Abstract
Persistent homology has recently emerged as a powerful technique in topological data analysis for analyzing the emergence and disappearance of topological features throughout a filtered space, shown via persistence diagrams. Additionally, (co)sheaves have proven to be powerful instruments in tracking locally defined data across global systems, resulting in innovative applications to network science. In this paper, we combine the topological results of persistent homology and the quantitative data tracking capabilities of cosheaf theory to develop novel techniques in network data flow analysis. Specifically, we use cosheaf theory to construct persistent homology in a framework geared towards assessing data flow stability in hierarchical recurrent networks (HRNs). We use cosheaves to link topological information about a filtered network encoded in persistence diagrams with data associated locally to the network. From this construction, we use the homology of cosheaves as a framework to study the notion of “persistent data flow errors.” That is, we generalize aspects of persistent homology to analyze the lifetime of local data flow malfunctions. We study an algorithmic construction of persistence diagrams parameterizing network data flow errors, thus enabling novel applications of statistical methods to study data flow malfunctions. We conclude with an application to network packet delivery systems.

1 Introduction
Applications of (co)sheaves and sheaf cohomology to network science have recently emerged in the burgeoning field of topological data analysis from the works of Ghrist [7] and Curry [5]. The motivation is that sheaves provide a link between local and global data associated to portions of a topological space, enabling one to extract useful information pertaining to information flow from sheaves on a network. A fruitful application of (co)sheaf theory is sheaf cohomology, or dually cosheaf homology. This has previously been used in network
analysis with an information-theoretic incentive [7] to study information flow via network coding sheaves.

Additionally, persistent homology has surfaced as a powerful computational and algebraic tool in topological data analysis to study the emergence and disappearance of topological features across a filtered topological space. Its applications and uses are wide ranging. See for example [1, 4, 6] and the recent book [8]. A particularly useful construction in persistent homology is that of a persistence diagram, which provides a visual depiction of the lifetime of topological features in the extended real half plane.

In this paper, we consider an approach to analyzing data flow in hierarchical recurrent networks (HRNs) that combines both the topological approach of persistent homology and the quantitative information tracking abilities of cosheaves. Neural networks carrying the topological structure of a HRN have a variety of applications ranging from speech recognition [9] to image classification [11]. Our main results generalize techniques in persistent homology to study the notion of "persistent" data flow malfunctions in HRNs.

The paper is organized as follows. Sections 2 and 3 provide an introduction to HRNs, our main object of study which will be used to model data flow in a network. Section 4 gives a cosheaf theoretical construction of the notions in the previous sections and develops an application of homology to study data flow errors, as well as an algorithmic construction of generalized persistence based on results from Section 3. We conclude in Section 5 with a discussion of applications to complex networks.

2 Preliminaries

In this section, we establish the terminology and notation used throughout the body of the paper.

2.1 Hierarchical Recurrent Networks

Let \( \mathbb{N} = \{0, 1, 2, 3, \ldots \} \). We will use standard graph theory language. Our reference for the basics of graph theory is [10].

By a directed cycle, we mean a cycle on \( k \geq 3 \) vertices \( v_1, \ldots, v_k \) with directed edge given by \( v_i \rightarrow v_{i+1} \) for all \( 1 \leq i \leq k-1 \) and \( v_k \rightarrow v_1 \).

**Definition 2.1** Let \( G \) be a directed path on \( 2m + 1 \) vertexes \( v_1, \ldots, v_{2m+1} \) with directed edge \( v_{2i} \rightarrow v_{2i-1} \) and \( v_{2i} \rightarrow v_{2i+1} \) for each \( 1 \leq i \leq m \). If \( H^1, \ldots, H^m \) are any \( m \) directed cycles, a Hierarchical Recurrent Network (HRN) \( H \) is constructed by identifying a directed edge in cycle \( H^i \) with edge \( v_{2i} \rightarrow v_{2i-1} \) such that the orientations are consistent. Each \( H^i \) is called the \( i \)th recurrent subprogram of \( H \), or sometimes just subprogram.
Example As an example to illustrate Definition 2.1, the directed graph $H$ below is a HRN with $m = 3$.

The $i^{th}$ subprogram, $1 \leq i \leq 3$, is given by

- $H^1 : v_1 \rightarrow w_1 \rightarrow v_2 \rightarrow v_1$
- $H^2 : v_3 \rightarrow w_2 \rightarrow w_3 \rightarrow v_4 \rightarrow v_3$
- $H^3 : v_5 \rightarrow w_4 \rightarrow v_6 \rightarrow v_5$.

2.2 Representation of HRN Data Flow

Although the bare structure of an HRN can be conveniently encoded as a directed graph, we need more structure to capture the notion of data flow inside an HRN as well as error detection. In order to accomplish this, we will make use of a representation of an HRN; i.e associating vector spaces to each node of an HRN and linear maps to the directed edges to represent the magnitudes of data that they contain.

Definition 2.2 Let $H$ be a HRN, and $H^n$ be the $n^{th}$ recurrent subprogram of $H$. Suppose $H^n$ has vertex set \{v_1, v_2, ..., v_k\} and direction \{e_i : v_i \rightarrow v_{i+1}\} and $e_k : v_k \rightarrow v_1$, and for each $n$, let $\sigma_i^n = \sigma_i$ be nonnegative integers, $1 \leq i \leq k$. To each $v_i \in H^n$, we assign a vector space $\lambda_i^n$ over a field $k$ such that $\dim(\lambda_i^n) \leq \sigma_i$. The value $\sigma_i$ is the capacity of vertex $v_i$.

Additionally, each directed edge $e_i$ is assigned a linear map $\phi_i : \lambda_i^n \rightarrow \lambda_{i+1}^n$, with $\phi_i$ a $\dim(\lambda_i^n) \times (\dim(\lambda_{i+1}^n) + \ell_n)$ matrix for a fixed $\ell_n \in \mathbb{N}$. This constitutes a quiver representation $\lambda^n$ of $H^n$. When there is no chance of confusion, we refer to the quiver representation of a subprogram as a subprogram.

Example Let $H$ be as in Example 2.1. We choose capacities $2, 3, 2$ for $H^1$, capacities $3, 5, 5, 0$ for $H^2$ and capacities $3, 3, 3$ for $H^3$ along with $\ell_1 = 2, \ell_2 = 0$, and $\ell_3 = 1$. Letting $\lambda(i)$ denote a vector space of dimension $i$, a quiver representation of $H^1, H^2$, and $H^3$ is given by

$$
\begin{array}{c}
\lambda(5) \\
\lambda(3) \\
\end{array} \quad \begin{array}{c}
\lambda(0) \rightarrow \lambda(0) \\
\lambda(0) \rightarrow \lambda(0) \\
\lambda(1) \rightarrow \lambda(3) \\
\end{array} \quad \begin{array}{c}
\lambda(2) \\
\lambda(0) \\
\end{array}
$$
respectively. The idea here is that data (in the form of vector spaces) is cycled through $H^n$, and each linear map “adds” data to its domain vector space (i.e increases its dimension by a value of $\ell_n$). Notice that the single occurrence of 0 in the capacities for $H^2$ forces all vector spaces to be 0-dimensional as well as $m_2 = 0$.

We now wish to allow the data to cycle through $H^n$ multiple times. We can accomplish this by enhancing the quiver representation $\lambda^n$ by first defining a singular data flow and using it to more generally define the concept of data flow.

**Definition 2.3** Let $H^n$ be a subprogram on $k$ vertices in an HRN $H$ and let $\lambda^n$ be its associated quiver representation. A singular data flow is the sequence of vector spaces $\lambda^n_1, \lambda^n_2, \ldots, \lambda^n_k$ of $\lambda^n$.

Note that Definition 2.3 does not include the linear transformations between vector spaces since a singular data flow represents the travel of data through the vertices in $H^n$.

We can now define a general data flow, i.e a data flow built from multiple singular data flows.

**Definition 2.4** Let $H^n$ be a subprogram on $k$ vertices in an HRN $H$ and let $\lambda^n$ be its associated quiver representation, $m_n = m$ a positive integer. A data flow $D^n_m$ over $\lambda^n$ is a sequence of vector spaces $\lambda^n_{1,1}, \lambda^n_{2,1}, \ldots, \lambda^n_{k,1}, \lambda^n_{1,2}, \lambda^n_{2,2}, \ldots, \lambda^n_{m,2}, \ldots, \lambda^n_{m,k}$.

Note that many data flows can be associated to a single subprogram $\lambda^n$.

**Remark 2.5** Since index $m$ is the total number of times that the data flow circulates through the cycle $H^n$, it is clear that a singular data flow is the same as a data flow which iterates through the cycle exactly once.

A data flow $D^n_m$ in $\lambda^n$ can be described by the following diagram of vector spaces:
The sequence of vector spaces in each column represents a singular data flow and the linear map \( \beta_{n,k,q} : \lambda_{n,k,q} \rightarrow \lambda_{n,k,q+1} \) “connects” each singular data flow together that makes up the data flow.

Although it is not of great relevance to the following sections, we may consider a representation \( \lambda \) of the entire HRN \( H \) once we are given representations \( \lambda^o \) of all the cycles \( H^o \). Note that there is exactly one directed edge between the “last” vertex in cycle \( H^i \) and the “first” vertex in cycle \( H^{i+1} \). We will impose that no data addition occurs from the representation \( \lambda^i \) restricted to its last vertex and of \( \lambda^{i+1} \) restricted to its first vertex, i.e the two vector spaces are equal (this makes sense because we think of the linear maps in \( \lambda^o \) as “adding data” by mapping its domain to a vector space of potentially higher dimension). We then require that the representation of the edges that are not part of a cycle are identity maps.

3 Data Flow Control and Errors

For the following definitions, fix a data flow \( D^n_m \) for each subprogram representation \( \lambda^n \).

Definition 3.1 To each \( \lambda^n \), assign a natural number \( \delta^n \), the desired data output. For any data flow \( D^n_m \) of \( \lambda^n \), the final data dimension of \( \lambda^n \) with respect to \( D^n_m \) is \( \theta^n := \dim(\lambda^n_{k,m}) \). We sometimes say “final data output of \( \lambda^n \)” with \( D^n_m \) understood.
The desired data output is the ideal output of information that we would like to accumulate within the subprogram while the final data dimension is the maximum possible amount of information that any particular data flow can output. We are thus interested in whether or not any particular data flow has the capacity to accumulate the ideal amount of information for the subprogram.

Definition 3.2 A subprogram \( \lambda^n \) is faulty with respect to \( D^n_m \) (or just faulty) if \( \theta^n < \delta^n \) while a subprogram is able with respect to \( D^n_m \) (able) if \( \theta^n > \delta^n \). If \( \lambda^n \) is faulty, define \( \tilde{\lambda}^n := k^{\delta^n - \theta^n} \) where \( k \) denotes the ground field. Similarly, if \( \lambda^n \) is able, define \( \hat{\lambda}^n := k^{\theta^n - \delta^n} \). We will occasionally replace the phrase “\( \lambda^n \) is faulty” with “a data flow error occurs in \( \lambda^n \)” and replace the phrase “\( \lambda^n \) is able” with “a data flow fix occurs in \( \lambda^n \).” For completeness, we call a subprogram sufficient with respect to \( D^n_m \) (sufficient) if \( \theta^n = \delta^n \).

The point of these definitions is that data flow deficits created by faulty subprograms can be compensated for based on data surpluses produced by subsequent able subprograms. We will use the tools of cosheaf theory to express this mathematically and to create a framework in which one can systematically analyze these data flow errors as “fixes,” in the same way one would study the life and death of holes in a space using persistent homology.

Example From the three subprograms \( \lambda^1, \lambda^2, \) and \( \lambda^3 \) in Example 2.2, we obtain three singular data flows \( D^1_1, D^2_1, \) and \( D^3_1 \) with \( \theta^1 = 7, \theta^2 = 0, \) and \( \theta^3 = 3 \). If we choose desired data outputs \( \delta^1 = 1, \delta^2 = 2, \) and \( \delta^3 = 3 \), then we see that \( \lambda^1 \) is able, \( \lambda^2 \) is faulty, and \( \lambda^3 \) is sufficient. Note that \( \lambda^2 \) can never be able.

Note that one can determine if a singular data flow is faulty or able if one knows the initial value \( \dim(\lambda^n_1), \ell_n \), and the number of vertices \( k \).

4 Main Results

We first recall some relevant information about cosheaves. This is a very brief exposition; we refer the reader to [5, 2, 3, 8] for a more in-depth introduction to cosheaf theory and homology.

4.1 Cosheaves

Definition 4.1 Let \( X \) be a topological space and \( J \) an abelian category. A \( J \)-valued precosheaf \( F \) on \( X \) is a covariant functor \( F : \text{Open}(X) \to J \) from the category of open subsets of \( X \) and inclusion maps to \( J \). If \( U \subset X \), an element \( x \in F(U) \) is a cosection of \( F \) over \( U \). For a pair of embedded open subsets \( V \subset U \subset X \), the induced map on the inclusion \( F(V) \to F(U) \) is called the corestriction map. A precosheaf \( F \) on \( X \) is a cosheaf if for any open \( U \subset X \) and any open cover \( \{U_i\} \) of \( U \), the following sequence is exact: \( \bigoplus \bigoplus F(U_i) \to F(U) \to 0 \).

In this section, our domain space for cosheaves will be the real line \( \mathbb{R} \) endowed with the standard topology. The subprograms \( \lambda^n \) will be represented by points.
$n \in \mathbb{Z} \subseteq \mathbb{R}$. We will construct presheaves $G_1, G_2 : \text{Open}(\mathbb{R}) \to \text{Vect}_k$ as follows. For $U \subset V$, let $p_U$ denote the largest $n \in U$ that is the index of a subprogram. If $\lambda^n$ is faulty, then $G_1(U) = \lambda^{p_U}$. If $\lambda^n$ is able, then $G_1(U) = 0$. Hence $G_1$ records the margins of error in the maximal point of $U$. We now define $G_2$, which records the margins of surplus data of the maximal point of $U$. If $\lambda^n$ is able, then $G_1(U) = \lambda^{p_U}$. If $\lambda^n$ is faulty, then $G_1(U) = 0$.

**Definition 4.2** Let $X$ be a topological space and $\mathcal{U} = \{U_i\}$ an open cover of $X$, and $F$ a presheaf of abelian groups on $X$. The group of Čech $k$-chains associated to $\mathcal{U}$ is the group $C_k(\mathcal{U}, F) = \bigoplus_i F(U_{0,1,...,k})$, where $U_{0,1,...,k} = \bigcap_{i=0}^k U_i$.

Equipped with differentials $\partial_k : C_k(\mathcal{U}; F) \to C_{k-1}(\mathcal{U}; F)$, we obtain a Čech complex $C_*(\mathcal{U}; F) = 0 \to C_k(\mathcal{U}; F) \to C_{k-1}(\mathcal{U}; F) \to \ldots \to C_0(\mathcal{U}; F) \to 0$. We denote the $n^\text{th}$ Čech homology group associated to $F$ and covering $\mathcal{U}$ by $H_n(\mathcal{U}; F)$.

### 4.2 Main Results

In this section, we state our results generalizing notions in the persistent homology theory of filtered topological spaces to studying network data flow security.

We present a theorem allowing for a homological description of the emergence of data flow errors and resolutions. This allows us to construct error $P$-intervals, which parametrize the lifetime of network data flow malfunctions.

We will make use of a specific covering of $\mathbb{R}$, which we now describe:

Let $\mathcal{U} = \{U_i\}$ be a (co)filtered open covering of $\mathbb{R}$ with $U_m \subset U_{m-1} \subset \ldots \subset U_0 = \mathbb{R}$ and $k \in \mathbb{R}$, if and only if $r \leq k$. From now on, $\mathcal{U}$ will denote this specific covering throughout the paper.

Let $T[n]$ be the shift endofunctor over $\text{Ch}(\text{Vect}_k)$ (shifts indexes of a chain complex up $n$). Also, let $\beta_{>k}$ denote the brutal truncation endofunctor on $\text{Ch}(\text{Vect}_k)$, trivializing all terms indexed $j > k$. Let $\varphi^k_n : \beta_{>k} \circ \beta_{<k}(C_k(\mathcal{U}; G_n)) \to \beta_{>k} \circ \beta_{<k} T[n](\hat{C}_{k-1}(\mathcal{U}; G_n))$ be the projection.

**Theorem 4.3** A data flow error (resp. fix) occurs in the $k^\text{th}$ subprogram of $\lambda^*$ if and only if $H_k(\text{Cone}(\varphi^k_n))$ is non trivial for $n = 1$ (resp. $n = 2$).

**Proof** Let $\hat{C}_k(\mathcal{U}; G_n)$ and $\hat{C}_{k-1}(\mathcal{U}; G_n)$ denote the domain and codomain of $\varphi^k_n$, respectively. Clearly, $H_k(\text{Cone}(\varphi^k_n)) \simeq \ker(\varphi^k_n)$. Since every exact sequence in $\text{Vect}_k$ splits, the exact sequence: $0 \to \ker(\varphi^k_n) \to \hat{C}_k(\mathcal{U}; G_n) \xrightarrow{\varphi^k_n} \hat{C}_{k-1}(\mathcal{U}; G_n) \to 0$ splits and we obtain that $\ker(\varphi^k_n) \simeq \hat{C}_k(\mathcal{U}; G_n) / \hat{C}_{k-1}(\mathcal{U}; G_n)$. From the cofiltered nature of $\mathcal{U}$, it holds that $\hat{C}_k(\mathcal{U}; G_n) / \hat{C}_{k-1}(\mathcal{U}; G_n)$ yields the amount of data flow deficit (resp. surplus) created at subprogram $k$ for $n = 1$ (resp. $n = 2$). Thus, a non trivial $H_k(\text{Cone}(\varphi^k_n))$ reveals the emergence of a network data flow error or resolution for $n = 1$ or $n = 2$, respectively. The converse is trivial. \qed
4.3 “Persistent Homology” for Data Flow Errors

In this section, we will provide motivation for the subsequent subsections by providing basic definitions regarding persistent homology. The next subsections will use the results of the previous subsection (a homological description of data flow errors and fixes) to formulate an application of persistent homology to study the birth and death of data flow errors (a data flow fix is seen as the death of an error).

Basically, persistent homology is a tool which one uses to study the birth and death of topological features in a filtered space. Formally:

**Definition 4.4** Let $X$ be a filtered topological space, i.e a space $X$ equipped with a sequence of nested subspaces $X_0 \subset X_1 \subset \ldots \subset X_n = X$. Let $H_n(X)$ denote the $n^{th}$ singular homology (with coefficients in a field $k$) vector space of $X$. Fix indexes $i$ and $j$ with $i \leq j$. The $(i, j)$-persistent $n^{th}$ homology vector space $H_{i,j}^n(X)$ is defined as $H_{i,j}^n(X) = \text{im}(H_n(X_i) \to H_n(X_j))$, where $H_n(X_i) \to H_n(X_j)$ is induced by the inclusion $X_i \subset X_j$.

**Remark 4.5** The dimension of the $k$-vector space $H_{i,j}^n(X)$ is the number of $n$-dimensional holes present in subspace $X_i$ that are also present in $X_j$.

**Example** If $i = j$, then it is clear that $H_{i,j}^n(X) = H_n(X_i)$ since the $k$-linear map induced by $\text{id} : X_i \to X_i$ must be the identity map on $H_n(X_i)$.

**Definition 4.6** Let $X$ be a filtered topological space. Let $\mathbb{Z}_\infty$ denote the set $\mathbb{Z} \cup \{\infty\}$. A degree $n$ persistence diagram is a multiset over $\mathbb{Z}_\infty \times \mathbb{Z}_\infty$, where the multiplicity $\mu(i, j)$ of a point $(i, j) \in \mathbb{Z}_\infty$ is $\dim(H_{i,j}^n(X))$. The points parametrize the lifetimes of homology classes (“holes”) in $X$, i.e the presence of a point $(i, j)$ means that a hole was created at index $i$ in the filtration and destroyed at index $j$. The multiplicity function records how many holes were created and destroyed for each interval.

**Definition 4.7** The intervals $(i, j)$ in the persistence diagram are called homological $P$-intervals, or simply $P$-intervals.

**Definition 4.8** A point $(i, j)$ is called proper if $j < \infty$ and a point at infinity otherwise. A proper point indicates the presence of holes in $X$ that are born and die within the “bounds” of the filtration. An improper point indicates the presence of a hole that is created but never dies.

4.4 Error $P$-intervals

We can adapt the notion of a $P$-interval from persistent homology to the notion of an error $P$-interval. Instead of parametrizing the lifetime of a hole in a filtered topological space, an error $P$-interval parametrizes the lifetime of a data flow error in an HRN.
We can now give an algorithmic description of $P$-intervals which parametrize the lifetimes of network data flow errors. First, we fix some notation. Let $S^1$ denote the set of indexes of all the subprograms in $\lambda$ that are faulty. Let $S^2$ denote the set of indexes of all the subprograms in $\lambda$ that are able.

**Data:** $S^n$ for $n \in \{1, 2\}$ //Lists of indexes of subprograms with nontrivial error and fix contributions for $n = 1$ and $n = 2$, respectively

**Result:** Set of $P$-intervals $\mathcal{P} \subset \mathbb{Z} \times (\mathbb{Z} \cup \infty)$

while $i \leq \max(S^1)$ do
  
  { Starting from $i = \min(S^1)$
    
    //pairs index of subprogram in which network error is created (in $S^1$) with index in which error is resolved (in $S^2$)
  }
  
  Assign to $i \in S^1$ the least $k \in S^2$ not already chosen such that $\dim(\ker(\phi^1_i)) = \dim(\ker(\phi^2_k))$;

  increment $i$ by one;

  if $|S^1| = |S^2|$ then
    $\mathcal{P} = \{(i, k)\}$, where $i$ is assigned to $k$;
  else
    $\mathcal{P} = \{(i, k)\} \cup \{(S^1 \setminus S^2) \times \infty\}$ //Accounts for points at infinity;

end

Algorithm 1 assigns each error index from $S^1$ to its “fixing” index in $S^2$ based on availability of space in able subprograms. Thus, the output $\mathcal{P}$ can be thought of as a persistence diagram, since a point $(i, j) \in \mathcal{P}$ indicates that a data flow deficit occurs in subprogram $\lambda^i$ but $\lambda^j$ has enough data flow surplus to completely account for the deficit created.

The set of assignments constitute proper points, i.e they indicate the presence of faulty subprograms whose data flow deficit is accounted for by a subsequent surplus. Additionally, elements in $S^1$ that do not have “fixing” indexes in $S^2$ are paired with $\infty$, constituting points at infinity.

### 4.5 Persistent Error Homology Groups

We can now provide a construction of persistent error homology groups as an analogue to the persistent homology groups of a filtered topological space. This is due to the observation that a persistence diagram $\Dgm(X)$ determines completely determines the persistent homology groups $H^{i,j}_k$ of a filtered topological space $X$, concretely stated by the following: $\dim(H^{i,j}_k(X)) = \sum_{u,v; u \leq i, v \leq j} \mu(u, v)$, where $\mu : \Dgm(X) \ni (i, j) \rightarrow \mathbb{N}$ is the multiplicity function. We adjust this statement to define persistent error homology groups of an HRN given a persistence diagram generated by Algorithm 1. We can define the $(i,j)$ - persistent error homology group $H^{i,j}(\mathcal{Y}_{\lambda^*})$ of $\mathcal{Y}_{\lambda^*}$ by $\dim(H^{i,j}(\mathcal{Y}_{\lambda^*})) = \sum_{u,v; u \leq i, v \leq j} \mu(u, v)$, where $(i, j) \in \Dgm_{\mathcal{Y}_{\lambda^*}}$, and $j < \infty$. In this context, the multiplicity $\mu(i, j)$ of a proper point $(i, j)$ records the dimension of data flow deficit created at index $i$. 

9
and resolved at index \( j \).

## 5 Complex Networks Application

In this section, we state the applications of our results to complex networks, along with explicitly describing an application to Internet package delivery systems.

The most fruitful applications of this work stem from the results generalizing concepts in persistent homology to study problems in network science. Particularly, our technique to construct error persistence diagrams and providing a visual parametrization of the lifetime of data flow errors in HRNs creates a framework for the novel application of statistical tools to assess data flow malfunctions recorded in persistence diagrams. Now results from such an assessment would have implications not in the topological features of a network (as have been shown previously), but in the nature of its data flow.

From immediate inspection of an error persistence diagram, one can see the significance of a single data flow error to overall network security. This is based on the distance from the point to the diagonal of \( \mathbb{Z}_\infty \times \mathbb{Z}_\infty \). An accumulation of points along the diagonal indicates data flow “noise,” as opposed to points far from the diagonal, which show the presence of significant data flow malfunctions within the network. More interestingly, extended metrics on the space of (error) persistence diagrams can be used to evaluate similarities between instances of data flow across the same network, as we demonstrate below.

Let \( \mathcal{V}_{\lambda_1^*} \) and \( \mathcal{V}_{\lambda_2^*} \) be two instances of data flow in an HRN represented in \( \text{Vect}_k \) by \( \lambda_1^* \) and \( \lambda_2^* \), respectively. By instances of data flow, we mean two different situations in which data flows through the same network. Practically, this situation emerges if the HRN in question acts as a model for a frequently occurring process. Let \( \tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_1^*}} \) and \( \tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_2^*}} \) be the error persistence diagrams of \( \mathcal{V}_{\lambda_1^*} \) and \( \mathcal{V}_{\lambda_2^*} \), respectively. Let \( \varphi: \tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_1^*}} \rightarrow \tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_2^*}} \) be a bijection of persistence diagrams. Recall that the bottleneck distance \( d_B(\tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_1^*}}, \tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_2^*}}) \) between persistence diagrams \( \tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_1^*}} \) and \( \tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_2^*}} \) is defined as

\[
    d_B(\tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_1^*}}, \tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_2^*}}) = \inf_{\varphi} \sup_p \| p - \varphi(p) \|_\infty,
\]

where \( p \in \tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_1^*}} \). The bottleneck distance functions as an extended metric to compare similarities between persistence diagrams, with a lesser distance indicating greater similarities in the distribution of points throughout each diagram. In our network theoretic context, \( d_B(\tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_1^*}}, \tilde{\text{Dgm}}_{\mathcal{V}_{\lambda_2^*}}) \) is an indicator of the differences between the persistence of data flow errors between \( \mathcal{V}_{\lambda_1^*} \) and \( \mathcal{V}_{\lambda_2^*} \).

As a toy example of an application of these techniques, consider the following portion of the HRN \( H \) defined in Example 2.1 above which models a recurrent
network packet delivery system (i.e a packet-switched network arranged in a HRN structure). The output of our techniques can be seen as a framework to statistically study error-detection and correction.

The nodes of \( H \) represent the location of delivery sites and the directed edges model the flow of network packet data from one node to another. We will illustrate our procedure on this system. Since this network is quite small, it does not fully illustrate the capabilities of our techniques (which are most powerful in large HRNs), but describing the procedure explicitly and visually serves as a demonstrative tool. The reader can assume that \( H \) is the first part of a larger HRN which continues on to the right.

Denote by \( H^1 \), \( H^2 \), and \( H^3 \) the three subprograms in \( H \). We will refer to subsequent subprograms (not shown) to the extending to the right of the picture by \( H^i \) for \( i \geq 4 \). Let \( \lambda^n \) be their respective vector space representations and fix a data flow \( D^n \) in each \( \lambda^n \). In this case, one can choose the vector spaces \( \lambda^n_{p,q} \) in each \( D^n \) to represent the magnitude of the network packets in its position, i.e \( \dim(\lambda^n_{p,q}) \) is a natural number indicating the quantity of formatted data units contained within its vertex. We assign desired data outputs \( \delta^n \) to each \( \lambda^n \).

Denote the final data output of each \( \lambda^n \) (as determined by \( D^n \)) by \( \theta^n \). Let \( F \) denote the set of indexes of the faulty subprograms, i.e \( \delta^i > \theta^i \) for all \( i \in F \), ordered by increasing magnitude. Let \( A \) denote the set of indexes of all able subprograms, i.e \( \delta^i < \theta^i \), ordered in the same way. We represent each \( \lambda^n \) by a point \( n \in \mathbb{R} \), where \( \mathbb{R} \) is endowed with the standard topology. In the manner of section 4.1, define precosheaves \( G_1, G_2 : \text{Open}(\mathbb{R}) \to \text{Vect}_k \). We recall the definition on open sets (see section 4.1 for notation and more explanation): If \( \lambda^n \) is faulty, then \( G_1(U) = \overline{\lambda^n_U} \). If \( \lambda^n \) is able, then \( G_1(U) = 0 \). \( G_1 \) records the margins of error in the maximal point of \( U \). If \( \lambda^n \) is able, then \( G_1(U) = \hat{\lambda^n_U} \). If \( \lambda^n \) is faulty, then \( G_1(U) = 0 \). By Theorem 4.3, we can detect the margin of deficit (resp. surplus) contributed by a faulty (resp. able) subprogram \( \lambda^i \) algebraically based on the dimension of the vector space \( H_k(\text{Cone}(\varphi^n_k)) \simeq \ker(\varphi^n_k) \). Thus we can input the ordered sets \( F \) and \( A \) as \( S^1 \) and \( S^2 \) in Algorithm 1 “Generation of \( P \)-intervals” to obtain a set of points \( \{(i,j) \in \mathbb{Z}_\infty \times \mathbb{Z}_\infty \} \). A point \( (i,j) \) parametrizes the lifetime of a single network packet malfunction, i.e a error is created at subprogram \( i \) and is detected and fully corrected by subprogram \( j \). Outputted points \( (k, \infty) \) are points at infinity and therefore are errors in \( H \) that are never detected and corrected. The multiplicity of any point \( (i,j) \) is precisely the dimension of the data flow error created at \( i \) and resolved at \( j \).
The outputted persistence diagram whose points parametrize the lifetime and magnitude of data flow malfunctions across the packet delivery system allows the application of statistical tools to assess the clustering of points, which in this specific case reveals information regarding the clustering of packet delivery malfunctions across various time slots. This in turn provides novel information to enable systematic assessments of flaws in the network packet delivery system and pathways to fixing them.

Acknowledgments

The authors would like to thank Dr. Chad Giusti and Dr. Robert Ghrist for helpful discussions and ideas, as well as an anonymous referee who greatly helped improve the readability of the paper.

References

[1] D. Attali, U. Bauer, O. Devillers, M. Glisse, and A. Lieutier, Homological reconstruction and simplification in $\mathbb{R}^3$, Comput. Geom. 48 (2015), no. 8, 606–621.
[2] G. E. Bredon, Cosheaves and homology, Pacific J. Math. 25 (1968), 1–32.
[3] ______, Sheaf theory, second ed., Graduate Texts in Mathematics, vol. 170, Springer-Verlag, New York, 1997.
[4] P. Bubenik, Statistical topological data analysis using persistence landscapes, J. Mach. Learn. Res. 16 (2015), 77–102.
[5] J. Curry, Sheaves, cosheaves and applications, arXiv:1303.3255 [math.AT] (2013).
[6] T. K. Dey, F. Fan, and Y. Wang, Graph induced complex on point data, Comput. Geom. 48 (2015), no. 8, 575–588.
[7] R. Ghrist and Y. Hiraoka, Applications of sheaf cohomology and exact sequences to network coding, Proc. NOLTA (2011).
[8] S. Y. Oudot, Persistence theory: from quiver representations to data analysis, Mathematical Surveys and Monographs, vol. 209, American Mathematical Society, Providence, RI, 2015.
[9] B. Schrauwen and L. Buesing, A Hierarchy of Recurrent Networks for Speech Recognition, NIPS, Proceedings (2009).
[10] D. B. West, Introduction to graph theory, Prentice Hall, Inc., Upper Saddle River, NJ, 1996.
[11] Z. Zuo, G. Liu X. Shuai, B. Wang, and B. Wang, X. Wang, Learning Contextual Dependencies with Convolutional Hierarchical Recurrent Neural Networks, arXiv:1509.03877 [cs.CV] (2016).