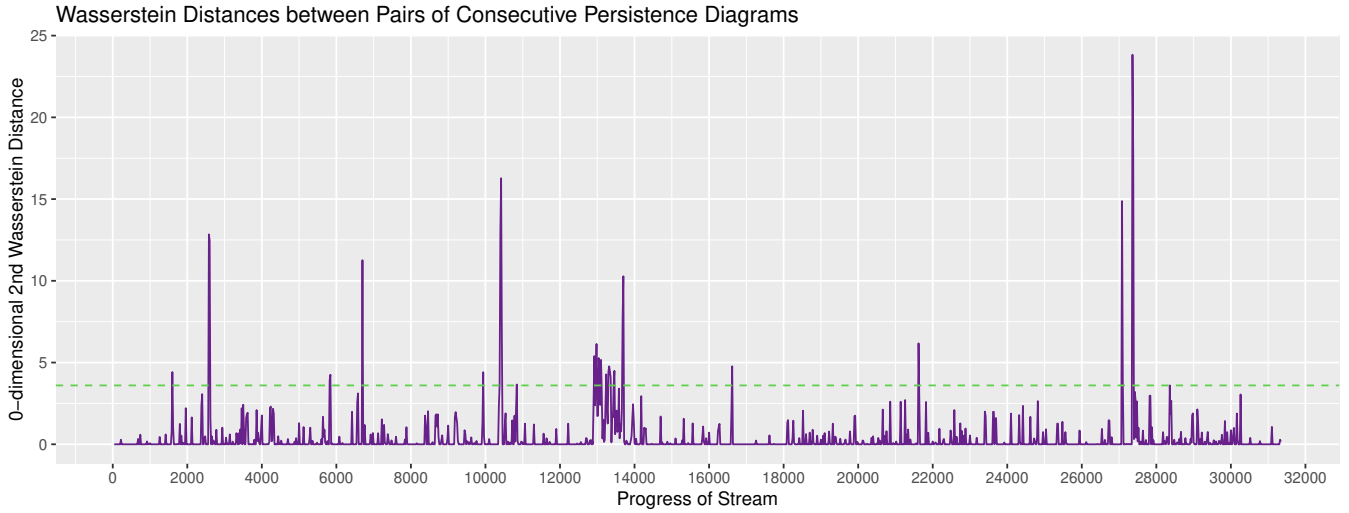

FIGURE 10 Detection of attacks using Wasserstein distances between pairs of consecutive persistence diagrams for NSL-KDD data stream. The dashed horizontal line at $W_2 = 3.6$ represents the threshold for 'spike' detection.

6.2 | NSL-KDD Data Set

In addition to the KDD Cup 1999 data, the performance of the sliding-window model is evaluated on another data set, called the NSL-KDD data set[30], that has been widely used in the literature on classification of network traffic in the recent years. In the previous section, the effect of the window size on the performance of the sliding-window model was discussed. A larger window was used for the detection and identification of DoS and probing attacks while a smaller window was used for the detection of R2L and U2R attacks. It was shown that for the binary categorization of traffic into normal or anomalous connections, the model configured with a small window size is more advantageous due to its speed and lower memory requirements than the model with a larger window size. In this section, a small window with $m = 20$ is used for the detection of all types of attacks (namely: DoS, probing, R2L and U2R) in a subset of the NSL-KDD data set. The interval of offline computation is set to $\Delta n = 20$, and the update sensitivity factor f is chosen to be 6. As with the KDD Cup 1999 data, the subset is constructed considering 34 continuous valued features from the *training set* of the NSL-KDD data to utilize the baseline or ground-truth information on the traffic type of each connection. In order to help evaluate network anomaly detection techniques that are primarily based on classification-based machine learning models, the sequence of the connections are randomly shuffled in the original NSL-KDD data set[30]. To produce a data stream with sequential changes in the traffic types, the sequence of the connections is modified in the subset of the NSL-KDD data used for the evaluation of the sliding-window model. In particular, the traffic types are arranged such that there are occasional attacks that appear (and subsequently disappear) sequentially during the flow of network traffic.

Table 9 shows the types of traffic and the progress of the stream in the subset constructed from the NSL-KDD data set. Figure 10 shows the plot of the 0-dimensional 2^{nd} Wasserstein distance (W_2) between pairs of consecutive persistence diagrams. In this plot, each 'spike' represents a change in the traffic type, and is defined as $W_2 \geq 3.6$. The confusion matrix obtained from the detection of attacks in NSL-KDD data stream is given in Table 11. The accuracy, run time and memory usage of the sliding-window model for NSL-KDD data stream is shown in Table 10. The definitions of the performance metrics (given in Table 10) and of the components of the confusion matrix (given in Table 11) are provided in Section 6.1. From Table 10 it is seen that the model accurately detects almost all occurrences of the attacks ($TP = 9$ and $FN = 1$). The low value of $Precision$ is primarily caused by 12 false positives during the occurrence of 'portsweep' (probing) attack that appears as a set of relatively short spikes between $12,920^{\text{th}}$ and $13,460^{\text{th}}$ data points in Figure 10. In addition, there are several shorter or insignificant 'spikes' throughout the stream that do not represent changes in traffic types. Those shorter 'spikes' are caused by variations within individual traffic types.

Figure 11 shows an attempt of the detection of attacks in the NSL-KDD data stream using the microcluster-based framework, which was implemented with the help of the libraries mentioned in Section 6.1.3. The following set of parameter values was used: $maxHeight = 20$, $horizon = 25$, $\Delta n = 20$. Based on the distribution of values of W_2 between pairs of consecutive persistence diagrams, a 'spike' in Figure 11 is defined as $W_2 \geq 10.8$. The confusion matrix obtained from the detection of

TABLE 9 The occurrence of different types of connections with the progress of NSL-KDD data stream

Type	No. of Connections	Stream Progress
Normal	2562	2562
buffer overflow (U2R)	20	2582
Normal	3219	5801
teardrop (DoS)	879	6680
Normal	3700	10380
guess password (R2L)	20	10400
Normal	2481	12881
portsweep (Probing)	780	13661
Normal	13400	27061
warezclient (R2L)	279	27340
Normal	4000	31340

TABLE 10 Performance of the sliding-window model on NSL-KDD data stream

Metric	Value
m	20
Accuracy	0.986
Precision	0.300
TPR	0.900
FPR	0.013
Avg Runtime: Online Component (sec/1,000 points)	0.224
Avg Runtime: Offline Component (sec/matrix reduction)	0.002
Avg Num of Simplices	717
Avg Memory Use (MB)	0.093

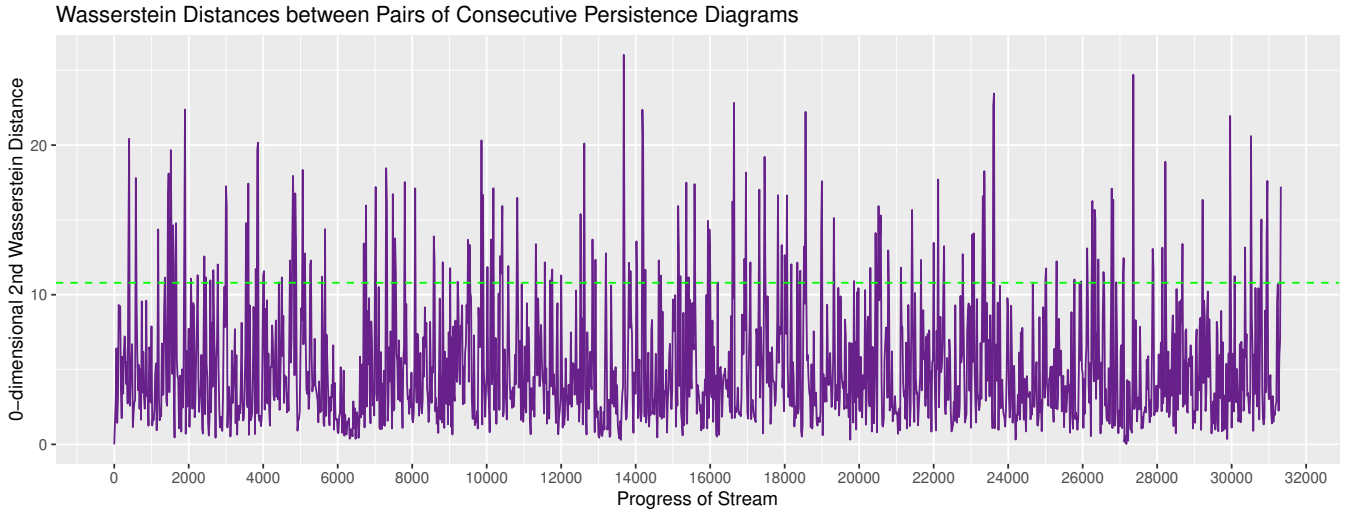

FIGURE 11 Detection of attacks by the microcluster-based framework using Wasserstein distances between pairs of consecutive persistence diagrams for NSL-KDD data stream

attacks by the microcluster-based framework is given in Table 12. The accuracy of the framework for NSL-KDD data stream is shown in Table 13.

Tables 12 and 13 show that the microcluster-based framework is unable to accurately categorize the traffic in NSL-KDD data stream. The framework fails to detect the changes in traffic type in three instances ($FN = 3$). In addition, a large number of false positives results in a poor precision for the framework.

The microcluster-based framework, when compared to the sliding-window model, reduces the TPR (0.7 vs. 0.9 for both Subset 2 and NSL-KDD data streams) and increases the FPR (0.367 vs. 0.111 for Subset 2, and 0.110 vs. 0.013 for NSL-KDD data stream). This is because the microcluster-based framework only provides *indirect* control over the size and the update frequency of the data summary. As a result, it is difficult to configure the summary to accurately represent a small number of data objects from the stream. This, in turn, renders the microcluster model inaccurate for applications that can involve 'sharp' temporal changes (or, 'spikes') in the metric that is being monitored. On the other hand, the sliding-window model provides direct and precise control over the size and the update frequency of the data summary through the input parameters m and f and, thus, is more accurate for detecting 'spikes' in the stream. These observations demonstrate that the microcluster-based framework is not suitable for detecting changes of short duration.

TABLE 11 Confusion matrix obtained for the sliding-window model on NSL-KDD data stream

		Actual	
		Positive	**Negative**
Detected	**Positive**	$TP = 9$	$FP = 21$
	Negative	$FN = 1$	$TN = 1535$

TABLE 12 Confusion matrix obtained for the microcluster-based framework on NSL-KDD data stream

		Actual	
		Positive	**Negative**
Detected	**Positive**	$TP = 7$	$FP = 171$
	Negative	$FN = 3$	$TN = 1385$

7 | CONCLUSION AND SUGGESTIONS FOR FUTURE RESEARCH

This section concludes the paper by discussing some of the insights derived from studying the application of persistent homology to streaming data. Those insights subsequently lead to some suggestions for future research that will potentially advance the work examined in this paper.

7.1 | Summary of Findings

This paper presents a computational model, called the sliding-window model, for applying persistent homology to potentially unbounded real data streams. The model maintains a topology-preserving summary of the streaming data using a combinatorial structure called the complex. The model is evaluated in terms of execution time, memory usage and its ability to effectively identify topological changes in evolving data streams.

The sliding-window model is applied to the task of detection of anomalous traffic in computer networks. Based on the size of the window, the model has two different modes of operation. If a larger window is used, the model is capable of the visual identification of several types of traffic in addition to the binary categorization of the connections into normal or anomalous traffic. On the other hand, a smaller window trades the capability of visual identification of traffic for extremely fast and memory efficient processing of the stream. Based on the application and its requirements, the user may choose one of the modes of operation by setting the size of the window through an input parameter.

TABLE 13 Accuracy of the microcluster-based framework on NSL-KDD data stream

Metric	Value
$maxHeight$	20
Accuracy	0.889
Precision	0.039
TPR	0.700
FPR	0.110

The microcluster-based computational model used in the previous work [10] is capable of identifying steady or gradual concept drifts. However, if a stream exhibits abrupt changes of short duration, it is shown that the sliding-window model configured with a small window is a more accurate tool for the detection of those changes than the microcluster-based framework. While the size and the update sensitivity of the microcluster-based model can be configured to some extent with the help of its input parameters, the sliding-window model provides a more direct way to configure the data summary using the parameters m and f. As a result, the sliding-window model is able to detect changes of extremely short duration. Since the streams of network traffic can have attacks executed with very small number of TCP packets (such as R2L and U2R attacks), the study of network anomaly detection is a suitable target application for the evaluation of the sliding-window model.

7.2 | Suggestions for Future Work

In real-world applications, the data stream may evolve due to changes in its underlying probability distribution, a phenomenon known as the *concept drift*. In most cases, the concept drift causes sequential changes to the stream that can be identified by the sliding-window model. However, the model will not be effective in the event of multiple parallel concept drifts, or the simultaneous generation of data objects from different probability distributions, that produce overlapping topological features.

In other words, if the stream consists of a random mixture of multiple different types of data objects that have overlapping topological properties, the sliding-window model as described in this paper is not ideal to be directly applied to that stream. In such a case, the model can be modified to use multiple complexes instead of one. In particular, the model can have a separate simplicial complex built on each partition in the window that would be maintained concurrently to other complexes. This approach would be equivalent to maintaining multiple separate windows in memory. Each window would attempt to maintain only one type of data objects, and would have its own simplicial complex built on the points in the window. Those windows and complexes can be maintained in parallel in the memory. Since each window would consist of only one type of data points that would be different from those in the other windows, the topological structure of the complex constructed on the points in a window would be expected to be different from those of the complexes built on other windows. The multiple window or the multiple complex model can be considered as a generalization of the 'single' sliding-window model described in this paper. Alternatively, it can be thought of as an approach to maintaining multiple clusters of data points in the memory. By building and incrementally maintaining simplicial complexes on those clusters, one would summarize the stream with the help of multiple complexes instead of one. A simplified version of this approach is presented in Appendix B that provides some preliminary evidence that may motivate future research in this direction.

ACKNOWLEDGMENTS

Support for this work was provided in part by the National Science Foundation under grants ACI–1440420 and IIS–1909096.

References

1. de Silva Vin, Carlsson Gunnar. Topological estimation using witness complexes. In: Gross Markus, Pfister Hanspeter, Alexa Marc, Rusinkiewicz Szymon, eds. *Eurographics Symposium on Point-Based Graphics*, SPBG '04:157–166The Eurographics Association; 2004; Goslar, DEU.

2. Carlsson Gunnar, Ishkhanov Tigran, de Silva Vin, Zomorodian Afra. On the Local Behavior of Spaces of Natural Images. *International Journal of Computer Vision.* 2008;76(1):1–12.

3. Ghrist Robert. Barcodes: The Persistent Topology of Data. *Bulletin of the American Mathematical Society.* 2008;45(1):61–75.

4. Chan Joseph Minhow, Carlsson Gunnar, Rabadan Raul. Topology of viral evolution. *Proceedings of the National Academy of Sciences.* 2013;110(46):18566–18571.

5. Petri Giovanni, Scolamiero Martina, Donato Irene, Vaccarino Francesco. Topological Strata of Weighted Complex Networks. *PLOS ONE.* 2013;8(6):1–8.

6. Hajij Mustafa, Wang Bei, Scheidegger Carlos Eduardo, Rosen Paul. Visual Detection of Structural Changes in Time-Varying Graphs Using Persistent Homology. In: PacificVis 2018:125–134IEEE Computer Society; 2018; USA.

7. Zomorodian Afra. Fast construction of the Vietoris–Rips complex. *Computer and Graphics.* 2010;34:263–271.

8. Silva Jonathan A., Faria Elaine R., Barros Rodrigo C., Hruschka Eduardo R., de Carvalho André C. P. L. F., Gama JoḊata Stream Clustering: A Survey. *ACM Computing Surveys.* 2013;46(1):3.1–13.31.

9. Shindler Michael, Wong Alex, Meyerson Adam W.. Fast and Accurate K-Means for Large Datasets. In: :2375–2383; 2011.

10. Moitra Anindya, Malott Nicholas O., Wilsey Philip A.. Persistent Homology on Streaming Data. In: ICDMW '20:636–643IEEE; 2020; USA.

11. Edelsbrunner Herbert, Harer John. *Computational Topology, An Introduction*. American Mathematical Society; 2010.

12. Gomes Heitor Murilo, Read Jesse, Bifet Albert, Barddal Jean Paul, Gama JoṀachine learning for streaming data: state of the art, challenges, and opportunities. *ACM SIGKDD Explorations Newsletter.* 2019;21(2):6–22.

13. Seversky Lee M., Davis Shelby, Berger Matthew. On Time-Series Topological Data Analysis: New Data and Opportunities. In: :1014–1022; 2016.

14. Gidea Marian, Katz Yuri. Topological data analysis of financial time series: Landscapes of crashes. *Physica A: Statistical Mechanics and its Applications.* 2018;491:820–834.

15. Pereira Cássio MM, de Mello Rodrigo F.. Persistent Homology for Time Series and Spatial Data Clustering. *Expert Systems with Applications.* 2015;42(15–16):6026–6038.

16. Stolz Bernadette J., Harrington Heather A., Porter Mason A.. Persistent homology of time-dependent functional networks constructed from coupled time series. *Chaos: An Interdisciplinary Journal of Nonlinear Science.* 2017;27(4):047410-1–047410-17.

17. Wright Matthew. *Introduction to Persistent Homology.* (last viewed Jan 2018).

18. Chazal Frédéric, Michel Bertrand. *An Introduction to Topological Data Analysis: Fundamental and Practical Aspects for Data Scientists.* 2017.

19. Zomorodian Afra, Carlsson Gunnar. Computing Persistent Homology. *Discrete Comput Geom.* 2005;33(2):249–274.

20. Otter Nina, Porter Mason A., Tillmann Ulrike, Grindrod Peter, Harrington Heather A.. A Roadmap for the Computation of Persistent Homology. *EPJ Data Science.* 2017;6(1).

21. Forman Robin. Morse Theory for Cell Complexes. *Advances in Mathematics.* 1998;134(1):90–145.

22. Edelsbrunner Herbert, Letscher David, Zomorodian Afra. Topological Persistence and Simplification. In: FOCS '00IEEE Computer Society; 2000; Washington, DC, USA.

23. Bubenik Peter. Statistical Topological Data Analysis Using Persistence Landscapes. *The Journal of Machine Learning Research.* 2015;16(1):77–102.

24. Aggarwal Charu C., Han Jiawei, Wang Jianyong, Yu Philip S.. A Framework for Clustering Evolving Data Streams. In: VLDB 03, vol. 29: :81–92VLDB Endowment; 2003.

25. Cao Feng, Ester Martin, Qian Weining, Zhou Aoying. Density-Based Clustering over an Evolving Data Stream with Noise. In: :328–339; 2006.

26. Kranen Philipp, Assent Ira, Baldauf Corinna, Seidl Thomas. The ClusTree: indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems.* 2011;29(2):249–272.

27. Kerber Michael, Schreiber Hannah. Barcodes of Towers and a Streaming Algorithm for Persistent Homology. *Discrete & Computational Geometry.* 2018;.

28. Carlsson Gunnar, de Silva Vin, Morozov Dmitriy. Zigzag Persistent Homology and Real-Valued Functions. In: SCG '09:247–256Association for Computing Machinery; 2009; New York, NY, USA.

29. Dua Dheeru, Taniskidou Efi Karra. *UCI Machine Learning Repository.* 2017.

30. Tavallaee Mahbod, Bagheri Ebrahim, Lu Wei, Ghorbani Ali A.. A Detailed Analysis of the KDD CUP 99 Data Set. In: :1–6; 2009.

31. Lightweight Framework for Homology .

32. Ahmed Mohiuddin, Mahmood Abdun Naser, Hu Jiankun. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications.* 2016;60:19–31.

33. Aggarwal Charu C., Han Jiawei, Wang Jianyong, Yu Philip S.. A Framework for Projected Clustering of High Dimensional Data Streams. In: VLDB 04:852–863VLDB Endowment; 2004.

34. O'Callaghan Liadan, Mishra Nina, Meyerson Adam, Guha Sudipto, Motwani Rajeev. Streaming-Data Algorithms For High-Quality Clustering. In: :685–694; 2002.

35. Tasoulis Dimitris K., Adams Niall M., Hand David J.. Unsupervised Clustering In Streaming Data. In: :638–642; 2006.

36. Aggarwal Charu C., Yu Philip S.. A Framework for Clustering Uncertain Data Streams. In: :150–159; 2008.

37. Lühr Sebastian, Lazarescu Mihai. Incremental clustering of dynamic data streams using connectivity based representative points. *Data & Knowledge Engineering.* 2009;68(1):1–27.

38. Pereira Cássio M. M., Mello Rodrigo F.. Data Clustering Using Topological Features. In: :360–365; 2014.

39. Hahsler Michael, Forrest John. streamMOA: Interface for MOA Stream Clustering Algorithms2019. R package version 1.2-2.

40. Bifet Albert, Holmes Geoff, Kirkby Richard, Pfahringer Bernhard. MOA: Massive Online Analysis. *Journal of Machine Learning Research.* 2010;11:1601–1604.

41. Fasy Brittany T., Kim Jisu, Lecci Fabrizio, Maria Clement, Millman David L., Rouvreau Vincent. TDA: Statistical Tools for Topological Data Analysis2019. R package version 1.6.9.

42. Maria Clément, Boissonnat Jean-Daniel, Glisse Marc, Yvinec Mariette. *The Gudhi Library: Simplicial Complexes and Persistent Homology.* RR-8548: INRA; 2014.

43. Project The GUDHI. *GUDHI User and Reference Manual*. GUDHI Editorial Board; 3.2.0 ed.2020.

APPENDIX

A VISUAL IDENTIFICATION USING PERSISTENCE DIAGRAMS

In Section 6.1.1, Figure 6 shows the barcodes of different types of traffics in Subset 1 data stream. The barcodes demonstrate that the topological structure of each type of traffic is distinct from those of other types of traffics. In this section, Figure A1 shows the topological structures of the same instances of traffic as in Figure 6 using persistence diagrams. In addition to the persistence diagrams, Figure A1 includes histograms for the birth and death times of H_0 and H_1 features along the horizontal and vertical axes, respectively. In particular, the histogram along the horizontal axis of each plot in Figure A1 represents the counts of birth times of H_0 and H_1 features at different values of ϵ_{birth}. Similarly, the histogram along the vertical axis of each plot shows the counts of death times of H_0 and H_1 features in different bins across the range of values of ϵ_{death}. The number in the parenthesis against each legend in the persistence diagrams in Figure A1 represents the number of topological features in the corresponding homology group.

From the persistence diagrams and histograms, it is evident that the instances of the same type of traffic have similar topological structures whereas the topology of different types of traffic are different.

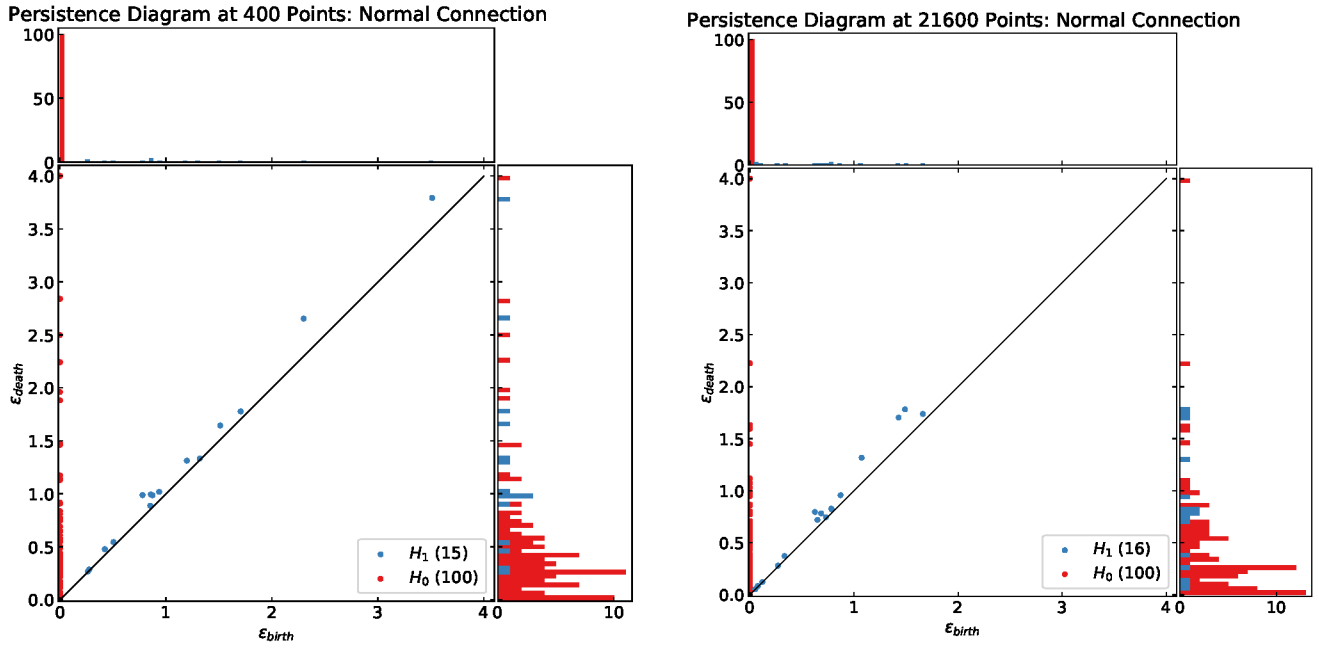

FIGURE A1 Identification of traffic types in Subset 1 data stream using persistence diagrams

B PERSISTENT HOMOLOGY AND DATA STREAM CLUSTERING

This section presents some preliminary evidence that may motivate the development of a multiple window or multiple complex model in future. The sliding-window model, as described in Section 4, can have multiple partitions in the window that represent clusters of different types of data objects. However, a single simplicial complex is constructed on all the data objects in the window. A future research endeavor may explore the possibility of maintaining multiple separate complexes concurrently on the

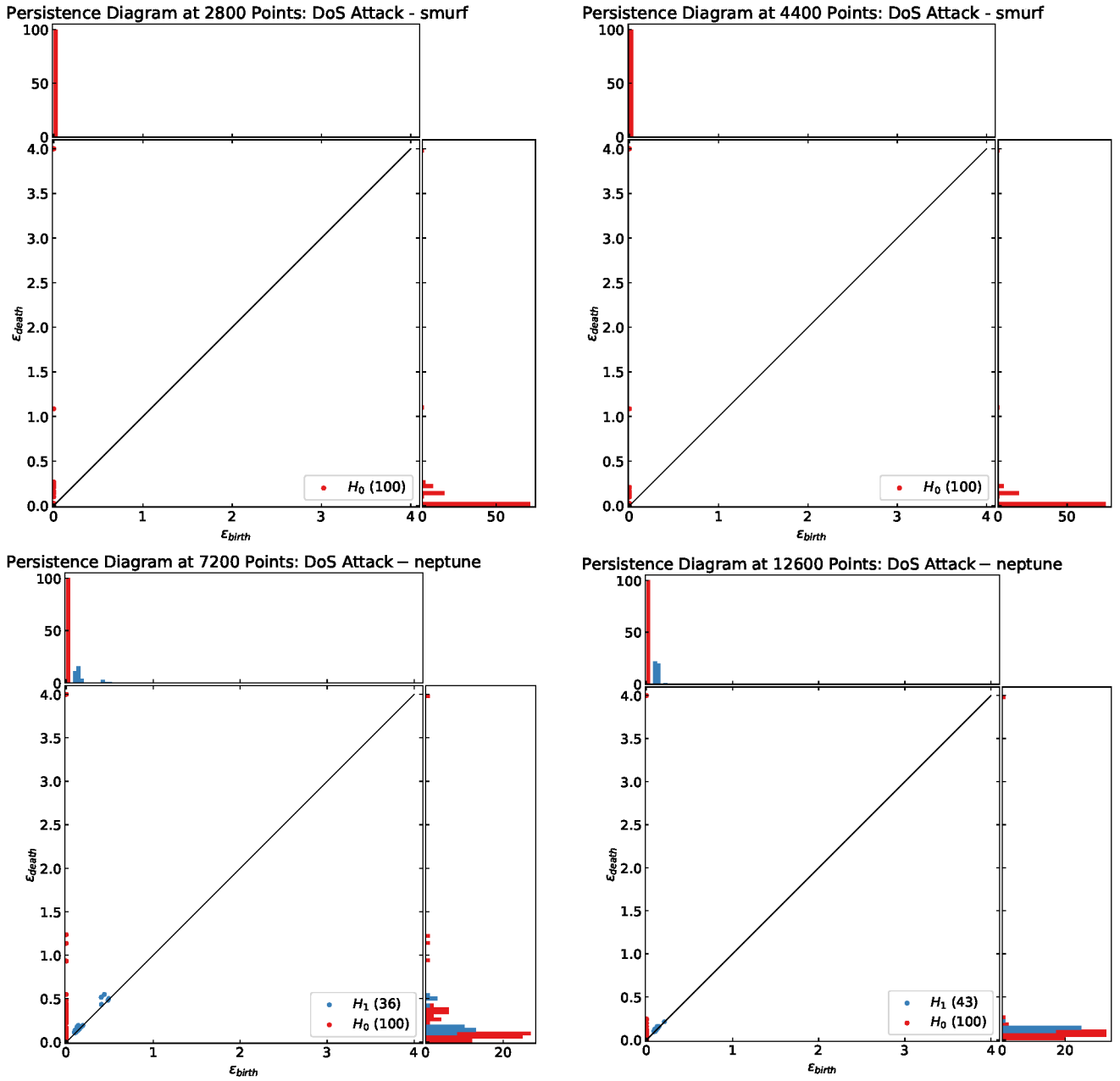

FIGURE A1 Identification of types of traffic in Subset 1 data stream using persistence diagrams (cont.)

different partitions in the window. Equivalently, one could maintain multiple separate windows or clusters of data points with a separate complex built on each of them. Such a multiple complex or multiple window model would extend the applicability of the current sliding-window model to data streams that consist of random mixtures of different types of data objects. This section presents some preliminary evidence that suggests that persistent homology computed on different clusters of data objects can help identify the *types* of those clusters.

A data stream S that consists of a random mixture of different types of data points is generated by randomly shuffling the TCP connection records in the 10% sub-sampled set of the KDD Cup 1999 data stream. Even though there are a total of 23 types of traffic (including normal connections) in the 10% sub-sampled set, the data stream has a high class imbalance. Out of the 494, 021 records, only three types of traffic ('neptune': 107, 201, 'smurf': 280, 790, and normal: 97, 278) constitute 485, 269 of the records.

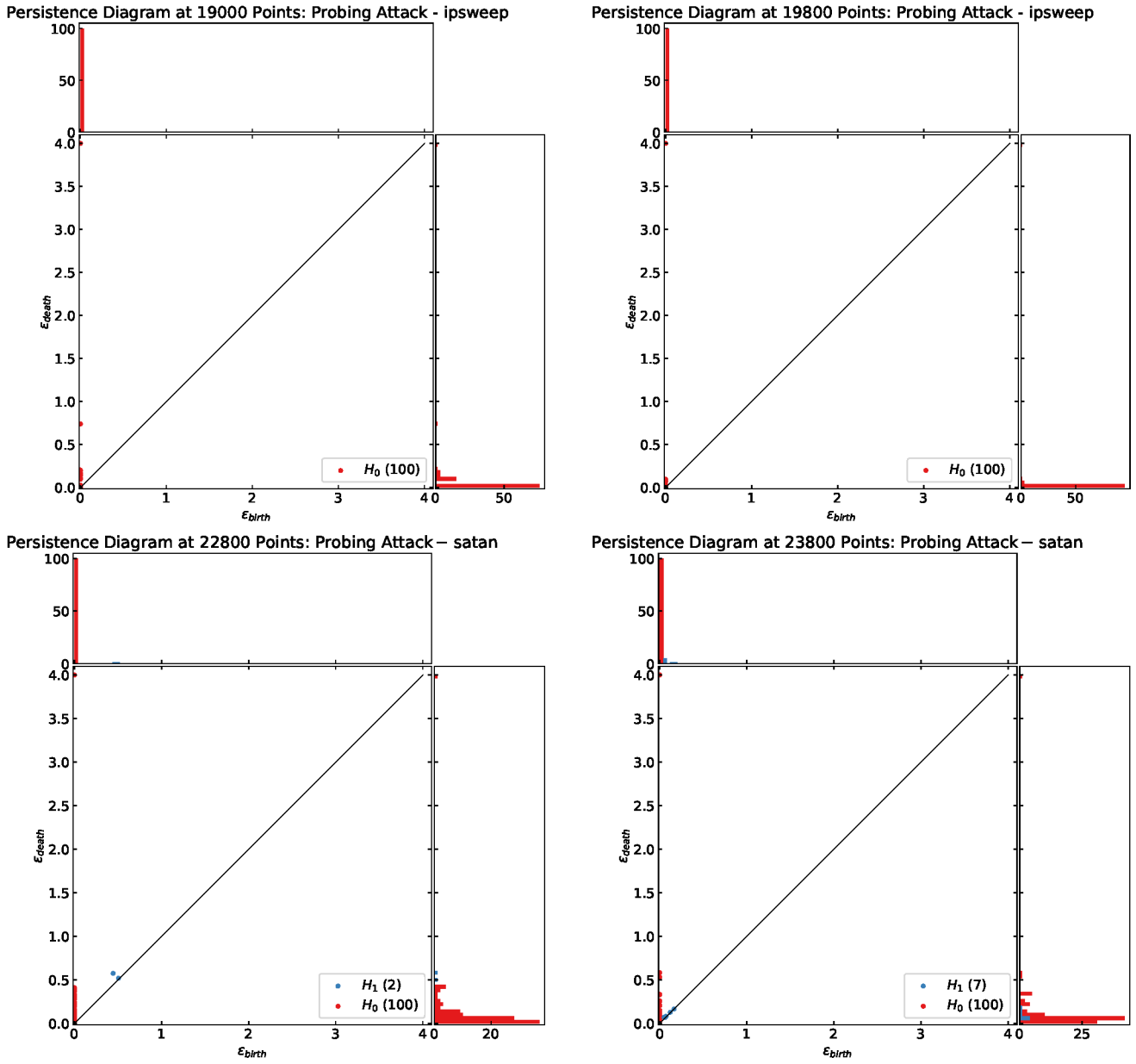

FIGURE A1 Identification of types of traffic in Subset 1 data stream using persistence diagrams (cont.)

The stream clustering algorithm DenStream[25] was applied to the data stream S, and is evaluated on the first 20,000 points from the stream. DenStream was able to achieve a median Adjusted Rand Index of 0.896 and a median purity of 0.935 over all the *horizons* of 1,000 data points the algorithm was evaluated on. DenStream, however, detected only 3 clusters out of the $k = 14$ clusters present in the first 20,000 data points in S. This was due to the class imbalance in the input set with only three types of traffic ('neptune': 4,321, 'smurf': 11,391, and normal: 3,932) comprising 19,644 records out of the first 20,000 points in S.

A random sample of 100 data points was selected from each of the 3 clusters detected by DenStream, and then persistent homology was computed on each of those samples. Figures B2 and B3 show the output of persistent homology, displayed as persistence diagrams and barcodes, respectively. In addition to the persistence diagrams, Figure B2 includes histograms for the birth and death times of H_0 and H_1 features along the horizontal and vertical axes, respectively (see Appendix A for details). By looking at the barcodes, persistence diagrams, or the histograms, it is possible to identify the types of traffic in the clusters

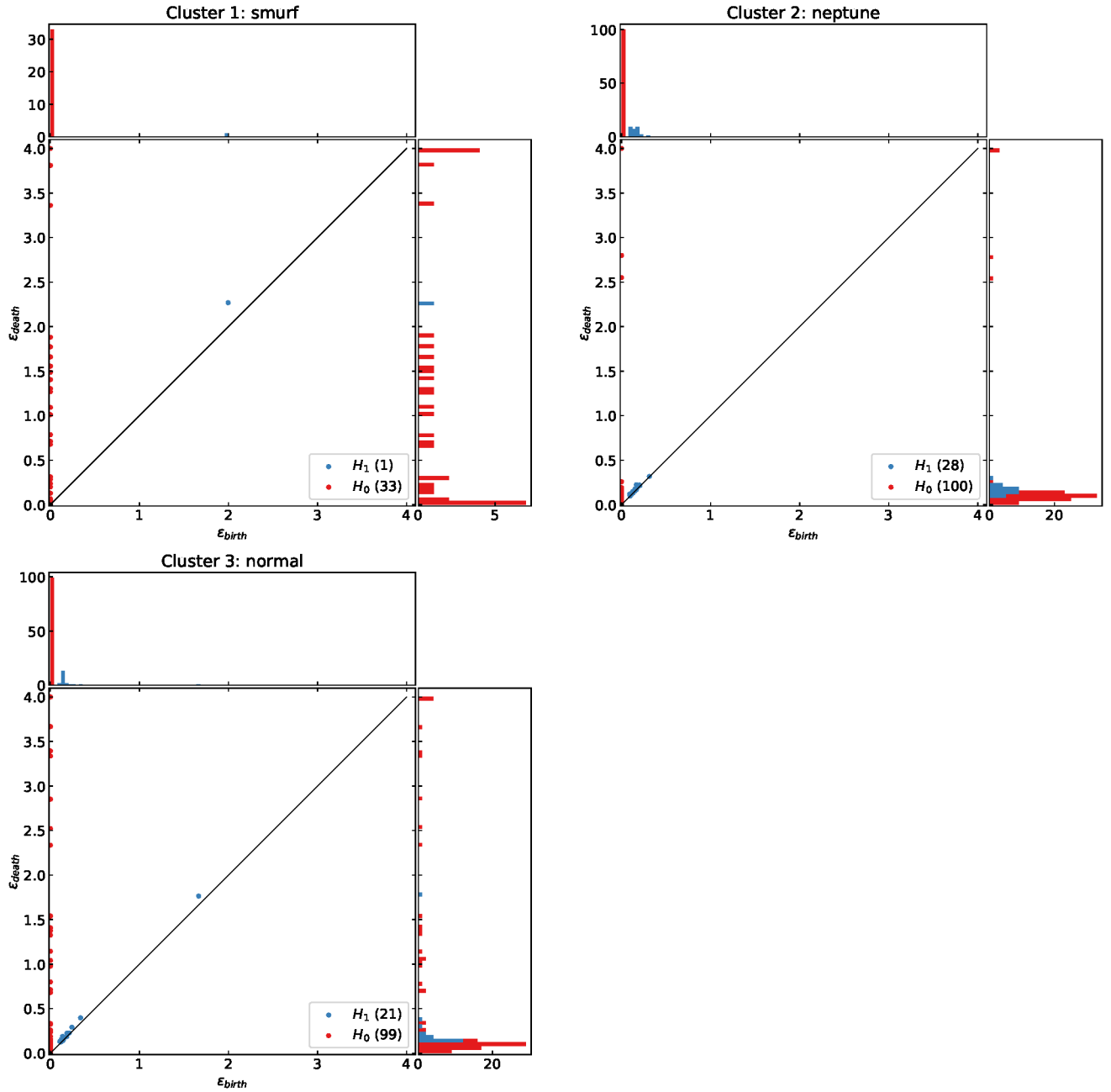

FIGURE B2 Identification of clusters, generated by stream clustering, using persistence diagrams

generated by DenStream. For instance, a comparison between the plots in Figure 6 and Figure B3 suggests that the clusters 1, 2, and 3 primarily contain 'smurf', 'neptune', and normal traffic, respectively.

Since the complexes built on different clusters or partitions of data points have topological structures that are different from one another, persistent homology computed on those complexes can be able to identify the types of the clusters. Whenever a new data point $\mathbf{x}$ arrives from the stream, certain conditions are checked to determine if $\mathbf{x}$ is similar to the points in one of the existing partitions. If such an existing partition is found, $\mathbf{x}$ is discarded. Otherwise, a new partition is created with only $\mathbf{x}$ in it. By keeping track of which of the existing partitions is similar to each new data point $\mathbf{x}$, it is possible to detect any changes in the stream. For instance, if the last n incoming points were similar to the points in an existing partition $\mathcal{P}$, the stream did not go through any change during that time. However, if the next point $\mathbf{x}$ is similar to the points in another existing partition $\mathcal{P}_o \neq \mathcal{P}$, the new point $\mathbf{x}$ can imply the advent of an upcoming concept drift. Similarly, if $\mathbf{x}$ is not similar to the points in any of the

existing partitions, it can either be an outlier or the beginning of a concept drift. As each partition of data points would have its own complex maintained separately from other complexes, the multiple complex model would be able to deal with data streams that have different types of data objects with overlapping topological properties shuffled together.

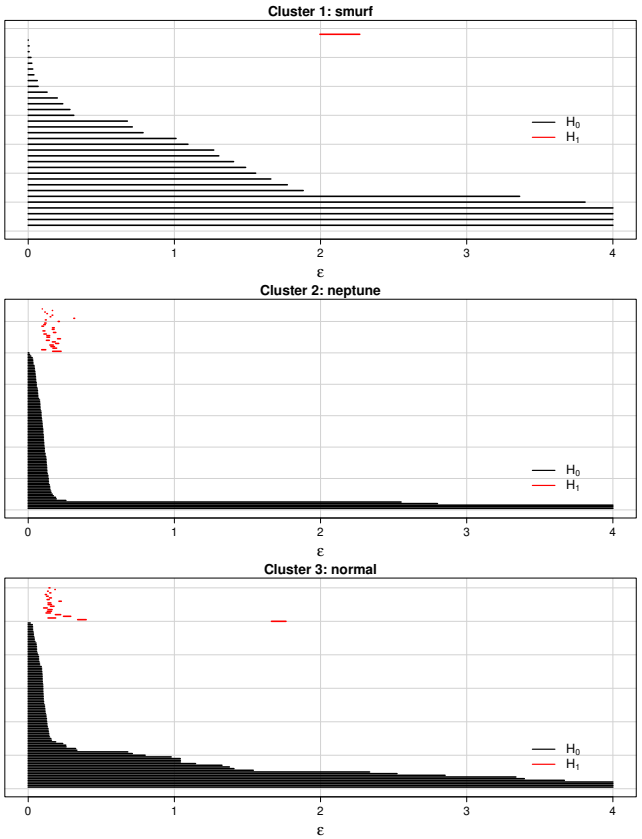

FIGURE B3 Identification of clusters, generated by stream clustering, using barcodes