2006-06-29

A Dynamic Attribute-Based Load Shedding and Data Recovery Scheme for Data Stream Management Systems

Amit Ahuja
Brigham Young University - Provo

A DYNAMIC ATTRIBUTE-BASED LOAD SHEDDING AND DATA RECOVERY SCHEME FOR DATA STREAM MANAGEMENT SYSTEMS

by

Amit Ahuja

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

July 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Amit Ahuja

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Yiu-Kai Dennis Ng, Chair
Date	Phillip Windley
Date	Eric Ringger

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Amit Ahuja in its final form and have found that (1) its format, citations and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Yiu-Kai Dennis Ng Chair, Graduate Committee

Accepted for the Department

Bryan S. Morse
Associate Chair

Accepted for the College

Thomas W. Sederberg
Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

A DYNAMIC ATTRIBUTE-BASED LOAD SHEDDING AND DATA RECOVERY SCHEME FOR DATA STREAM MANAGEMENT SYSTEMS

Amit Ahuja
Department of Computer Science
Master of Science

Data streams being transmitted over a network channel with capacity less than the data rate of the data streams is very common when using network channels such as dial-up, low bandwidth wireless links. Not only does this lower capacity creates delays but also causes sequential network problems such as packet losses, network congestion, errors in data packets giving rise to other problems and creating a cycle of problems hard to break out from. In this thesis, we present a new approach for shedding the less informative attribute data from a data stream with a fixed schema to maintain a data rate lesser than the network channels capacity. A scheme for shedding attributes, instead of tuples, becomes imperative in stream data where the data for one of the attributes remains relatively constant or changes less frequently compared to the data for the other attributes. In such a data stream management system, shedding

a complete tuple would lead to shedding of some informative-attribute data along with the less informative-attribute data in the tuple, whereas shedding of the less informative-attribute data would cause only the less informative data to be dropped.

In this thesis, we deal with two major problems in load shedding: the intra-stream load shedding and the inter-stream load shedding problems. The intra-stream load shedding problem deals with shedding of the less informative attributes when a single data stream with the data rate greater than the channel capacity has to be transmitted to the destination over the channel. The inter-stream load shedding problem refers to shedding of attributes among different streams when more than one stream has to be transferred to the destination over a channel with the channel capacity less than the combined data rate of all the streams to be transmitted. As a solution to the inter-stream or intra-stream load shedding problem, we apply our load shedding schema approach to determine a ranking amongst the attributes on a singe data stream or multiple data streams with the least informative attribute(s) being ranked the highest. The amount of data to be shed to maintain the data rate below the capacity is calculated dynamically, which means that the amount of data to be shed changes with any change in the channel capacity or any change in the data rate. Using these two pieces of information, a load shedding schema describing the attributes to be shed is generated. The load shedding schema is generated dynamically, which means that the load shedding schema is updated with any change in (i) the rankings of attributes that capture the rate of change on the values of each attribute, (ii) channel capacity, and (iii) data rate even after load shedding has been invoked. The load shedding schema is updated using our load shedding schema re-evaluation algorithm, which adapts to the data stream characteristics and follows the attribute data variation curve of the data stream. Since data dropped at the source may be of interest to the user at the destination, we also propose a recovery module which can be invoked to recover attribute data already shed. The recovery module maintains the minimal amount of information about data already shed for recovery purpose. Preliminary experimental results have shown that recovery accuracy ranges from *90%* to *99%*, which requires only *5%* to *33%* and *4.88%* to *50%* of the dropped data to be stored for weather reports and stock exchanges, respectively. Storing of recovery information imposes storage and processing burden on the source site, and our recovery method aims at satisfactory recovery accuracy while imposing minimal burden on the source site.

Our load shedding approach, which achieves a high performance in reducing the data stream load, (i) handles wide range of data streams in different application domains (such as weather, stocks, and network performance, etc.), (ii) is dynamic in nature, which means that the load shedding scheme adjusts the amount of data to be shed and which attribute data to be shed according to the current load and network capacity, and (iii) provides a data recovery mechanism that is capable to recover any shedded attribute data with recovery accuracy up to 90% with very low burden on the source site and 99% with a higher burden on some stream data. To the best of our knowledge, the dynamic load shedding scheme we propose is the first one in the literature to shed attributes, instead of tuples, along with providing a recovery mechanism in a data stream management system. Our load shedding approach is unique since it is not a static load shedding schema, which is less appealing in an ever-changing (sensor) network environment, and is not based on queries, but works on the general characteristics of the data stream under consideration instead.

ACKNOWLEDGEMENTS

I acknowledge and thank the people who gave me a lot of help along the way while I was working on this thesis. More than anything, I thank my advisor Dr. Dennis Ng for his amazing guidance, support and insights that he gave me throughout all phases of the thesis. This thesis would not have come about without his tremendous efforts.

I also greatly thank Dr. Phillip Windley for his invaluable suggestions and review of this thesis which made my progress accelerated.

Finally, I cannot exclude my gratitude for my family: my father, Ramesh Ahuja, for always being the source of inspiration, and mother, Sunaina Ahuja, for always suppoting me. Without their support, patience and sacrifice during many days and nights, even their weekends and holidays, nothing would get done.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The recent years have seen tremendous change in the way data are transferred over the Internet on which data streams are defined. A data stream is a large volume of data transmitting over a network from a source site to a destination site continuously. Since most of the applications using stream data work in real-time (i.e., data is processed as it arrives at the destination site, and most of the times current stream data are not re-processed again at a later time), it is unnecessary to store the entire data in some form of memory, e.g., seismic changes monitoring applications. Along with being unnecessary, it is also impractical to store a data stream on the disk at the destination site, since storing the entire data stream would require infinite amount of space from a practical as well as theoretical point of view. Since a data stream is defined [HIT] as a "sequence of digitally encoded signals used to represent information in transmission," a data stream represents a sequence of input data that comes at a very high rate. As the high-rate stresses are introduced in the communication and computing infrastructure [Mut03], so it may be hard to (i) transmit the entire input to an application program, (ii) compute sophisticated functions on large pieces of inputs at the rate it is presented, and (iii) store temporarily or archive the entire data stream.

In many real-world applications today, such as network monitoring, telecommunications data management, web personalization, manufacturing, sensor networks, data takes the form of continuous data streams rather than finite stored data sets, and clients require long-running continuous queries as opposed to one-time queries. For such applications data streams tend to be more appropriate as opposed to simple databases, since these applications work on continuously flowing infinite data. Traditional database management systems (DBMS) are ill-equipped to handle numerous continuous queries over data streams [MWA$^+$03]. The authors of Aurora [ACE$^+$03] claim that the static model of databases, with dynamically changing queries being executed over static data, is not designed for handling stream data, which has static queries being executed over dynamically changing data. In addition, they suggest that handling data streams in a DBMS would require the DBMS to serve real-time applications, making it imperative that the DBMS employ intelligent resource management (e.g., scheduling) and graceful degradation strategies (e.g., load shedding) during periods of high load, which are not features of a DBMS, as it is designed as a store-and-query model instead. Also, limited computing power of the destination site or low channel capacity of the network over which data streams are to be transmitted lead to long delays on data, which may be acceptable with the static model of DBMS but inappropriate with data streams applications.

In order to handle the problems imposed by high-speed stream data in data processing, load shedding has been proposed as a solution. In dealing with the high data stream input rates, different load shedding approaches [ABC$^+$05, BDM04, CWY05, Gol04] have recently been introduced. With load shedding, data from data streams are dropped and a reduced data stream is supplied to any application processing on it. Most of the load shedding methods [MWA$^+$03, RH05] employ either the naive approach, static load shedding approach, and dynamic load shedding using delay-based QoS graphs or value-based approach [Mut03]. Upon detecting an overload, a data stream management system (DSMS) launches an effort to reduce the volume of the incoming stream data via load shedding. The naive approach sheds tuples at random points in the network without having any control over which tuples to be dropped. This works in a manner similar to dropping overflow packets in packet-switching networks [ACE$^+$03]. Although being simple, the naive approach has two potential problems: (i) overall system utility might be degraded more than necessary, and (ii) application semantics might be arbitrarily affected. Approaches using the static load shedding scheme employ availability of a priori knowledge of the tuple delays or frequency distribution of values to determine what tuples to be shed. Static load shedding, although overcomes the problems of the naive approach, still faces at least one significant drawback. The data tuples considered to be less important on the basis of some priori knowledge are shed; however, these tuples may not remain less important after a while, since the priori knowledge may not always be valid over a period of time. For example, priori knowledge may project the temperature attribute in the weather information to be less important, but at a later time the temperature attribute in the weather information may become important. Load shedding triggered as a result of dynamic information, instead of priori knowledge about a data stream, would be able to overcome the drawback of using a static load shedding model. The load shedding method proposed in this thesis is dynamic and adaptive, which means that our load shedding approach applies its load shedding scheme re-evaluation mechanism in a regular periodic manner to the pattern in which the data stream attribute values vary, and re-configurate the load shedding scheme accordingly. Also, besides load shedding approach, we present a data recovery method in retrieving shed data, since data shed at the source site may be of interest to the user at the destination site, which has not been discussed in the literature. Our recovery method provides high data recovery accuracy, low information loss, and with a low overhead of storing recovery information.

We note that all data streams have a static schema, which means that the involved attributes and their corresponding data types do not change, but have a high dynamic data rate. Some of the data values of an attribute vary more often than the values of other attributes, whereas others may remain nearly constant. Many applications process stream data in which all tuples are important, with some attribute values being more "informative" than the others. For example, a weather data stream may contain data for temperature, pressure, wind speed, barometer pressure, humidity, precipitation, and visibility. Some of these weather attribute values may be more "informative" than the values of other attributes (for certain geographical locations). For example, deserts have temperature varying tremendously between day and night times, whereas precipitation may be constant over weeks. However, around the coastal areas, temperatures vary slightly over days, whereas the precipitation might change tremendously within the same day. These scenarios demand a new approach towards load shedding for stream data by dropping less-informative attribute (values) in tuples, instead of the entire tuples, since a complete tuple may contain informative attributes with varying attributes values as well as less-informative attributes. As a result, shedding less-informative attribute values in data stream tuples, instead of entire tuple in a data stream, does not shed informative-attribute values. In this thesis, we propose a dynamic method which would cater to this need. The shedding of attribute data equivalent in data size to the amount of tuple data which was needed to be shed creates the same effect in terms of amount of data shed as when shedding complete tuples. We call our load shedding approach an attribute-based load shedding approach for data streams.

Differed from existing load shedding approaches, our load shedding approach preprocesses all data values in a data stream using moving averages, which serves the purpose of diminishing any rare irregularities in the data values. Consider, for instance, in a weather information data stream, in which it shows one day of heavy rains in the middle of 100 days of drought. The results of the preprocessing step are compared with the data before preprocessing using line graphs to show that the irregularities were diminished and keeping the data after preprocessing close to the actual data before preprocessing. The data after preprocessing is used to compute the load shedding scheme of a data stream, which comprises of the designated attributes to be dropped and their data. The number of attributes to be dropped, n, is computed by using (i) the information on the channel capacity and (ii) the data rate of the stream. According to the value of n, we compute the standard deviation on each segment, called the sliding window, of the stream data to be shed, which yields a ranking of the n lowest informative attributes; with the least-informative attributes being ranked highest, which generated the load shedding scheme of the segment. The load shedding scheme that controls the data to be shed is computed and updated dynamically, which means it adapts its re-evaluation pattern according to the pattern in which the ranking changes.

Besides the load shedding scheme of a data stream, we also propose a data recovery method on the shedded data by maintaining information about the data shed at the source site. This information is used at a later time when data needs to be recovered, which occurs when the destination site needs some attribute data which had been shed from the stream data sent to the destination site. In such a situation, the source site uses the recovery information it had stored and sends the best approximation of the shed to data to the destination site. To measure the accuracy of our data recovery approach and to show the high accuracy of our data recovery method, we conduct various experiments on different data streams, and we compare the actual data with the recovered data. We plots graphs between the actual values and the recovered values for the comparison purpose.

We proceed to present our results as follows. In Chapter 2, we discuss related works in load shedding in data streams. In Chapter 3, we (i) introduce our two load shedding approaches: the intra-stream and inter-stream load shedding approaches, (ii) present the architectures for both the intra-stream and inter-stream load shedding approaches, and (iii) include some initial experimental results of our load shedding approach to verify the correctness and effectiveness of our load shedding approach along with the proposed data recovery method. In Chapter 4, we include the experimental results on our load shedding and data recovery approach to measure the quality of the overall design. In Chapter 5, we give a concluding remark.

Chapter 2

Related Work

In this chapter, we discuss related work in load shedding in data streams (in Section 2.1), as well as video/audio lossy compression (in Section 2.2), that appear in the literature.

2.1 Existing work in data stream load shedding

Many efforts have been made in the past to handle load shedding mechanisms in data stream management systems (DSMS). Well known load shedding techniques are presented in (i) Borealis [ABC$^+$05], which accomplishes load shedding by shedding tuples using temporarily adding drop operators, (ii) Data Triage [RH05], which deals with the arrival of bursty data in data streams, (iii) Loadstar [CWY05], which uses a QoD-based load shedding scheme for load shedding, (iv) [Gol04], which introduces a technique for load shedding by applying data optimizations on sliding windows, (v) [BDM04], which discusses load shedders used to shed tuples at various points in a query plan, and the load shedders can be removed and applied as and when needed, and (vi) Aurora [ACE$^+$03], which invokes load shedding by using QoS-based quality metrics.

Borealis [ABC$^+$05], a data stream management system, accomplishes load shedding by temporarily adding "drop" operators to the Borealis processing network as a way to shed tuples. Shedding tuples is either based on the values of the tuples or in a randomized fashion, which rectifies the overload situation and provides better overall end-to-end latency at the expense of reduced answer quality. Borealis also presents a distributed architecture with different load shedding nodes in the query plan indulging in the dropping tuples, and dropping tuples at a node reduce the load on the downstream nodes. Borealis uses loss-tolerance QoS as a quality metric, which is a static QOS-based approach, to decide which tuples in a data stream to drop, whereas our load shedding approach, as proposed in this thesis, uses a dynamic approach to decide what attributes to in a data stream to drop.

Data Triage [RH05] treats load shedding as a problem to deal with bursty data arrival that has not been adequately addressed in previous work. In between bursts, Data Triage generates completely accurate data recovery results. When a crisis situation causes a burst of unusual data, Data Triage sheds load to maintain low result latency but keeps enough data to produce a relatively accurate picture of what happened during the burst. Data Triage has several other important benefits for handling bursty loads, which include (i) responding quickly to changes in load and (ii) providing relatively accurate query results across a wide variety of data rates and available amounts of bandwidth. Data Triage is very close to our approach of handling load shedding on data streams as it handles the bursty data in an efficient manner, but misses any design for data recovery. Our load shedding approach overcomes this limitation with a data recovery design, which imposes only minimal cost in terms of storage.

The designers of Loadstar [CWY05] consider a QoD-based load shedding scheme. They focus on two load shedding problems: (i) the classification of data that are dropped by the load shedding scheme and (ii) the decision on when to drop data from a stream. Loadstar classifies the data streams into different application domains and assigns them CPU resources accordingly. Loadstar employs a QoD measure for predicting the classification in the next time unit and is able to learn and adapt to changing data characteristics in stream data. The approach of using QoD, instead of QoS, by Loadstar is more adaptive than QoS in Borealis. Even though Loadstar can decide when to apply load shedding to a data stream, it lacks the ability to control the communication rates of the data streams. (For example, given many video streams, the frame rate of each stream is proportional to its importance.) Our load shedding approach, when dealing with multiple streams, sheds a greater number of attributes from data streams with less-informative attributes than the number of attributes shed from the data streams with more-informative attributes. Thus, our load shedding approach overcomes the limitation of Loadstar and is more effective in controlling the communication rates of stream data.

In another approach [Gol04] towards load shedding on data streams, the researcher views sliding windows as approximations to infinite data streams and proposes the application of optimization techniques, like functional dependencies, to these sliding windows to reduce the amount of data in these sliding windows. The researcher views the reduction in data in these sliding windows as a form of load shedding and believes that reducing the amount of data in the sliding windows by employing functional dependency between various tuples is a way of load shedding by prematurely evicting tuples from their windows, which is beneficial since premature eviction saves any processing time which would have been spent on them before they would have been evicted later. [Gol04], however, does not address the problem on dynamically scaling the amount of optimization applied to reduce the window data to be shed. Our load shedding approach determines the amount of data to be shed dynamically with an adaptive load shedding scheme, i.e., the data to be shed and the amount of data to be shed change with any modifications in the ranking amongst the attributes in a data stream based how informative they are, along with any change in the data rate of a data stream.

In [BDM04], the authors handle load shedding by introducing load shedder at various points in a query plan. Every incoming tuple, which serves as an input to a load shedder, is passed onto the next load shedder with a probability p, called the sampling rate. The authors propose the usage of aggregate values to compensate for data lost by load shedding. The decisions about where to introduce load shedders and how to set the sampling rate for each load shedder, as presented in [BDM04], are based on statistics about the data streams, which include observed stream arrival rates and operator selectivities. The major drawback of this approach is its approach towards data recovery, since it uses aggregate values to compensate for data lost, whereas our approach uses a synopsis-based intelligent approach and handles data recovery much more efficiently.

In Aurora [ACE+03, ZCC+02], another data stream management system, an overload on a system is detected as a result of static or dynamic analysis on the resources at the system. Aurora attempts to reduce the volume of tuple processing in a data stream via load shedding. Aurora relies on QoS information to guide the load shedding process by using a QoS monitor that watches over system performance and activates the load shedder whenever an overload is detected. If the performance is below a specified level, Aurora sheds load till the performance reaches a certain level. Aurora also uses a router which decides if incoming stream data should be processed or stored in persistent storage. Aurora, however, uses the QoS information, which is predefined, and thus lacks the dynamic nature. Our load shedding approach is dynamic in nature since it uses an adaptive load shedding technique.

Other load shedding approaches adopted by numerous data stream management systems have also been proposed. STREAM [MWA$^+$03], the data stream management system developed at Stanford, uses query plans for handling data streams, whereas CQL [ABW03] is capable of handling relation-to-relation, stream-to-relation, and relation-to-stream operators. One of the relation-to-stream operators in CQL, the stream-sample operator, is used only for system-managed load shedding, which drops a specified fraction of stream tuples from its input queue based on a uniform random sample. The designers of STREAM also identify the two primary consumers of memory in their data stream management system as synopses and queues. They use approximation techniques to reduce the synopsis and queue sizes, and further suggest that if the queues grow too large, then simply dropping packets could also help. Furthermore, STREAM includes an extended SQL language which supports the processing of both data streams and conventional relations.

Our load shedding approach is different from existing load shedding methods. None of the existing load shedding approaches considers a dynamic load shedding system for data streams with focus on dropping attributes in a tuple, but the tuple itself. Under certain conditions, where the values for some attributes may be of less interest to the user while other attributes are significantly important, the dropping of attributes is more feasible than dropping tuples. We discuss in detail in subsequent chapters our load shedding approach in this thesis, along with the method in determining and updating a load shedding scheme. Hereafter, we will present our data recovery approach on shed data.

2.2 Audio/Video lossy compression

The state of the current research [TRL01, MQ02, AA03, MMdEdT03, CLL$^+$04] also shows that a lot of works have been done in lossy compression of streaming audio/video data, which can be viewed as consecutive frames of continuous data. Often portions of audio/video data in two consecutive frames are replicated and thus redundant when they are transmitted with other data in their frames through the network. For example, a streaming video with men performing on a stage with a constant black screen in the background would have the video information regarding the positions of the men on the stage changing between consecutive frames; however, the video information about the black screen remain constant, which are considered redundant across multiple frames. Approaches in lossy compression of streaming audio/video data remove redundant information between multiple frames, which can be viewed as less informative data in this thesis. We are well aware that streaming audio/video data are processed as continuous frames of information, and each frame contains pixels, each of which is represented in the binary form as 1 or 0. Pixels in a frame for which the information do not change over consecutive frames would have the same binary representation, which are treated as redundant. We can adopt our load shedding approach for shedding these redundant information by comparing the binary representation of the pixels at the same positions in consecutive frames, and only the portions of audio/video data that vary from one frame to another with a change greater than a predefined redundant threshold would be retained, whereas other portions with a change less than the redundant threshold would be shed. We proceed to discuss works already done in the field of lossy compression of audio/video data and how to apply our load shedding approach on them below.

[MQ02] propose a method capable of delivering streaming video sequences with nearly constant perceptual quality. The proposed method classifies video data packets as premium or regular, depending on the network conditions (such as network bandwidth and traffic) and the pre-defined desired level of quality of service. Premium packets are transmitted lossless, with a low delay, whereas regular packets are delivered as "best-effort packets" with losses. The proposed method demonstrates higher effectiveness experimentally when compared to the method that "punishes" each video data packet equally by transmitting them as best-effort packets with losses, i.e., without classifying them into premium and regular packets. Our load shedding approach can be modified to take an entirely different approach towards shedding information from video packets as addressed in [MQ02]. Our load shedding approach can convert audio/video data to stream data and perform lossy compression on them by shedding. This conversion can be accomplished by considering the 0s and 1s bits in the frames of audio/video data as textual stream data, the type of data handled by our load shedding and data recovery approach. Bits that remain constant at the same positions between two consecutive frames could be redundant and shed. We call such a conversion and shedding approach as the audio/video load shedding compression approach (AVSCOM, for short). Since the proposed streaming video method in [MQ02] is a pre-defined quality of service-based approach, it lacks dynamic nature in terms of classifying the packets as premium or regular packets. AVSCOM, on the other hand, determines the video data which should be shed dynamically by (i) analyzing each video frame (which plays the role of a sliding window) and (ii) detecting the amount of informational change in the video frame from the previous video frame to decide whether portions of the video frame should be retained or shed.

[MMM+03] propose a change-detection and high-interest, coded-region based approach for compressing streaming video data in wireless sensor networks. The proposed algorithm considers consecutive frames of video data and analyzes each frame to determine the regions of frequent changes in the frame, which are considered as the regions of high interest. After determining the regions of high interest, the algorithm captures and encodes only the changes between the regions of high interest on consecutive frames before transmitting the video information from the source to destination. Our AVSCOM can detect the amount of informational change in different regions of a frame, i.e., detecting the changes occurred in different groups of bits on two consecutive frames considered by [MMM+03], with groups of bits containing higher informational change to be treated as regions of higher interest, which are treated as more informative than others.

[CLL$^+$04] propose a video encryption technique for face-to-face video conferencing. The proposed algorithm is based on the assumption that positions of different parts of a human face are similar between two frames if the frames show similar orientation of the face. [CLL$^+$04] divide each frame into parts so as to disintegrate the face in the frame into eyes, nose, ears, and mouth. The information about the eyes, nose, ears, and mouth in a frame like the color of the eyes, the shape of the nose, lips, ears, etc., are not transmitted over the network; instead, the information about the orientation of the face is. This is done to reduce the amount of information to be transmitted by not transmitting any detailed information. A set of frames containing a complete face in different orientations, called a set of reference frames at the destination site, which are transmitted earlier from the source site, is used for recovering the information that was not transmitted from the source site. The orientation of each transmitted face is compared with the reference frames at the destination site, and the position of the eyes, nose, ears, and mouth from the reference frames is utilized for regenerating the complete picture. Our AVSCOM handles the regeneration of video frames in lossy compression without using any reference frames; instead, it uses recovery information which has been stored before shedding redundant video information from a frame at the source site. Also, we have observed a limitation of the proposed method in [CLL$^+$04], i.e., adequate training is required to obtain the reference frames to be able to handle data recovery. Our AVSCOM, when applied to lossy compression of streaming audio/video data, does not require any training to be able to handle data recovery and thus would not face the limitation of the adequate training requirement problem in [CLL$^+$04], since the data recovery method in our AVSCOM uses recovery information captured in real-time for data recovery instead of the reference data generated in [CLL$^+$04] by the training approach.

[TRL01] propose a content-sensitive video streaming approach for streaming video over a very low bit-rate lossy wireless network. The authors claim that their algorithm reduces the video frame rate while preserving the quality of displayed frame. The algorithm performs content analysis to extract and rank all video frames according to the amount of informational change in the frames from the preceding frames. Frames with more informational change from their preceding frames are ranked higher and have a higher priority of being sent by the server. The authors also present what is called an efficient, adaptive, and robust streaming protocol (SSP) for transferring data over varying bandwidth and high lossy networks. Our AVSCOM, however, does not rank frames; instead, it sheds redundant information between consecutive frames and retains recovery information needed for data recovery. Our AVSCOM, when applied to the lossy compression problem for audio/video data, can analyze the bits in each frame and decide which data bits are to be shed and which data bits are to be retained based on the amount of informational change among all the bits in the current frame from their corresponding bits in the previous frame.

SH-AMBTC, which stands for semi-hexagonal absolute moment block truncation coding [AA03], adopts a compression method for video conferencing by viewing a stream of Mpeg video as sequential frames of bitmap images. The authors propose a predictive scheme which constructs the middle frames in a stream of video data based on the information from end frames in a stream of video data instead of transmitting the middle frames, since there can be a large amount of redundant information between the end frames and the middle frames. Adapting a new predictive technique, the authors try to obtain the bitmap of the middle frames in a sequence of frames, from the bit-map of the end frames only (i.e., the first and the last frames in the group), by using an interpolation formula to obtain the bit-map of the middle frames. Our AVSCOM, however, does not employ interpolation because the textual data stream from sensor networks that we consider in this thesis may not contain any continuous pattern. For example, temperature in one tuple could be 40°F, followed by 60°F in the next tuple, which is then followed by 47°F in the next tuple. Patterns, however, exist for audio/video data. For example, the running motion of a man would always follow a continuous pattern by raising his left leg and putting it down, followed by raising his right leg and putting it down. Our AVSCOM does not deal with middle and end frames, instead determines the redundant, i.e., replicated information, between consecutive frames and sheds the bitmap representation of the replicated information, which is treated as redundant information. Before shedding the redundant information, our AVSCOM retains recovery information regarding the redundant information, and at a later point of time uses this recovery information to recover any shed data. Thus, our AVSCOM, unlike [AKA03], does not depend on detecting or regenerating the middle frames using the end frames by interpolation.

In the above discussion of the potential application of our AVSCOM to the lossy compression problem in audio/video data, we have seen that our load shedding approach could be adopted in solving the lossy compression problem in transmitting audio/video data through the network. Leaving it as a topic of further research and according to the current state of our load shedding approach, we do not make any claims regarding the performance of AVSCOM when applied to lossy compression on audio/video data. Though, we feel that with subsequent work and design, our load shedding approach could be enhanced to cover the lossy compression problem in transmitting audio/video data, in addition to identifying the less informative attributes in a data stream, shedding them, and providing a recovery method.

Chapter 3

Our Load Shedding Approaches

In this chapter, we propose two different strategies for load shedding: the *intra-stream load shedding* and the *inter-stream load shedding.* The intra-stream load shedding is defined as shedding data of less-informative attributes within a particular data stream to lower the data transmission rate at the source site to meet the limited capacity of the data transmission channel, i.e., the channel capacity is lower than the data transmission rate of the data stream. The uniqueness of our intra-stream load shedding approach includes (i) minimizing information loss by shedding less-informative attributes instead of tuples, since tuples may contain less-informative as well as (more-)informative attributes, and (ii) the ability to recover data shed at the source site, requiring only minimal recovery data from data streams to be stored at the source site.

The inter-stream load shedding, on the other hand, deals with shedding on various data streams when multiple streams have to be transmitted over a single channel with the channel capacity less than the cumulative data rate of the data streams. The inter-stream load shedding may have been preceded by intra-stream load shedding individually on each of the involved data streams. Our inter-stream load shedding approach sheds a number (i.e., zero or more) of less-informative attributes and their data from each involved data stream to bring the cumulative data transmission rate below the channel capacity. Our inter-stream load shedding is unique, since the inter-stream load shedder co-ordinates with the central load shedder that obtains information regarding the data transmission rates of each (shed) data stream being transmitted over the channel to determine the number of attributes to be further shed from each data stream.

The two different load shedding processes, i.e., intra-stream and inter-stream load shedding, can be consolidated into one if the central load shedder can co-ordinate directly with each data stream source site to perform intra-stream load shedding without conducting further inter-stream load shedding. In order to determine the amount of data to be shed for intra-stream load shedding, the central load shedder would have to gather from each data stream (i) the data transmission rate and (ii) the capacity of the individual channel (at the source site) on which the data stream is to be transmitted. Although the central load shedder can sense the data transmission rate for each data stream, it cannot sense the channel capacity at the source site of the individual channels, and thus unable to determine the amount of data to be shed at the source site from each data stream for intra-stream load shedding. As a result, the central load shedder has to (i) rely on each data stream to perform intra-stream load shedding independently, which requires the existence of both intra-stream load shedding and inter-stream load shedding as distinct entities.

We use sliding windows, each of which can be viewed as an approximation of an infinite data stream [Gol04], to capture the data stream to be processed. The purpose of capturing data momentarily is to use them for any data processing. In load shedding, the data in a sliding window are analyzed to determine which portion of the data are to be shed.

The rest of this chapter is organized as follows. In Section 3.1, we introduce sliding windows. In Section 3.2, we propose our intra-stream load shedding approach. In Section 3.2.2 we define estimated *weighted means* and explain how they are used as a preprocessing step of the intra-stream load shedding process for smoothening the data in the current sliding window. In Section 3.2.3, we present our (intra-stream) load shedding scheme generation and maintenance approach. In Section 3.2.4, we discuss the different strategies of recovering data as building blocks of our load shedding and data recovering system. In Section 3.2.5, we include the architecture for our intra-stream load shedding sub-system. In Section 3.3, we introduce our inter-stream load shedding sub-system.

3.1 Sliding Windows

Data stream is a stream of tuples or rows [HIT] continuously flowing over the network from the source site to a destination site. The properties a data stream exhibited are very different from those exhibited by a database. Unlike databases, data streams are unbounded and flowing continuously, and storing a data stream in a static location is impractical [ACE$^+$03] , since the amount of data flowing in a data stream is in the order of hundreds of megabytes every hour, and storing such huge amount of data is not feasible. Furthermore, queries executed on data streams are called *continuous queries*. Unlike queries on databases that are executed on data already stored on disks, continuous queries are executed on stream data online in real time. In reality, databases generally exhibit *dynamic queries* on *static data*, whereas data streams exhibit *static queries* on *dynamic data*. Since data streams cannot be treated like static databases, a technology is required to process static queries on unbounded dynamic data in real time. The most effective way of executing continuous queries on a data stream is by using sliding windows, since in real-world applications data streams are processed on excerpts of the data streams rather than the whole streams. A sliding window slides over the incoming data stream, is continuously updated, and considers the most recent tuples in the window. With the use of sliding windows, continuous queries can be executed on a never-ending data stream in real time.

Two different types of sliding windows for stream data processing have been proposed in the literature [MWA$^+$03]: *time-based* windows and *tuple-based* windows. Time-based windows define the window size as a time-frame (e.g., death statistics over the last 10 minutes), whereas tuple-based windows define the window size as number of tuples (e.g., the last 100 phone calls during the last hour). The number of tuples captured in a time-based window increases as the rate of flow of tuples increases, whereas a tuple-based window has a fixed number of tuples and thus the window size is not affected by the rate of flow of tuples at all.

3.2 The Intra-Stream Load Shedding Approach

According to each incoming sliding window of a data stream, our intra-stream load shedding approach first identifies the less-informative attributes, i.e., attributes whose data *vary lesser* when compared to the data of other attributes, in the data stream. We shed less-informative attributes, instead of entire tuples in a data stream, because when we shed less-informative attributes, all the data we shed is the less-informative data, whereas when we shed tuples, we shed some informative data, in the form of informative attributes, along with the less-informative attributes as a tuple may contain data of both informative and less-informative attributes. The major functions of our intra-stream load shedding strategy, as well as our inter-stream load shedding approach, include (i) creating the load shedding scheme of a data stream S[1], which enlists data to be shed from S by the load shedder, and (ii) recovering shed data of S, if needed.

The load shedding scheme generation step is preceded by a preprocessing step that smoothens out any "irregularities" in the source data. An example of such an *irregularity* can be imagined as the weather stream data over the last hour for a county with the precipitation remaining nearly constant except the 13^{th} minute when it rains heavily. Our preprocessing step would smoothen any such irregularities by replacing the original data in a sliding window with the *exponential weighted mean average* (EWMA) of the data values under consideration. (See Section 3.2.2 for details.) After the preprocessing step, the load shedding scheme can be generated according to the required amount of data and their corresponding attributes to be shed, which are dictated by the *channel capacity* of the corresponding data stream. To determine which attributes to be shed, we use *standard deviation* to compute the ranking amongst the attributes of the data stream over the current sliding window with the least informative attribute to be assigned the highest ranking value. Hereafter, the data from the source site is shed according to the load shedding scheme, which is re-evaluated in real-time and enjoys a dynamic nature. Since a complete data stream S can not be stored at the source site, our load shedding scheme generation algorithm uses an excerpt of S, i.e., the current sliding window of S to generate the load shed-

[1] The load shedding scheme, which is *dynamic* in nature, meaning that the load shedding scheme is modified in real-time with changes in (less- or more-) informative attributes of a data stream S, comprises of (a) the attributes to be shed and (b) the corresponding data in S to be shed.

ding scheme of S and subsequent excerpts to update the load shedding scheme of S continuously. In addition, we also propose a recovery algorithm which can be utilized at the destination site to recover any shed attribute data with the recovery accuracy ranging from *90%* to *99%*, while retaining minimal amount of data for recovery in a variety of ranges.

3.2.1 Sliding Window Size

Different segments of a data stream S, which convey up-to-the-moment information, are separated by the *cycle identifier* (*CID*, for short), which is defined as either a single attribute or a combination of attributes, of S. The *CID* of S serves as a key in the recovery matrix of our load shedding approach where recovery information of S are recorded and extracted, and the *CID* values follow a fixed-length repetitive cycle in S, which consist of tuples in S such that the order of appearances of various *CID* values in the tuples fall in the same cycle, and the number of tuples in each fixed-length repetitive cycle in S is called the *cycle length* of S. The cycle length of S is treated as the size of each tuple-based sliding window of S for load shedding and data recovery purpose (see Section 3.2.4 for details). Since the *CID* of S must be transmitted to the destination site and cannot be shed, the *CID* should be minimal, i.e., with the fewest possible attributes that individually come with a cycle length of repetitive values in each fixed-length cycle of S. In this section, we discuss a method in determining the *CID* of S and thus the cycle length of S.

The *CID* of S is detected during the training phase of S, which is carried out before our load shedding system actually starts shedding data and retaining recovery information from S. During the training phase of S, we analyze and evaluate all the data values for each attribute in the training set of S. All the attributes in S

that individually follow a repetitive pattern for their values *form* the set of replicated attributes ($RepA_s$, for short) of S, and whenever a replicated attribute is detected, its cycle length is also recorded. The replicated attributes in $RepA_s$ are partitioned into sets $S_1, S_2, \ldots, S_n (n \geq 1)$ according to the cycle length of each attribute such that each S_i ($1 \leq i \leq$ n) contains all the attributes with the same cycle length. Furthermore, all the attributes in each S_i have a one-to-one relationship with each other, i.e., the value of each of these attribute in a tuple t in S_i of S can uniquely identify the values for all the other attributes in t of S_i, and the sets S_1, $S_2, \ldots,$ S_n are called *one-to-one relationship sets.* Since the replicated attributes in each S_i have a one-to-one relationship with all the other attributes in the same set, only one attribute from each S_i is required, to form the chosen attribute to generate the *CID* of S, and the cycle length of each chosen attribute is used to compute the fixed-cycle length of S. The amount of computation required to identify the *CID* of S is one time and does not impose a lot of burden on the source data stream site.

In finding $RepA_s$ of S, we compare every tuple, starting from the 2^{nd} tuple, in the training data set of S with the 1^{st} tuple in the training data set, till we have found the first repeated value of an attribute, which indicates a potential repetition cycle of the attribute. Assume that there are p ($p \geq 1$) distinct tuples in the training set, and p is sufficiently large, i.e., there are sufficient training data to identify all the replicated attributes in S. Further assume that the 1^{st} match in the comparison to find the repetition cycle of the values of an attribute A is found between the j^{th} ($1 < j \leq p$) tuple and the 1^{st} tuple. This discovery will be followed by the comparison on the values of A between the $j{+}1^{th}$ tuple and the 2^{nd} tuple to determine whether the two tuples have the same value on A. If the values of A are the same, the comparison is followed by yet another comparison between the attribute values of A in the $j{+}2^{nd}$

tuple and the 3rd tuple, and so on till all the tuples in the training set are covered. If any of these comparisons fail, then A is not replicated. For every attribute A in S, a total of p comparisons for A would have to be carried out. Assume that there are q ($q \geq 1$) attributes in S, discovering the $RepA_s$ of S would require $\mathcal{O}(p \times q)$ computations to find all the replicated attributes of S.

We now present an algorithm that finds the *CID* of a data stream S. To discover the *CID* of S, the algorithm first determines each $RepA_s$ of S. Hereafter, replicated attributes in $RepA_s$ are partitioned into a number of one-to-one relationship sets $S_1, S_2, \ldots,\ S_n$, as mentioned before. The replicated attributes in each one-to-one relationship set S_i ($1 \leq i \leq n$) may have different bit lengths, i.e., the numbers of bits occupied by the domain values of different attributes in S_i can be differed. The *CID* of S is formed by selecting the attribute with the <u>least</u> bit length from each one of $S_1, S_2, \ldots,\ S_n$, which guarantees that the chosen attributes to form the *CID* of S have the minimum bit length amongst all the other possible combinations of replicated attributes to form the *CID* of S, which is minimal.

Algorithm 1. *CID Discovery Algorithm*

```
Input: A set of training tuples $T$ of data stream $S$with $r$ attributes and $p$ tuples, with
          $p$ being sufficiently large, i.e., there are sufficient training data to identify all
          the replicated attributes in $S$. The $i^{th}$ tuple in $T$ is denoted as $T_i$
Output: $CID$ and $CycleLength$, the cycle length, of $S$
1. Initialize $CID = \{\}$; $CycleLength\ (\mathrm{A}_i) = 1,\ 1 \leq i \leq r$; $RepA_s = \{\}$.
   /* Detect $RepA_s$,the set of replicated attributes of $S$*/
2. For each attribute $A$ in $T$
   /* Test each attribute in the training set */
     For $i = 2$ to $p$, $j = 1$
```

```
  /* From the $2^{nd}$ tuple onwards, start comparing every
  tuple with the $1^{st}$ tuple for a repeated pattern of $A$*/
    If $T_j[A] = T_i[A]$, then
    /* The value of attribute $A$ in the $i^{th}$ tuple is
    the same as the value for $A$ in the $j^{th}$ tuple */
      (a) While $i \leq p$ /* Verify that all tuples in the training set following
          the $i^{th}$ tuple also follow a pattern of repeated values for attribute $A$ */
            (i) $j = j + 1$
            (ii) $i = i + 1$
            (iii) If $T_j[A] = T_i[A]$, then Pattern = ‘True’
                  Else Pattern = ‘False’ and Break
                  End If
            End While
            /*Attribute $A$ follows a repetitive pattern and is included in $RepA_s$;
            $Cyclen(A)$ is the cycle length for a replicated attribute $A*$ /
          (b) If pattern = ‘True’, then $RepA_s = RepA_s \cup \{A\}$ and $Cyclen(A) =$
              $j - i$
              End If
      End If
  End For
End For
/* Compute the $CID$ of $S$ from $RepA_s$,the set of replicated attributes of $S$*/
Group the attributes in $RepA_s$ according to their cycle lengths by sorting the
(a) /*Partition the sorted attributes in $RepA_s$ into the one-to-one relationship
    sets*/  $S_1 = \{ \}$, $S_1 = S_1 \cup \{ RepA_s[1]\}$, $m = 1$
    For $i$ = 2 to | $RepA_s$|
      If $Cyclen(RepA_s\ [i\ ]) = Cyclen(RepA_s\ [i\ \text{-}\ 1])$
          $S_m = S_m \cup \{\ RepA_s\ [i]\}$
      Else
          $m = m + 1$
      End If
    End For
(c) $CID = \{\text{Min}(S_1) \cup \text{Min}(S_2) \cup \ldots \cup \text{Min}(S_m)\}$, where
    Min($S_i$), $1 \leq i \leq m$, denotes the attribute with the minimum bit length amongst
    all the attributes in $S_i$; if there are more than one attribute in $S_i$ with the
    minimum bit length, then any of these minimum bit length attributes can be
    selected as the attribute to represent $S_i$.
(d) $CycleLength = 1$
(e) For each attribute $A \in CID$
    $CycleLength = CycleLength \times CycleLength\ (A)$
    /* $CycleLength$ is the product of cycle lengths for the attributes in $CID$ of $S$ */
    End For
```

Example 1 Consider the training data in Table 3.1 as an example that demonstrates the discovery of *CID* for a data stream S with attributes A_1, A_2,A_3, and A_4.

In Table 3.1, the attributes A_1, A_2, and A_3 are replicated attributes. Thus, the set of replicated attributes $RepA_s$ is $\{A_1, A_2, A_3\}$, with $Cyclen(A_1) = 3$, $Cyclen(A_2) = 3$, and $Cyclen(A_3) = 4$. Partitioning the attributes in $RepA_s$ into sets, with each set having attributes of the same cycle length, yields sets $S_1 = \{A_1, A_2\}$ and S_2

$\mathbf{A}_1$	$\mathbf{A}_2$	$\mathbf{A}_3$	$\mathbf{A}_4$
a_1	aaaa	a	29.91
a_2	bbbb	b	27.96
a_3	cccc	c	30.09
a_1	aaaa	d	30.09
a_2	bbbb	a	29.91
a_3	cccc	b	29.8
a_1	aaaa	c	29.77
a_2	bbbb	d	30.09
a_3	cccc	a	30.18
a_1	aaaa	b	29.94
a_2	bbbb	c	29.91
a_3	cccc	d	30
a_1	aaaa	a	30
a_2	bbbb	b	30.09
a_3	cccc	c	29.78
:	:	:	:
:	:	:	:

Table 3.1: A sample data stream training data set

= $\{A_3\}$. Assume that the bit lengths for attributes A_1, A_2,and A_3 are 4, 8, and 2 bytes, respectively. Then the *CID* of S is $\{\text{Min}(S_1) \bigcup \text{Min}(S_2)\} = \{\text{Min}(\{A_1, A_2\}) \bigcup \text{Min}(\{A_3\})\} = \{A_1\} \bigcup \{A_3\} = \{A_1, A_3\}$, and the cycle length of S is $\mathit{CycleLength} =$ Cyclen(A_1)$\times$ Cyclen(A_3) $= 3 \times 4 = 12$.

3.2.1.1 Errors in Training Set Data

The accuracy of our *CID* discovery method relies on the correctness of training set data. If the attribute values in a training set data have errors, causing the loss of information about the repetition of values of an attribute, then the *CID* of the corresponding data stream may not be detected correctly. These errors are sometimes referred to as *bit-errors* as the bit(s) in a byte of a data value is (are) changed from a '0' to '1', or vice versa. One widely-used method to detect and correct these errors is the *Hamming Code* [Ham50], an error correcting code, which is a commonly accepted error detection technique in computer networks. A lot of other error detection and correction codes have been proposed in the recent years, which include Reed-Solomon code [RS60], Reed-Muller code [TSZS01], Binary Golay code [Gol49], Convolutional code [Vit67], Turbo code [Gum98], and others. The simplest error correcting codes can correct single-bit errors and detect double-bit errors. Different error correcting codes differ in their computational complexity and error detection/correction overhead, like parity bits. Having considered these error correcting codes, we adopt the Hamming code due to (a) its low computational complexity, and (b) low error detection/correction overhead (parity bits, in this case). Even though the Hamming code can correct only single bit-errors, it detects double bit-errors. The Hamming code is a good choice for detecting and correcting attribute values, partially due to the fact that error correction in training data set is not critical for our load shedding approach, in addition to that we can use more than one training data set for error correction at very low cost. The same procedure can also detect two flawed bits, for successful error detection but no correction.

The Hamming code detects errors in *data bits* of a data value by inserting error-correcting bits, called *parity bits*, in the data value. Different parity bits check different data bits of a data value to be protected against bit-errors, and each parity bit is set as a '1' if the total number of 1s in the data bits being checked by the parity bit is an *odd* number, and is set as a 0, otherwise. The steps involved in using the Hamming code to insert the parity bits into the data bits are given below, with the following layout showing how parity bits and data bits are arranged:

Bit position:

1 2 3 4 5 6 7 8 9 10 11...

Position occupied by parity or data bit:

$P_1P_2D_1P_3D_2D_3D_4P_4D_5D_6D_7 \ldots$

1. All bit positions that are powers of two are used as parity bits, labeled as P_i ($i \geq 1$), i.e., bit positions 1, 2, 4, 8, 16, 32, 64, etc.

2. All other bit positions are for the data to be encoded and are labeled as D_i ($i \geq 1$), i.e., bit positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.

3. Each parity bit P_i ($i \geq 1$) calculates the even parity according to only the data bits with the i^{th} bit (in the binary equivalent) of their position set to '1', and each P_i checks the combined parity, i.e., the number of bits, n, having their corresponding value in the data word set to '1' amongst the data bits being checked by P_i. $P_i = 1$ if n is odd; otherwise, $P_i = 0$.

By using the Hamming code, any error is detected and corrected, if possible, at the receiver site, which calculates the values for the Hamming code parity bits, called *calculated parity bits*, according to the data bits in the new (flawed) data word, i.e., the binary representation of a data word (with some bits flawed). The error is discovered when the receiver site compares the value of each parity bit in the new (flawed) data word to the value of the calculated parity bits. For each parity bit in the new (flawed) data word, the *parity bit check flag* is set as '1' if the parity bit present in the new (flawed) data word does not match the calculated parity bit; otherwise, the flag is set as 0.

The final step of the error detection and correction process is to evaluate the values of the parity bits check flag to correct the bit-errors, if any. The decimal equivalent of the parity bit check flag (always read in reverse order) is the position of the data bit with the error in the flawed data word (with parity bits). Flipping the data bit with the error in the flawed data word (with parity bits), and removing the parity bits yields the original data unflawed word. The same procedure can be used to detect errors in parity bits, when the parity bits are corrupted while no data bits are corrupted. (See a detailed example on error code detection and correction using Hamming code in Appendix A)

The Hamming code operates by considering a data item D that has to be provided with error detection and correction capabilities when transferring D between two network nodes, referred as the *sender* and *receiver*. Using the Hamming code method, the sender inserts parity bits into the bit string of D, and the receiver compares the values of the parity bits received to the values of the parity bits that it has calculated according to the bit string of D and detects and corrects any discovered errors, if possible, in D. In our load shedding approach, the sender is the sensor node generating (training) stream data, and the receiver is the site where the *CID* detection algorithm has been implemented. The sensor node inserts the parity bits into every tuple, which is treated as a separate data item D, in the training data set of data stream S. The error detection check is carried out individually on each training tuple right before the process of determining the *CID* of S. If any tuple in the training data set of S has an error, the receiver will attempt to correct the error; otherwise, the whole training data set is discarded, since the tuple with the uncorrectable error may cause the loss of the information about replicated attributes, which could lead to detecting an incorrect $RepA_s$, and subsequently the *CID*, of S. If a training data set of S is discarded, a new training data set of S is used to detect the *CID* of S and the error detection and correction process is carried out on the new training data set of S.

3.2.1.2 Time-based windows versus Tuple-based windows

In this section, we discuss the application of our load shedding approach by considering time-based windows (at different tuple rates) versus tuple-based windows. The major advantage of using tuple-based sliding windows is that the number of tuples it contains can be controlled easily and the number does not change with the alteration in the number of tuples being generated every second (line rate) at the source site. The number of tuples in a time-based sliding window, however, is a function of the line rate. Our load shedding approach is capable of working with both tuple-based and time-based sliding windows, since a time-based sliding window can be converted into a tuple-based sliding window, and vice-versa. A tuple-based sliding window can be constructed from a time-based sliding window by using the product of (i) the time-based window size and (ii) the line rate of the data stream as the size (i.e., number of tuples) of the tuple-based sliding window. For using time-based sliding windows, the data stream system would have to vary the window size with any change in line rate such that each window captures the number of tuples according to the cycle length of the data stream. Thus, whenever the line rate decreases, the time-size would have to be increased, and vice versa. Since it is required the number of tuples in each sliding window be equal to the cycle length of the data stream for generation of the rankings amongst the attributes correctly, the use of tuple-based sliding windows can more advantageous in our load shedding approach.

3.2.2 Exponential Moving Average

Before the load shedding scheme generator can be applied to a data stream S, we first compute the moving averages (MAs) of data in the current sliding window W of S, which smoothens the variations of data in W and forms the core of the preprocessing step. We discuss how MAs could be used as a preprocessing step in W, with just one data stream for the intra-stream load shedding problem and more than one data stream for the inter-stream load shedding problem.

Data in a data stream to be processed at the source site are presented in a sequence of sliding windows representing variant characteristics of different attribute data in the data stream, such as the diversity of data values for each attribute. Occasionally, data in a sliding window are found to have sudden and short-lived changes, deviating from the data stream variation properties, i.e., the variation of data values of attributes. For example, the attribute values of *precipitation* in a desert would generally remain more constant than other attributes, such as temperature and wind speed, in the same data stream over a long period of time. Since an attribute which varies less than another attribute is considered to be less-informative, we can claim that *precipitation* in the desert weather data stream is less-informative and is the candidate attribute to be shed from the corresponding weather information data stream. However, due to sudden and abrupt change in weather situations at a particular minute or two, there may be a lot of rains, causing a significant change in precipitation. The precipitation over the next hour, however, may remain relatively constant, i.e., low again and nearly constant. Though this abrupt change does not really represent the weather conditions in the desert on a regular, consistent basis, it may cause other informative attributes, such as *temperature*, being treated as less-informative (false positives) while the real less-informative attributes, i.e., *precipitation*, being treated as informative (false negatives). In order to (i) smoothen the data, (ii) suppress any short and sudden change in data, and (iii) reduce the false positives and false negatives, MAs is employed as a preprocessing step for determining less- and more-informative attributes in the current sliding window of a data stream, since MAs attempt to tone down the fluctuations to a smoothened trend so that distortions are reduced to a minimum in volatile data. Other real-world applications of MAs include tracking trends and signaling reversals, such as credit card authorization and monitoring systems, and interstate traffic monitoring systems. We consider the two most popular types of MAs, the *Simple Moving Average* (SMA) and the *Exponential Moving Average* (EMA), and compare the usability of the two averages and adopt one of them for preprocessing stream data for load shedding.

As SMA (given in Equation 3.1) applies equal weight to all the values in a list, EMA of the most recent values of a list are defined (given in Equation 3.2) as the *moving average* calculated by weighting *recent* values more heavily than *older* values in the list. The most recent value in a list is the newest value of the list, which is used along with other values in the list to calculate their EMA, and the older values in the list are the values which have already been used to calculate the previous EMA. Equation 3.2 uses a *multiplier* to decide the *weight* to be applied to the new value, unlike SMA where the new value and the old values are equally weighted.

$$SMA = (a_1 + a_2 + a_3 + \ldots a_n) \tag{3.1}$$

where a_n is the most recent value in a list $L = \{a_1, a_2, \ \ldots, \ a_n\}$ over which the SMA of L is being calculated.

$$EMA(a_n) = (a_n - EMA(a_{n-1}) \times Multiplier + EMA(a_{n-1})) \tag{3.2}$$

where a_n is the most recent value in a list $L = \{a_1, a_2, \ldots, \ a_n\}$ over which the EMA of L is being calculated, EMA(a_n) ($n \geq 1$) is the EMA for the most recent value in L over which EMA is being calculated, and *Multiplier* is the *weight* to the most recent value, and statistically proved to be 0.18 for most cases.

We have considered both SMA and EMA as the MA of the data values in the current sliding window of a data stream in the preprocessing step in our load shedding approach. Unarguably, the polished or smoothened data is desired to be close, and as accurate as possible, to the original data. Unlike SMA, EMA have the ability to stay closer to the actual data than SMA, and thus EMA is an obvious choice as the MA for our preprocessing step to smoothen the data in a current sliding window. (See an example in Appendix B, which demonstrates that EMA is a better choice over SMA.)

3.2.3 Load Shedding Scheme Generation and Re-Evaluation

As mentioned earlier, there are two design issues in load shedding scheme generation: (i) how much data should be shed, and (ii) which attributes should be shed. It should be noted that there are two load shedding schemes for each data stream, which are (i) intra-scheme load shedding scheme and (ii) inter-stream load shedding scheme. In this section, we present an approach that generates the load shedding scheme for a data stream, which handle the two design issues listed above.

3.2.3.1 Amount of data (Number of attributes) to be shed

Our intra-stream load shedding approach handles the data transmission problem of data streams when a data stream is to be transmitted over a network with capacity lower than the data rate of the data stream. We start out by determining a load shedding scheme, which includes a ranking amongst the attributes, of a data stream being sent over the network with the most-informative attribute being ranked lowest. The first load shedding scheme of a data stream is created by using the first sliding window of a data stream, and for the subsequent updates to the load shedding scheme, only the current sliding window is used.

According to various studies in computer networks, it is known that every channel has a capacity depending on the noise and bandwidth of the channel. If data are transmitted at a rate *higher* than the capacity of the channel, then data transmission *errors* and collisions occur exponentially. To overcome these problems, the *data transmission rate* of a data stream should be regulated and lower than the *capacity* of the channel. In transferring stream data, attributes and their corresponding data are shed at the source site, if needed, to bring down the data transmission rate according to the channel capacity.

The *capacity* of a channel is defined by Claude Shannon and Ralph Hartley [Sha49] as the maximum amount of error-free digital data that can be transmitted over a communication link with a specified bandwidth in the presence of noise interference. Claude Shannon and Ralph Hartley provide a theoretical maximum rate of clean data C that can be transmitted through an analog communication channel subject with some noise interference as

$$C = BW \times log_2(1 + S/N) \tag{3.3}$$

where C is the channel capacity in bits (for our system we assume that the channel capacity C is known), BW is the bandwidth of the channel in hertz, *S/N ratio* is the signal-to-noise ratio of the communication signal to the Gaussian noise interference expressed as a straight power ratio (and not as decibels), and *S/N decibels* $= 10 \times \log_{10}(S/N\ ratio)$.

Example 2 According to various scientific studies for a telephone communication, the S/N decibels is often 20 dB and the bandwidth (BW) available is 4 kHz, where 20

$= 10 \times \log_{10}(S/N\ ratio)$, which implies $S/N\ ratio = 100$, and thus $C = 4 \times \log_2(1 + 100) = 4 \times \log_2(101) = 26.63$ Kbps, which is the theoretical maximum capacity for a dial-up connection with $BW = 4$ and $S/N = 20$ dB. Most of the dial-up connections claim a data rate up to 56.6 Kbps. Since the theoretical maximum capacity for telephone communication is 26.63 Kbps, the claim made by the dial-up connections is untrue. Thus, for most of the dial up network connections and wireless links, the capacity is lesser than the claim, and the need for load shedding becomes inevitable.

As a solution to the data transmission problem, we consider the other aspects of Shannon theorem. The Shannon theorem states that given a channel with information capacity C and information transmission rate R, if the information capacity is more than the transmission rate, i.e., $R < C$, then the probability of error at the receiver is made very small. This means that theoretically, it is possible to transmit information with nearly no error. The converse is also important, i.e., if $R > C$, then the probability of error at the receiver increases (exponentially) with R, which implies that no useful information can be transmitted beyond the channel capacity. Our intra-stream (inter-stream) load shedding approach is designed to maintain a transfer rate $R^{'}$, such that $R^{'} < C$. Whenever $R > C$, attributes are shed from a data stream being transmitted over the network, starting with shedding the less-informative attributes such that the transfer rate R falls to $R^{'}$ $(\leq C)$. The rate at which data has to be shed is $R - C$, such that R falls to $R^{'}$ $(\leq C)$, and thus the percentage of data to be shed is $(R– C)/R$. Assume that a data stream S has n attributes, then the number of attributes to be shed from S is

$$n^{'} = \lceil((R - C)/R) \times n\rceil \tag{3.4}$$

Example 3 Consider a channel with capacity C of 120 Kbps and an attempt to transmit a data stream with seven attributes at a transmission rate R of 160 Kbps. Since $R > C$, the transmission would produce errors. To attain error-free transmission, the number of attributes should be shed (using Equation 3.4) are $\lceil ((160 - 120)/160) \times 7 \rceil$, i.e., 2. Figure 3.1 shows how the number of attributes to be shed varies with (i) the capacity of the channel and (ii) the data transmission rate of a data stream. As shown in Figure 3.1, the number of attributes to be shed increases linearly with increase in data rate when the channel capacity is kept constant.

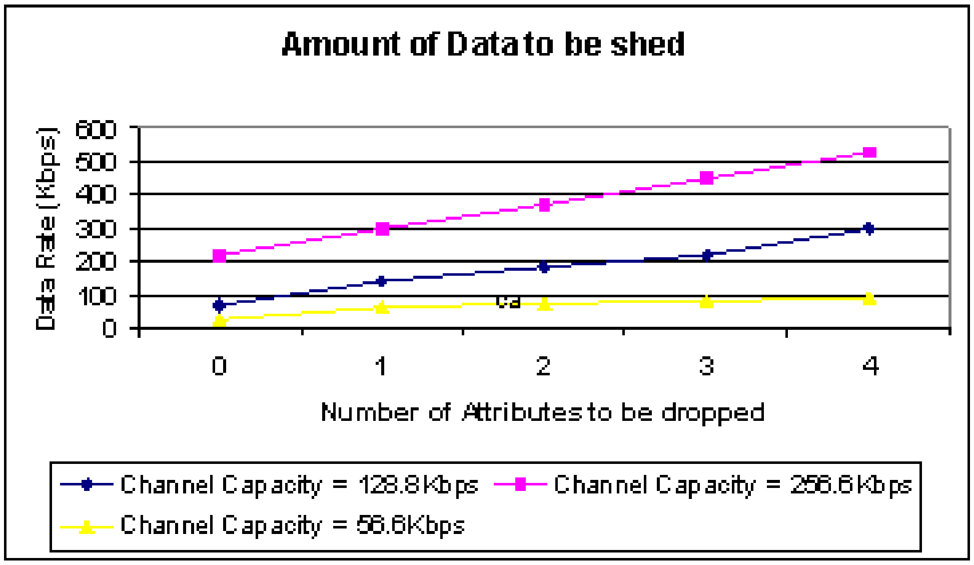


Figure 3.1: Variations in the number of attributes to be shed with various channel capacities and data rates on a data stream with seven attributes

3.2.3.2 Attribute Shedding Using Standard Deviation

Besides determining the number of attributes to be shed from a data stream, the load shedder also needs to know *which* attributes of the data stream to shed. *Standard* *deviation* can be used as a measure to determine a ranking with less-informative attributes being ranked higher amongst the attributes under consideration for shedding, which applies to each sliding window of a data stream that is updated/replaced on a continuous basis.

In probability and statistics, ***standard deviation*** is the most commonly-used measure of statistical dispersion[2]. Simply put, standard deviation measures how spread out the values in a list of data is. Thus, standard deviation is a measure of the spread of a list of data values from the mean value. A large standard deviation indicates that the data points are far from the mean, whereas a small standard deviation indicates that they are clustered closely around the mean. For example, given the three sample lists of data items <0, 0, 14, 14>, <0, 6, 8, 14>, and <6, 6, 8, 8>, each has an *average* of 7. Their *standard deviations* are 7, 5, and 1, respectively, which indicate that the third list has a much smaller standard deviation than the other two because its values are all close to 7. We conjecture that lists of values that are more closely bound, i.e., having less variation in its data values, are "less-informative," whereas lists of valued which are less closely bound, i.e., having more variation in its data values, are "more-informative." Since standard deviation is a measure of how closely bound data values in a list are, we apply standard deviation to the data values of each attribute A in a data stream to calculate how closely the data values of A are.

After we have computed the standard deviation for all the attributes in the data stream, the attributes are ranked, with attributes having *lower* standard deviation ranked *higher* and attributes having *higher* standard deviation ranked *lower*. The attributes with *lower* standard deviation are *"less-informative,"* and the attributes with *higher* standard deviation are *"more-informative."*

[2] http://en.wikipedia.org/wiki/Standard_deviation

3.2.3.3 Re-Evaluation of a Load Shedding Scheme

It is required that the load shedding scheme of a data stream S be regularly re-evaluated as the standard deviations of different attributes in subsequent sliding windows of S may change, causing the load shedding ranking amongst the attributes to change. A *non-adaptive* load shedding scheme re-evaluation algorithm re-evaluates the load shedding scheme at regular intervals. One major problem with using a *non-adaptive* re-evaluation algorithm is that if the time interval is too short, the source site would be re-evaluating the load shedding scheme *too often*, which imposes the burden on the source site in terms of computational time required for re-evaluation. However, if the time interval is *too large*, the source site would not re-evaluate the load shedding scheme often enough, creating the risk of an obsolete load shedding scheme being used for a long time. The proposed re-evaluation algorithm in this thesis, however, resolves the time-interval problem, since the algorithm is *adaptive*, which starts out with a very small re-evaluation time interval, referred as the *original* time interval. For the first time, the proposed algorithm re-evaluates an existing load shedding scheme after waiting for the original time interval, and then checks if the re-evaluated (i.e., the newly generated) load shedding scheme of the current sliding window with smoothened data (due to EMA preprocessing) is differed (in terms of attributes to be shed) from the previous load shedding scheme (computed by using the previous sliding window with smoothened data). If the attributes to be shed are the *same*, the time interval is *doubled* so that the re-evaluation is invoked after a longer interval. However, if the attributes to be shed are *different*, then (i) the time interval is reset to the original time interval, since a change in the load shedding scheme has just been detected and we anticipate changes in the load shedding scheme in near future, and (ii) the load shedding scheme is also updated to be the modified load shedding scheme with new attribute(s) and data to be shed. The time interval keeps growing in its usual manner every time the anticipated change in the attributes to be shed is proved incorrect. Our adaptive load shedding scheme re-evaluation algorithm is shown in Algorithm 2.

Algorithm 2. *Load shedding scheme re-evaluation*

```
Input: (i) Set of tuples $S$ in the current sliding window,
  on which load shedding has to be performed, and (ii)
  the current load shedding scheme $C$
Output: The (updated) load shedding scheme
1. Initialize time $T$, i.e., $\Delta t := t := 1$ sec
2. While the data stream management system is running
      If current clock time $= T + \Delta t$, then re-compute
        the load shedding scheme using $S$
          (i) If the re-computed load shedding scheme
                $RS = C$, in terms of the attributes to be
                dropped, then $\Delta t := 2 \times \Delta t$
              Else
                (a) $\Delta t := t$
                (b) $RS := C$
          (ii) $T :=$ current clock time
  End While
```

The graph in Figure 3.2 shows how the load shedding scheme for a weather information data stream collected on September 11, 2005 varied when we applied the load shedding re-evaluation algorithm on the data stream. The graph shows that the

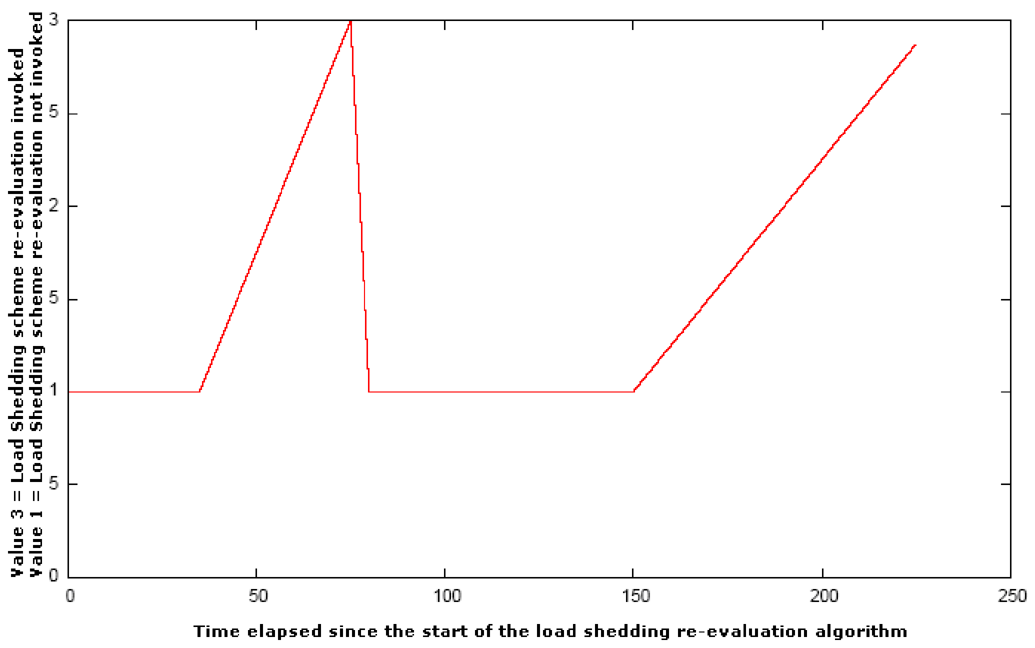


Figure 3.2: A sample application of our load shedding scheme re-evaluation algorithm on weather information data of world cities retrieved from www.yahoo.com/weather on September 11, 2005 at 3:00 PM (MST), where 3 indicates that the load shedding scheme re-evaluation has been invoked and other values indicate that the load shedding scheme re-evaluation has not been invoked.

load shedding scheme re-evaluation algorithm is able to detect changes in the load shedding scheme of weather information occurring even at irregular intervals, i.e., according to the frequency of changes in the load shedding scheme.

The adaptive load shedding scheme re-evaluation algorithm enjoys a major advantage over its non-adaptive counterpart, since the adaptive version notices the change more accurately in the load shedding scheme, whereas the non-adaptive load shedding scheme may not. Consider a data stream such that attribute A is the *least-informative* attribute during the first thirty minutes of every hour and attribute B is the *least-informative* attribute during the last thirty minutes of every hour. Assume that one attribute needs to be shed, and the non-adaptive load shedding scheme re-evaluation algorithm is invoked every hour, starting five minutes past the starting of the first hour. Since A is the *least-informative* attribute during the first thirty minutes of every hour, every time the load shedding scheme re-evaluation algorithm is invoked, A is found to be the *least-informative* attribute and is shed, and the load shedding scheme never changes. In such a scenario, the non-adaptive re-evaluation algorithm would fail to notice the change in the load shedding scheme. An adaptive load shedding scheme re-evaluation algorithm, however, adapts dynamic time-interval it follows regarding when the re-evaluation should be invoked according to the variation trend, which provides a more accurate mechanism in detecting a change in the load shedding scheme.

3.2.3.4 Preliminary verification of EMA and Standard Deviation

We have performed a number of preliminary experiments, using the proposed intra-stream load shedding approach, which have showed consistent and promising results, even on various data streams. For example, some data streams considered in the experiments include weather information stream data for capital cities of different countries across the globe with weather information stream data varying significantly from one capital city to another, whereas other weather information stream data include different cities within the same state with stream data for different cities varying lesser. The results are promising, since the ranking amongst the attributes is detected accurately. Shown in Table 3.2 is an extract of the training data retrieved from www.yahoo.com/weather on September 11, 2005 at 3:00 PM (MST).

Some tuples of the training data in the weather information data stream show unusually abrupt changes, which are not continual, and thus may not represent the actual frequency of changes of the corresponding attribute. To make sure that these abrupt changes do not cause any false positives (or negatives) in the training data, the training data were preprocessed using EMA on the data set of each attribute. Shown in Table 3.3 is the same set of training data after preprocessing.

We have compared the values for the temperature attribute, before and after the preprocessing step, as shown graphically in Figure 3.3, which clearly shows that the curve for the temperature values after preprocessing is smoother than the values before preprocessing. It shows that our preprocessing step using EMA was able to smoothen the data, and at the same time still maintain the smoothened data close to the original values.

Using the preprocessed weather information stream data as shown in Table 3.3 as the training data, we calculated the standard deviation value for each of the attributes to determine the ranking amongst the attributes. The ranking is shown in Table 3.4, which is generated by calculating the standard deviation on the stream data in Table 3.3.

The ranking amongst the attributes for the training data starting with the less-

Location	**Temp-erature**	**Dew Point**	**Baro-meter**	**Wind Speed**	**Humi-dity**	**Sun Rise**	**Visib-ility**	**Sun Set**
New Delhi	84	64	29.91	3	54	6.18	4.51	6.01
Ghurian	55	32	27.96	4	41	6.23	4.01	6
Abovyan	46	43	30.09	4	87	8.04	9.99	7.35
Baku	66	59	30.09	28	78	7.42	9.99	7.12
Al Manama	95	75	29.91	5	70	5.33	9.99	5.18
Brunei	97	73	29.8	7	59	6.07	9.99	6.1
Phnom Pen	96	77	29.77	3	74	5.5	9.99	5.47
Beijing	75	28	30.09	16	18	6.17	9.99	5.48
Gori	57	46	30.18	21	67	7.07	9.99	6.35
Abadan	82	55	29.94	5	39	6.14	9.99	5.55
Al Azamiyah	82	37	29.91	9	19	6.01	9.99	5.39
Acre	78	64	30	5	65	6.38	9.99	6.17
Akita	70	57	30	8	64	5.41	9.99	5.13
Alma Ata	66	34	30.09	4	30	6.56	9.99	6.23
Ch'ongjin	59	58	29.78	8	94	6.24	9.99	5.53
Al Jahrah	82	55	29.94	5	39	5.46	9.99	5.28
Bishkek	70	39	30.03	4	33	7.06	9.99	6.33
Vientiane	99	77	29.88	2	70	6.01	8	5.54
Ash Shuwayfat	81	66	30.06	2	65	6.37	8	6.14
Kuala Lumpur	99	75	29.83	8	62	6.59	9.99	7.03
Male	96	77	29.88	9	74	5.52	9.99	5.55
Rangoon	102	77	29.77	9	66	5.56	8	5.5
As Sib	91	68	29.94	3	52	6.01	9.99	5.49
Islamabad	88	59	29.94	5	38	6.06	5.01	5.44
Angeles	95	75	29.74	3	70	5.48	9.99	5.43
Al Wakrah	102	77	29.94	7	66	5.28	9.99	5.14
Abyar Ali	85	50	30	8	29	6.16	9.99	6.03
Panjang	97	73	29.8	9	59	6.5	9.99	6.56
Colombo	99	77	29.85	5	70	6.28	9.99	6.29
Al Mismiyah	64	54	30.06	6	68	5.33	9.99	5.11
Chang-hua	96	72	29.83	18	55	5.52	9.99	5.39
:	:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:	:

Table 3.2: Extracted weather information stream data with world cities, which serve as (training) data in a current sliding window, and were retrieved from www.yahoo.com/weather on September 11, 2005 at 3:00 PM (MST)

Location	**Temp-erature**	**Dew Point**	**Baro-meter**	**Wind Speed**	**Humi-dity**	**Sun Rise**	**Visib-ility**	**Sun Set**
New Delhi	84	64	29.91	3	54	6.18	4.51	6.01
Ghurian	63.7	41.6	28.55	3.7	44.9	6.22	4.16	6.01
Abovyan	51.31	42.58	29.63	3.91	74.37	7.5	8.25	6.95
Baku	61.6	54.08	29.96	20.78	76.92	7.45	9.47	7.07
Al Manama	84.98	68.73	29.93	9.74	72.08	5.97	9.84	5.75
Brunei Airport	93.4	71.72	29.84	7.82	62.93	6.04	9.95	6
Phnom Penh	95.22	75.42	29.8	4.45	70.68	5.67	9.98	5.63
Beijing	81.07	42.23	30.01	12.54	33.81	6.02	9.99	5.53
Gori	64.22	44.87	30.13	18.47	57.05	6.76	9.99	6.11
Abadan	76.67	51.97	30	9.04	44.42	6.33	9.99	5.72
Al Azamiyah	80.4	41.49	29.94	9.02	26.63	6.11	9.99	5.49
Acre	78.72	57.25	29.99	6.21	53.49	6.3	9.99	5.97
Akita	72.62	57.08	30	7.47	60.85	5.68	9.99	5.39
Alma Ata	67.99	40.93	30.07	5.04	39.26	6.3	9.99	5.98
Ch'ongjin	61.7	52.88	29.87	7.12	77.58	6.26	9.99	5.67
Al Jahrah	75.91	54.37	29.92	5.64	50.58	5.7	9.99	5.4
Bishkek	71.78	43.61	30	4.5	38.28	6.66	9.99	6.05
Vientiane	90.84	66.99	29.92	2.75	60.49	6.21	8.6	5.7
Ash Shuwayfat	83.95	66.3	30.02	2.23	63.65	6.32	8.18	6.01
Kuala Lumpur	94.49	72.39	29.89	6.27	62.5	6.51	9.45	6.73
Male	95.55	75.62	29.89	8.19	70.55	5.82	9.83	5.91
Rangoon	100.07	76.59	29.81	8.76	67.37	5.64	8.55	5.63
As Sib	93.72	70.58	29.9	4.73	56.61	5.9	9.56	5.53
Islamabad	89.72	62.48	29.93	4.92	43.59	6.02	6.38	5.47
Angeles	93.42	71.25	29.8	3.58	62.08	5.64	8.91	5.45
Al Wakrah	99.43	75.28	29.9	5.98	64.83	5.39	9.67	5.24
Abyar Ali	89.33	57.59	29.97	7.4	39.75	5.93	9.9	5.8
Panjang	94.7	68.38	29.86	8.52	53.23	6.33	9.97	6.33
Colombo	97.71	74.42	29.86	6.06	64.97	6.3	9.99	6.31
Al Mismiyah	74.12	60.13	30	6.02	67.1	5.62	9.99	5.47
Chang-hua	89.44	68.44	29.89	14.41	58.63	5.55	9.99	5.42
: :	: :	: :	: :	: :	: :	: :	: :	: :

Table 3.3: Data from Table 3.2 after preprocessing using EMA

Standard deviation for attribute Temperature is 12.07
Standard deviation for attribute Dew Point is 5.48
Standard deviation for attribute Barometer is 0.13
Standard deviation for attribute Wind Speed is 0.99
Standard deviation for attribute Humidity is 12.43
Standard deviation for attribute Sun Rise is 0.43
Standard deviation for attribute Visibility is 1.34
Standard deviation for attribute Sun Set is 0.26

Table 3.4: Ranking for the attributes in Table 3.3

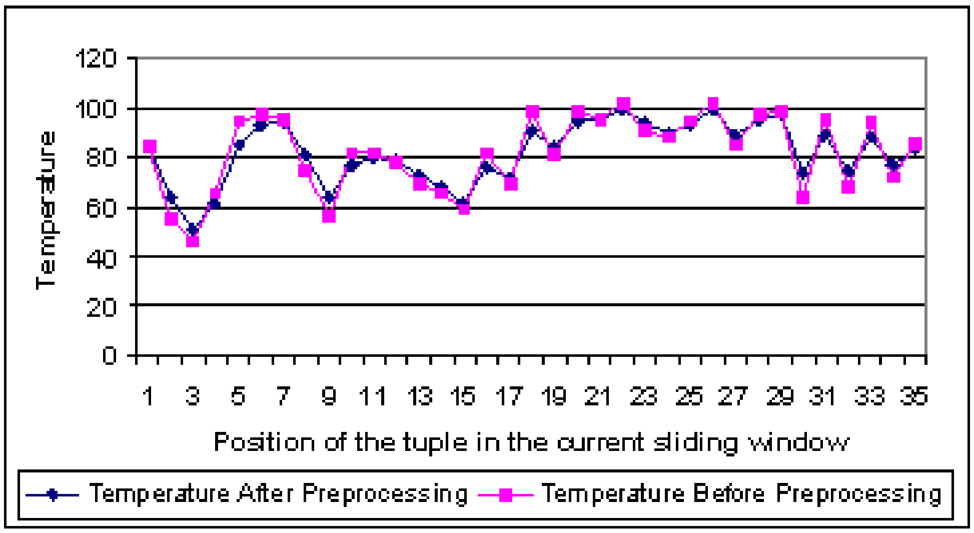


Figure 3.3: A Temperature stream data, as shown in Tables 3.3 and 3.4, before and after preprocessing, which were retrieved from www.yahoo.com/weather on September 11, 2005 at 3:00 PM (MST)

informative attributes is Barometer, Sun Set, Sun Rise, Wind Speed, Visibility, Dew Point, Temperature, and Humidity. These rankings are later used to generate the load shedding scheme, with Barometer being the first attribute to be chosen for shedding, and Humidity being the last.

Along with the weather information data stream, the stock exchange data stream was found to be another data domain in which the data vary more than the data in other data streams belonging to different applications domains, such as Internet traffic and road traffic. For this reason, we chose stock exchange, in addition to weather, information data stream, for conducting preliminary experiments to verify the performance of our preprocessing strategy on determining the less-informative attributes. We performed experiments on the stock data retrieved from the Web site *http://quotes.nasdaq.com/quote.dll?page=nasdaq100* on September 13, 2005 at 11:00 AM (MST). An extract of the source data from the raw training data is shown in Table 3.5:

Symbol	**Company**	**Last Sale**	**Net Change**	**Percent Change**	**Share Volume**	**NASDAQ 100 Index**
ADBE	Adobe Systems Incorporated	28.07	0.19	0.68	2878579	0.11
ALTR	Altera Corporation	18.68	0.13	0.7	3550201	0.08
AMZN	Amazon.com Inc.	42.16	0.08	0.19	2815159	0.03
APCC	American Power Conversion Corporation	25.25	0.25	1	507803	0.06
AMGN	Amgen Inc.	83.52	-0.48	-0.57	6421207	-0.34
APOL	Apollo Group Inc.	65.86	0.46	0.7	1375244	0.1
AAPL	Apple Computer Inc.	53.32	1.42	2.74	18031365	2.03
AMAT	Applied Materials Inc.	16.99	0.09	0.53	10651951	0.09
ATYT	ATI Technologies Inc.	13.56	0.29	2.19	3107687	0.09
ADSK	Autodesk Inc.	42.3	1.05	2.55	1750660	0.31
BEAS	BEA Systems Inc.	8.45	-0.11	-1.29	2392399	-0.05
BBBY	Bed Bath ; Beyond Inc.	40.02	0.32	0.81	2553449	0.16
BIIB	Biogen Idec Inc	38.76	-0.15	-0.39	2368033	-0.07
BMET	Biomet Inc.	35.92	0.24	0.67	954275	0.1
BRCM	Broadcom Corporation	45.1	0.89	2.01	5142755	0.28
CHRW	C.H. Robinson Worldwide Inc.	60.42	-0.67	-1.1	323476	-0.07
CECO	Career Education Corporation	36.54	0.5	1.39	1260953	0.06
CDWC	CDW Corporation	58.2	-0.3	-0.51	523510	-0.03
: :	: :	: :	: :	: :	: :	: :

Table 3.5: Training data, which served as data in a sliding window, were retrieved from *http://quotes.nasdaq.com/quote.dll? page=nasdaq100*, a stock exchange data stream, on September 13, 2005 at 11:00AM (MST)

Symbol	**Company**	**Last Sale**	**Net Change**	**Percent Change**	**Share Volume**	**NASDAQ 100 Index**
ADBE	Adobe Systems Incorporated	5.62	0.04	0.14	575715.8	0.03
ALTR	Altera Corporation	8.23	0.06	0.25	1170612.8	0.04
AMZN	Amazon.com Inc.	15.02	0.07	0.24	1499522.1	0.04
APCC	American Power Conversion Corporation	17.07	0.1	0.39	1301178.3	0.04
AMGN	Amgen Inc.	30.36	-0.02	0.2	2325184	0.04
APOL	Apollo Group Inc.	37.46	0.08	0.3	2135196	-0.01
AAPL	Apple Computer Inc.	40.63	0.35	0.79	5314429.8	0.4
AMAT	Applied Materials Inc.	35.9	0.3	0.74	6381934.1	0.34
ATYT	ATI Technologies Inc.	31.44	0.3	1.03	5727084.6	0.29
ADSK	Autodesk Inc.	33.61	0.45	1.34	4931799.7	0.3
BEAS	BEA Systems Inc.	28.58	0.34	0.81	4423919.6	0.23
BBBY	Bed Bath ; Beyond Inc.	30.87	0.34	0.81	4049825.5	0.22
BIIB	Biogen Idec Inc	32.45	0.24	0.57	3713467	0.16
BMET	Biomet Inc.	33.14	0.24	0.59	3161628.6	0.15
BRCM	Broadcom Corporation	35.54	0.37	0.88	3557853.9	0.18
CHRW	C.H. Robinson Worldwide Inc.	40.51	0.16	0.48	2910978.3	0.13
CECO	Career Education Corporation	39.72	0.23	0.67	2580973.2	0.12
CDWC	CDW Corporation	43.42	0.13	0.43	2169480.6	0.09
: :	: :	: :	: :	: :	: :	: :

Table 3.6: Data from Table 3.5 after preprocessing on September 13, 2005 at 11:00 AM (MST)

Table 3.6 shows the sample training set after applying the preprocessing step on its data. Comparing the values for "Last Sale" attribute in the stock exchange application domain in Figure 3.4, before and after the preprocessing step, we can clearly see that the curve for the training data values after preprocessing is smoother than the values before preprocessing.

Using the data set in Table 3.6 as the training data, we calculated the standard deviation value for each of the attributes to determine the ranking amongst the attributes. The results generated by the system are shown in Table 3.7. The ranking amongst the attributes for the stock training data set is NASDAQ 100 Index, Net change, Percent change, Share Volume, and Last Sale, which are later used to gener-

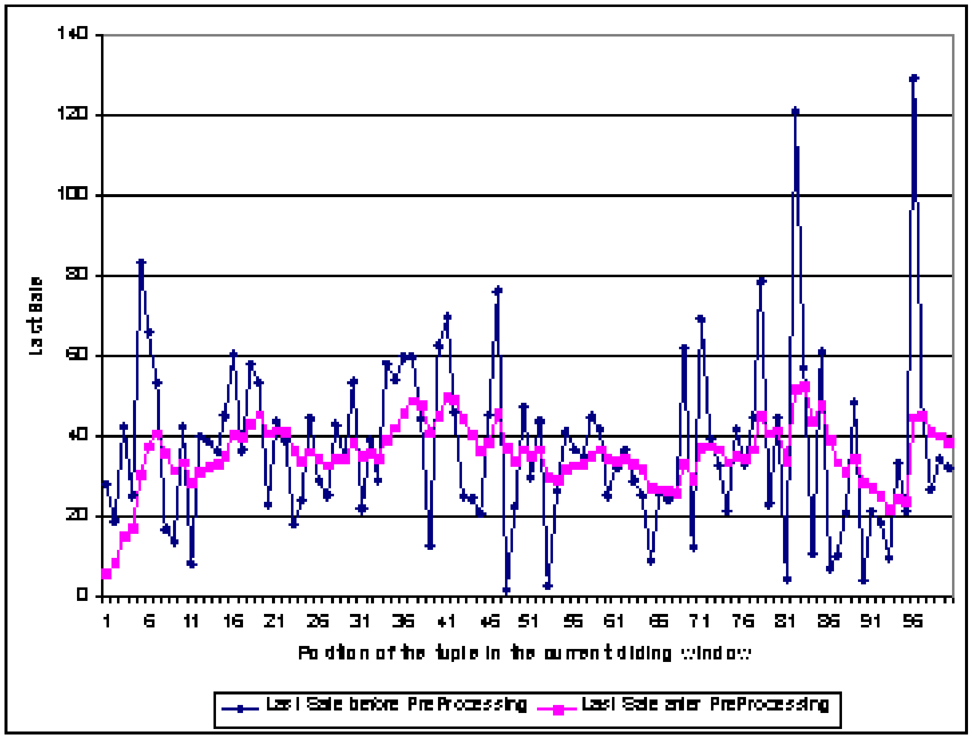


Figure 3.4: Data values of the *Last sale* attribute, as shown in Tables 3.5 and 3.6, before and after preprocessing, that are used to verify the performance of our preprocessing step on a stock exchange information data stream.

Standard deviation for attribute Last Sale is 6.92
Standard deviation for attribute Net Change is 0.15
Standard deviation for attribute Percent Change is 0.59
Standard deviation for attribute Share Volume is 5.68
Standard deviation for attribute NASDAQ 100 Index is 0.15

Table 3.7: Ranking for the stock exchange stream data in Table 3.6

ate the load shedding scheme on the corresponding stock exchange information data stream.

3.2.4 Recovering shed data

As discussed earlier, when needed, less-informative attributes are shed at the source site to reduce the data transmission rate (from the source site[3] to the destination site[4]) below the capacity of the transmission channel. It is possible that an end user at the destination site is interested in some of the attribute data shed at the source site. We present an approach to *recover* data at the destination site which was shed at the source site. In Section 3.2.4.1, we present our first data recovery approach, a method based on *Euclidean distances*. Hereafter, we draw the attention of the reader to the major drawback of using the Euclidean-distance based approach. In Section 3.2.4.2, we present our second data recovery approach, a method based on *synopsis matrices* to store information about shed data, which overcomes the drawback of the Euclidean distances approach.

[3]An example of a source site at which a data stream is generated is www.yahoo.com/weather.

[4] The site to which a data stream is to be transmitted, which is the Website of the data stream management system we have developed.

3.2.4.1 Euclidean Distance

In mathematics the *Euclidean distance*, or Euclidean metric, is the "ordinary" distance between any two points that one would measure with a ruler, which can be proven by repeated application of the Pythagorean Theorem. The Euclidean distance for any two points $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_n)$ in the Euclidean n-space is defined as

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_1 - y_1)^2} \tag{3.5}$$

To recover any shed data in a shed data stream, we once again make use of the current sliding window, as discussed earlier in Section 3.1. The current sliding window contains the most recent unshed data at the source site. When all (or some of) the data values for a certain attribute A in a set of tuples S, which makes up the content of the previous sliding window, at the destination site is to be recovered, the Euclidean distance between each tuple t in S and each tuple in the current sliding window C for all attributes, except A, is computed. For a tuple t in S, a tuple $i \in C$ with the Euclidean distance between t and i, which is less then the Euclidean distance between t and every other tuple j in C, is termed as the *nearest neighbor* of t. (It should be noted that the *CID* of the tuple is not used for the Euclidean-distance based method, it is used in the alternate recovery method we discuss later.) We formally define nearest neighbor as

$$N_S(t) = r, d(t, i) < d(t, j), t \in S, \forall i, j \in C \tag{3.6}$$

where $N_S(t)$ is the nearest neighbor of tuple$t$ and S is the set of tuples in the previous sliding window, and $d(t, i)$ is the Euclidean distance between tuples $t \in S$

and $i \in C$, the set of tuples in the current sliding window.

The data value of attribute A of the nearest neighbor $r(r \in C)$ of a tuple $t(t \in \mathrm{S})$ is the *recovered value* of A for t. Initial experiments have shown high data recovery accuracy of our Euclidean distance-based data recovery approach ranging from (i) *93.16%* for the test data in a weather information data stream with an error of *6.84%* to (ii) *96.81%* for an Internet traffic information data stream with an error of *3.19%*, the statistical data collected from performing a number of preliminary experiments using the randomly chosen sliding windows for the Yahoo weather and Internet-traffic-report data streams to compute the initial rankings amongst the attributes.

The experiment of the Internet traffic information and weather information domains showed very high recovered data accuracy. The average recovered data accuracy, which were conducted over 10 experiments, with the Internet traffic conditions varying from high traffic to low traffic, and weather conditions varying between high temperature and low temperature between each experiment, were *96.81%* and *93.16%*, respectively. Errors are noticed to increase as the distance between the current sliding window and the sliding window for which shed data has to be recovered increases, which is the reason for the errors noticed in this experiment.

The setup of the experiments included the source and the destination, both were simulated on the same machine, and the network channel, which was simulated by creating a pipeline between the two processes, one each for the source and destination site. The network channel acted as the low capacity channel between the source and destination sites. We started to shed data at the source site and transfer the remaining data to the destination. Since Euclidean distances are computed between (i) the (attribute values of) tuple for which data has to be recovered and (ii) the tuples in the current sliding window, there is no need to archive any data at the source site for the recovery purpose. We have attempted to recover the data, and the results were consistent, on the basis of which we can claim that the results are reliable. The experimental results conducted on stream data in an Internet traffic data stream are shown in Figure 3.5, whereas the experimental results on stream data in a weather information data stream are shown in Figure 3.6.

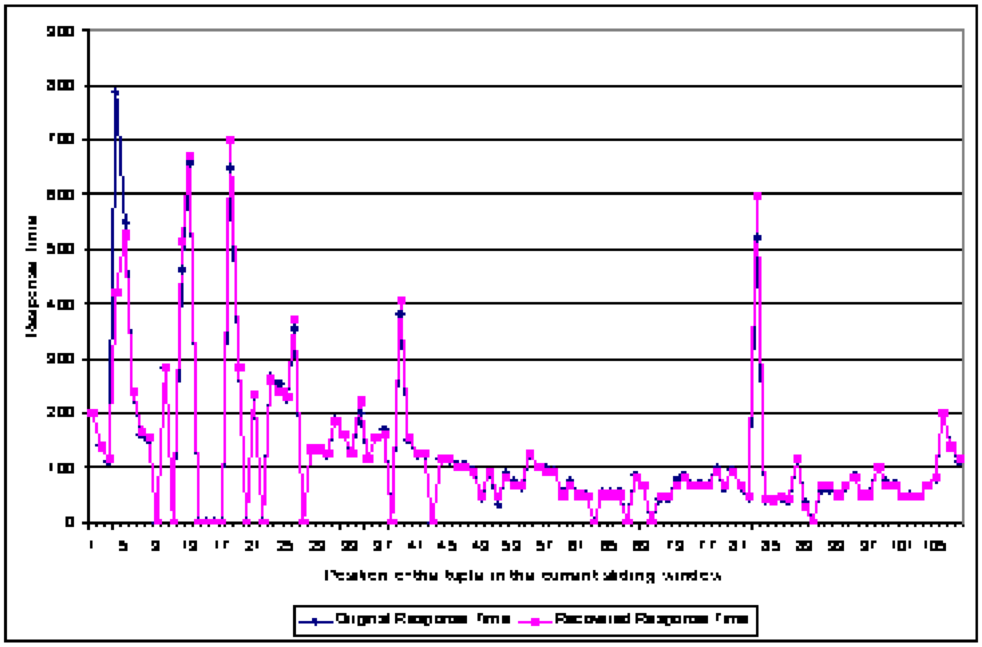


Figure 3.5: Original and recovered Response Time (test) attribute data in an Internet traffic information data stream retrieved from www.Internettraffic report.com on September 13, 2005 at 11:00 AM (MST)

The advantage of using the *Euclidean* distance-based data recovery approach is the ability of this method to recover shed data without requiring the source site to archive any data values that have been shed. This, however, also introduces the disadvantage of the Euclidean distance-based method, i.e., the requirement for the current sliding window and the tuple for which data values have to be recovered to be located at the same site, which is difficult to justify adopting the Euclidean distance-based method in shedding stream data, since the major design issue of our load shedding approach is to handle a low capacity channel between the source site and the destination site. Since the current sliding window is at the source site and the tuple for which data to be recovered is at the destination site, we would have to transport one of the two to the other site, which would add load to an already low capacity channel, an unacceptable requirement.

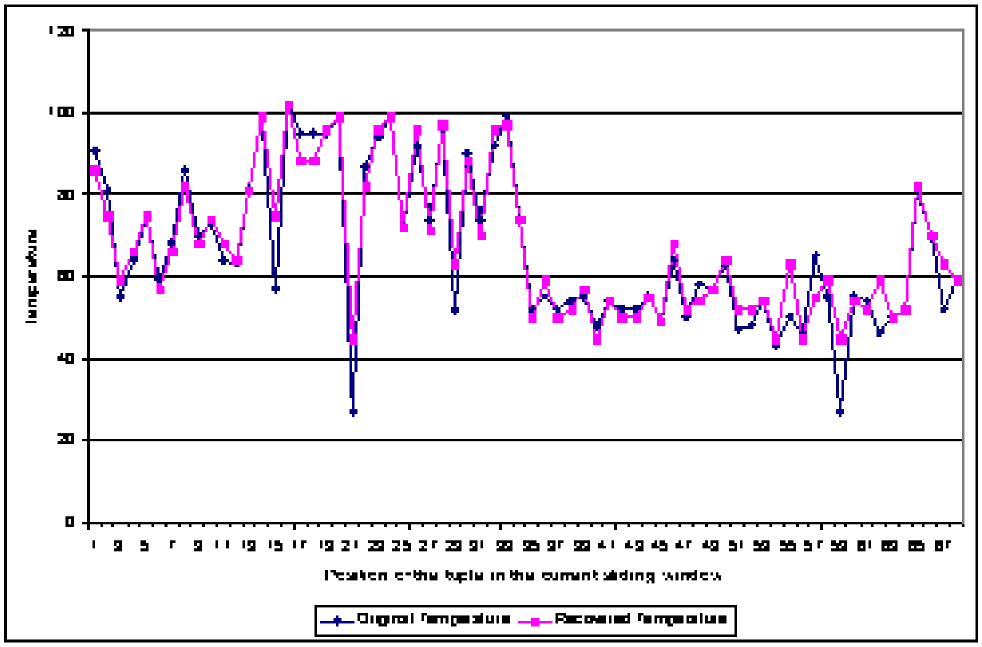


Figure 3.6: Original and recovered (test) data of temperature in a weather information data stream retrieved from www.yahoo.com/weather on September 11, 2005 at 3:00 PM (MST)

3.2.4.2 Synopsis Recovery Algorithm

In this section, we discuss the second data recovery approach which overcomes the drawback of the Euclidean-distance based approach, which we call the *Synopsis Recovery approach*. The basic idea behind the Synopsis Recovery approach (and its corresponding algorithm) is to store the shed data values in a matrix at the source site. This is carried out at the source site before attribute data are shed. Since storing all the shed values would require significant amount of memory/disk space and computational power at the source site, we propose a shed data recovery method that minimizes the memory/disk storage and computational power requirements.

As discussed in Section 3.2.1, we use the *CID* of a tuple for the recovery process. The data recovery process involves into two major modules. The first module, Module 1, maintains a synopsis of the shed data at the corresponding source site S, whereas the second module, Module 2, is the real recovery process when the destination site requests some shed data from S, which requires S to look up an (approximate) data value(s) in the synopsis and send the value(s) to the destination. Here, we introduce the concept of *error threshold* value, the acceptable error rate in data recovery (discussed in Section 3.2.4.3.) We do not consider the processing capacities of the source site and the destination site in determining the error threshold value as we have assumed in this thesis that the two sites have high computational power and the bottle-neck is the channel capacity. Furthermore, channel capacity is already considered indirectly in determining the error threshold value, since channel capacity influences the determination of the percentage of data should be shed and the number of attributes to be shed. The error threshold value varies from one application domain to another. For example, a patient information monitoring system would require more accuracy in recovered data than a weather information monitoring system. The synopsis maintenance module, i.e., Module 1, stores a value to be shed only when the value has been changed *more* than the error threshold value from what its last stored value was. The recovery values are stored in a 3-dimensional synopsis matrix with one dimension corresponding to each component: (i) the key value for the current tuple t, i.e., *CID(t)*, (ii) attribute A, i.e., $t[A]$, to be shed, and (iii) the *timestamp* for *CID(t)*. A *timestamp* is recorded in the synopsis (matrix) whenever a shed data value is recorded, which could be used to extract shed data values from the synopsis matrix at a later point in time, which is used in Module 2. Algorithm 3 shows the *Synopsis Recovery algorithm* for Module 1. The algorithm presents the procedure used to maintain the synopsis matrix. Every data value before being shed is compared with the most recent value of the same attribute stored in the matrix. If the change, i.e., the difference between the most recent value *mv* stored in the matrix and the value to be shed *sv* is more than the error threshold value E(to be discussed in details later), then *sv* is stored in the matrix; otherwise, *sv* is not stored, i.e., *sv* is stored in the synopsis recovery matrix only when $(|mv - sv|/sv) \times 100 > E$.

Algorithm 3. *Synopsis Recovery Algorithm*

```
Input: (i) Set of tuples $S$ in the current sliding window from where attribute data
          are to be shed, (ii) attribute $A$ to be shed, where the value of $A$ for tuple $t$ is
          denoted by $t[A]$, (iii) the key attribute value of tuple $t$, i.e., CID, (iv) the
          error threshold value $E$, and (v) the synopsis matrix, Synopsis.
Output: The (updated) synopsis matrix, Synopsis.
1. For each tuple $t \in S$
     $mv$ = Synopsis[CID] [A] [Timestamp(CID)], $sv = t[A]$
     /* the change in the value to be shed from the most recent value stored in
     the matrix is more than the error threshold */
     If $(|mv - sv|/sv) \times 100 > E$
        /* Increase the timestamp and store the value to be shed in the synopsis */
           matrix
     Then (i) If Timestamp(CID) = null, Then Timestamp(CID) = 0
                Else Timestamp(CID) = Timestamp(CID)+1
        (ii) Synopsis [$CID$] [$A$] [$Timestamp(CID)$] = $t[A]$
    Else
        /* Increase the timestamp by 1 so as to increase the timestamp for the
        $Timestamp(CID) = Timestamp(CID)+1$
```

The correctness of our shed data recovery approach is verified by experiments showing the accuracy in recovering shed data using the Synopsis Recovery approach to store recovery data in the synopsis matrix. Our preliminary experiments showed that high recovery accuracy can be achieved at the cost of storing very low percentage of the data shed as recovery data. We performed preliminary experiments on randomly chosen weather and stock exchange information data streams to verify the gain of using the Synopsis Recovery approach to maintain recovery data. The average amount of shed data for the weather information data domain, which had to be stored in the recovery matrix as (shown in Table 3.8), as recovery data with the recovery accuracy percentage of 90%, 95%, 98%, and 99% are 0.44%, 1.02%, 1.82%, and 7.77 %, respectively. The average amount of shed data for the stock exchange information data stream, which had to be stored in the recovery matrix (as shown in Table 3.9) as recovery data with the recovery accuracy percentage of 90%, 95%, 98%, and 99% are 4.88%, 18.11%, 42.66%, and 51.11%, respectively.

Storing all the shed data would mean storing 100% of the shed data, whereas with our Synopsis Recovery approach we store much lesser amount of the shed data (0.44%-7.77% with 90%-99% data recovery accuracy for a weather information stream data, and 4.88%-51.11% with 90%-99% data recovery accuracy for a stock exchange information stream data) in the synopsis recovery matrix as recovery data.

Example 4 Consider the weather information data stream for the capital cities of

	Data Recovery Accuracy			
	90%	**95%**	**98%**	**99%**
	2	2	4	19
	1	3	5	21
	0	2	6	18
Percentage of shed	2	2	5	16
data to be stored	1	4	4	22
in the synopsis	1	1	7	23
matrix as recovery	0	2	3	17
data for different	1	3	2	19
sliding windows of data	2	4	5	20
Average%	**0.44**	**1.02**	**1.82**	**7.77**

Table 3.8: The amount of shed data stored in the synopsis matrix for different data recovery accuracy ratios for a set of weather stream data over nine experiments, retrieved from www.yahoo.com/weather on September 11, 2005 at 3:00 PM (MST)

	Data Recovery Accuracy			
	90%	**95%**	**98%**	**99%**
	4	17	35	47
	6	21	50	56
	5	18	41	51
Percentage of shed	4	17	38	48
data to be stored	7	19	56	62
in the synopsis	3	15	32	42
matrix as recovery	6	21	52	56
data for different	4	17	37	46
sliding windows of data	5	18	43	52
Average%	**4.88**	**18.11**	**42.66**	**51.11**

Table 3.9: The amount of shed data stored in the synopsis matrix for different data recovery accuracy ratios for a set of stock exchange stream data over nine experiments, retrieved from http://quotes.nasdaq.com/ quote.dll?page=nasdaq100 on September 13, 2005 at 11:00 AM (MST)

different countries in Table 3.2 that requires the attribute *Barometer* to be shed, where the CID for the data stream is *location*. Assume that the *error threshold* value is 10%. In the 1st sliding window, the 1st tuple has location '*New Delhi*,' and its *Barometer* value is 29.91, and the 2nd tuple has location '*Ghurian*,' and its *Barometer* value is 27.96. Since the tuples have the first occurrences of locations *New Delhi* and *Ghurian*, the timestamps for both are 1, and these values are stored before they are shed from the data stream, which yield

Synopsis [*New Delhi*] [*Barometer*] [1] = 29.91, and

Synopsis [*Ghurian*] [*Barometer*] [1] = 27.96

In the 2nd sliding window, the 76th tuple has location '*New Delhi*,' and its *Barometer* value is 30.08, whereas the 77th tuple has location '*Ghurian*,' and its *Barometer* value is 31.64. (Note that these tuples are not shown in Table 3.2, since Table 3.2 includes only tuples in the 1st sliding window of the corresponding data stream.) Since the change in the *Barometer* for *New Delhi*, i.e., 0.17, is less than 10% of the previous stored *Barometer* value, i.e., 29.91, the new *Barometer* value for *New Delhi* is not stored. However, since the change in *Barometer* for *Ghurian*, i.e., 3.68, is more than 10% of the previous stored *Barometer* value, i.e., 27.96, the new *Barometer* value for *Ghurian* is stored, which yield

Timestamp[*New Delhi*] = Timestamp[*New Delhi*] + 1 = 2

Timestamp[*Ghurian*] = Timestamp[*Ghurian*] + 1 = 2

Synopsis [*Ghurian*] [*Barometer*] [2] = 31.64

Each tuple received by the destination site is marked by the destination site with a *timestamp* in an increasing order, with each tuple assigned a timestamp greater than one from the timestamp assigned to the previous tuple. In the case of inter-stream load shedding, with multiple streams flowing to the same destination site, the destination site maintains separate individual *timestamp* for each data stream, and appends the *timestamp* for each data stream with a unique source site identifier, such as A for data stream 1, B for data stream 2, and so on. When Module 2 is invoked, the destination site sends the source site the timestamp(s) and the CID of the tuple(s) for which the destination is interested in recovering. With each timestamp, the source site looks up the stored value in the synopsis matrix, which is ordered by the timestamps and the CIDs, for the value shed at the timestamp and sends the requested value to the destination. If no value was stored in the synopsis matrix for the given timestamp, i.e., when the difference between the most recent value stored in the synopsis matrix and the value to be shed is less than or equal to the error threshold, i.e., $((|mv - sv|/sv) \times 100 \leq E$, as shown in algorithm 3), the source site sends the value stored for the latest timestamp that is *smaller* than the timestamp for which the destination site had requested recovery. If a value for the *same* timestamp as the one sent by the destination is found in the synopsis matrix, then the recovered value is exactly the same value which was shed.

3.2.4.3 Error Threshold Value

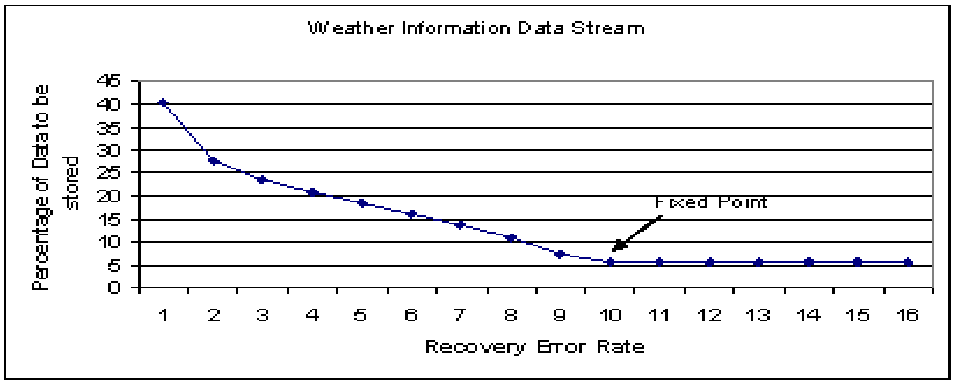
Our data recovery method applied to a data stream S must satisfy two criteria in order to perform well: (i) the amount of data in S to be stored in its synopsis matrix should be *low*, since we do not have infinite disk space for the synopsis matrix, and

(ii) the error in recovered data of S should also be low, i.e., the *recovery error rate* of S should be sufficiently *low.* Considering the two tasks closely, these two measures are *inversely* proportional to each other. When we attempt to *decrease* the amount of synopsis data to be stored, the recovery error rate increases and thus *suffers*, whereas when we attempt to *improve* the recovery error rate by decreasing it, the amount of synopsis data to be stored would *increase* proportionally. The optimal performance can be achieved by maintaining a balance between these two tradeoffs, which vary from one data stream application domain to another. For example, in critical data stream domains (such as medical information), a recovery error rate of 10% may be inadequate and the balance may be derived at a point with extremely low recovery error rate at the cost of larger amount of synopsis data to be stored. On the other hand, in a less critical data stream domain (such as weather information), a balance with higher recovery error rate at the benefit of lesser amount of data to be stored in the synopsis matrix is acceptable. In this section, we introduce (i) the *fixed point* for a data stream, (ii) the various categories that define the criticality of stream data, and (iii) the category recovery error rate. The two, (i) the fixed point for a data stream S and (ii) the category recovery error rate for the category to where S is assigned, are used to determine the error threshold value of S.

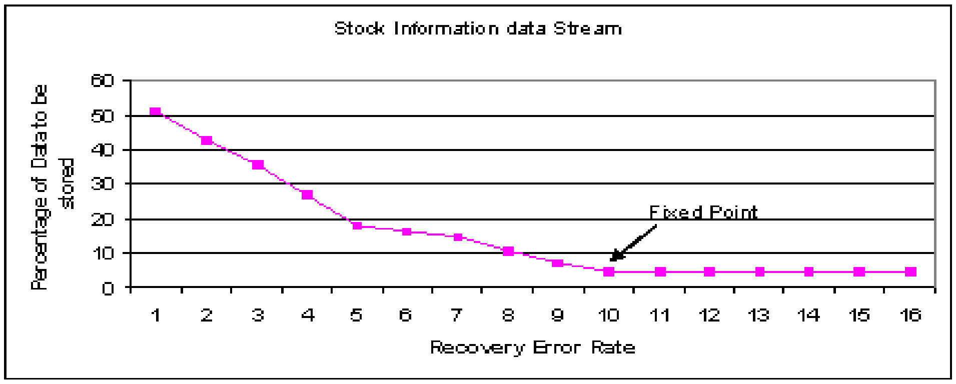
We first consider the size of the synopsis matrix of a data stream S in determining the potential error threshold value of S in our data recovery approach. Based on our observation, as the recovery error rate of S increases from 0% to 100%, the amount of synopsis data of S decreases to a point P beyond which any further increase in the recovery error rate does not affect the amount of synopsis data, i.e., the amount of synopsis data remains constant beyond P, which is referred as the *fixed point.* The existence of such a fixed point for S can be justified by the virtue of the fact that the variation in the values of an attribute A in S (to be shed) is finite, which is further strengthened by the fact that A is the least-informative attribute of S and thus least varying amongst the other attributes of S, i.e., the variation between most of the values of Awould be between 0% and x%, where x% is the percentage difference between the minimum and the maximum value of A. The amount of data to be stored in the synopsis matrix ceases to decrease as the recovery error rate is greater than or equal to x%. We determine the fixed point for a data stream by plotting a graph using the stream data of S, which captures the amount of synopsis data at each recovery error rate of S, starting out by increasing the recovery error rate of S from 0% to 100%.

Figures 3.7(a), 3.7(b), and 3.7 (c), show the average amount of data to be stored in the synopsis matrix at different recovery error rates using 10 experiments on each of the three different data streams, i.e., the weather information, stock information, and Internet traffic report from www.weather.yahoo.com, www.nasdaq.com, and www.internettrafficreport.com, respectively. The set of data used to construct each graph was collected over a 2-hour period on October 25, 2005 at 2:00PM (MST), which were split into 10 sets to conduct the 10 experiments on each of the data streams, and the results from the 10 experiments were then averaged. The averages of the fixed point values (i.e., the potential recovery error rates) for the 10 sets of data for the three data streams are 9.5%, 9.2%, and 20%, respectively. The graphs plotted in Figures 3.7(a), 3.7(b), and 3.7(c) show that the amount of data to be stored in each of the synopsis matrices becomes constant beyond the fixed point, as anticipated.

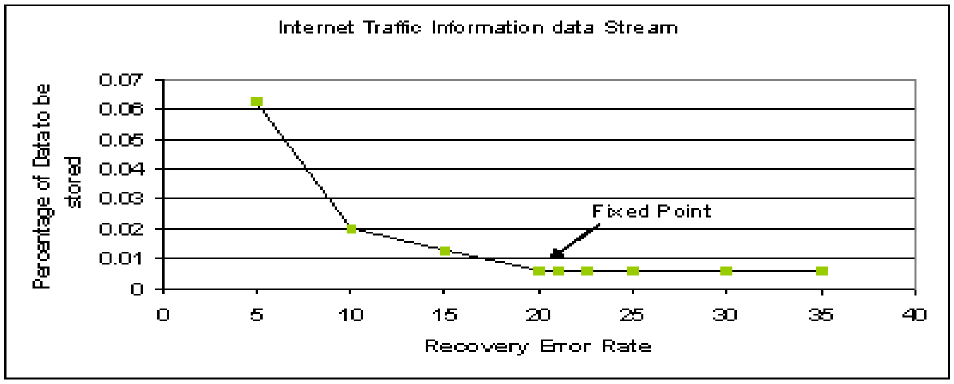
Although the fixed point approach can be adopted to determine the error threshold value of a data stream S automatically, it lacks the ability to incorporate the criticality of the data in S. We propose an automated recovery error rate detection



(a) Average amount of data to be stored in the synopsis matrix at different recovery error rates for the *weather information* data stream extracted from http://weather.yahoo.com on October 25, 2005 at 2:00PM (MST)



(b) Average amount of data to be stored in the synopsis matrix at different recovery error rates for the *stock exchange information* data stream extracted from $http : //quotes.nasdaq.com/quote.dll?\ page = nasdaq100$ on October 25, 2005 at 2:00PM (MST)



(c) Average amount of data to be stored in the synopsis matrix at different recovery error rates for the *Internet traffic information* data stream extracted from http://www.internettrafficreport.com on October 25, 2005 at 2:00PM (MST)

Figure 3.7: Amount of data to be stored for recovery

mechanism on any newly arrived data stream S in our DSMS, which determines another potential error threshold value of S based on the attribute names of S, which is done only once, i.e., prior to processing any data from S that is incorporated into our DSMS. For each new data stream S, the attribute names of S are analyzed and automatically matched with the attribute names of a data stream application domain in each of the predefined categories, which include the '*Extremely Critical*,' '*Very Critical*,' '*Moderately Critical*,' '*Low Critical*,' and '*Not Critical*' categories. Each of these categories is assigned a number of data stream application domains, such as the medical information data stream and emergency response data stream in the '*Extremely Critical*' category, and each data stream assigned to its corresponding category is associated with a predefined list of attribute names commonly found in the data streams of the same nature that best defines the corresponding application domain. We have predefined some of the commonly used data stream application domains in each category, e.g., medical information, stock exchange information, Internet traffic information, weather information, and population information in the '*Extremely Critical*', '*Very Critical*', '*Moderately Critical*', '*Low Critical*', and '*Not Critical*' categories, respectively. If the attribute names in a new data stream do not "match" any predefined list of attribute names of any one of the data stream application domains in any predefined categories, the new data stream is assigned to the category '*others*,' which has a category recovery error rate of "infinite", i.e., ∞, since it has no upper bound.

The matching of the list of attribute names for a new data stream with the predefined lists of attributes names is carried out by using the Fuzzy-Set IR model [YN05] and the distance matrix [Gar06]. We adopt the *fuzzy-set IR model* to compute the degree of similarity between (i) a predefined list of attribute names belonged to a particular category and (ii) the list of attribute names for a new data stream using the distance matrix [Gar06], in which row and column headings are words appearing in commonly used dictionaries. The *distance matrix* captures the degrees of similarity (i.e., correlation factors) among different words, which was generated using a set of Wikipedia [WIK] documents to compute the frequency of co-occurrence and relative distance of each pair of words in each Wikipedia document [Gar06]. Furthermore, we adopt the EQ function[5] [YN05], which is formally defined in Equation 3.7, to decide if any two lists of attribute names should be treated as the same using the correlation factors among the attribute names as defined in the distance matrix.

$$EQ(S_i, S_j) = \begin{cases} 1 & \text{if } MIN(Sim(S_i, S_j), Sim(S_j, S_i)) \geq \textit{Permission Threshold}, \wedge \\ & \quad |Sim(S_i, S_j) \text{ - } Sim(S_j, S_i)| \leq \textit{Variation Threshold} \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

After detecting the category that contains a list of attribute names that should be treated as equal to the list of attribute names in a new data stream S, S is then assigned to the category. We conducted 10 experiments in total for three different data stream application domains, i.e., the stock information, weather information, and Internet traffic information, using different randomly chosen data stream Web sites to demonstrate the accuracy of the Fuzzy-Set IR model approach in assigning data streams to categories, and the results showed *90%* accuracy. The 10% inaccuracy was due to the false positives and false negatives in matching the attribute names between two lists of attribute names.

The accuracy in assigning a new data stream S correctly to one of the categories

[5] The EQ function uses two threshold values, the *permission* threshold value and the *variation* threshold value, along with the correlation values of words. These threshold values are adjusted by us empirically for the purpose of computing the equality between any two lists of attribute names, and are presented in details in Chapter 4.

depends on the correct matching between the list of attribute names of S and the predefined list of attribute names in a data stream application domain, including synonyms or closely related words appeared in the two lists. Note that the list of attribute names in a data stream is quite "narrow," which means that two data streams belonged to the same application domain contain almost the same set of attribute names. For example, almost all data streams in the weather information domain contain the attributes *location*, *temperature*, *humidity*, *precipitation*, *sunrise*, *sunset*, and *wind*. Furthermore, two different lists of attribute names with synonymous or closely related names can still be accurately matched by using the Fuzzy-IR model [YN05]. For example, the attribute names '*precipitation*' and '*rain*' have very high correlation factor and are treated as the same attribute by the Fuzzy-IR model.

Example 5 Consider the predefined list of attribute names $L = \{$Location, Precipitation, Humidity, Temperature, Visibility, Sunrise, Sunset, Barometer, Dew Point, Wind$\}$ for the *weather* information application domain in the '*Low Critical*' category, where $V = \{$Symbol, Company Name, Last Sale, Net, Percent Change, Volume, Market Cap, Weight, Stock, Buy, Sell$\}$ for the *stock* information application domain in the '*Very Critical*' category. Consider the new data stream at http://www.bbc.co.uk/weather/5day.shtml?world=0100 with a list of attribute names $S = \{$Location, Max Day, Min Night, Wind, Visibility, Pressure, Relative Humidity, Sun Index, Pollution$\}$. The similarity values between V and S are $Sim(V, S)$ = 1.53E-6 and $Sim(S, V)$ = 1.41E-6 using the fuzzy set IR approach [YN05]. Applying the EQ function on these similarity values, using the threshold values of 0.173 for permission and 0.15 for variation threshold values, the EQ value is 0. Thus, the lists are treated as *different*, which is correct. However, $Sim(L, S) = 0.4$ and $Sim(S, L) = 0.3076$, and applying the EQ function with these similarity values, and using the same threshold values as given earlier, the EQ value is 1. Thus, the lists are treated as *similar*, which is correct.

Predefined with each of the first 5 categories is a category recovery error rate, and the 6^{th} category, i.e., '*others*', has the category recovery error rate of infinite. Each category recovery error rate of the first five categories is computed empirically during the design of our load shedding system, using the average fixed points of data streams in each category. The average of the fixed points of different data streams in a category is assigned as the category recovery error rate. The empirically determined category recovery error rates (see details in Chapter 4) of the first five categories are 1.67%, 4.7%, 4.81%, 13.5%, and 20.33% for the '*Extremely Critical*', '*Very Critical*', '*Moderately Critical*', '*Low Critical*', and '*Not Critical*' categories, respectively. Given the fixed point of the new data stream S that has been automatically detected and the category recovery error rate of S, our data recovery system assigns the error threshold value of S as the *lower* of the two, since the fixed point and the category recovery error rate are the maximum acceptable error rates in the recovered data for S. If the data stream S belongs to the '*other*s' category, the fixed point value of the new data stream is used as the error threshold value of S. Note that the fixed point value of a new data stream S is quite often different from the category recovery error rate of S because the fixed point value for S is computed for each new data stream when it is included in our DSMS, whereas the recovery error rate for S is the average of the fixed point values for a number of data streams belonging to that category.

Consider the categories for the data streams as shown in Figures 3.7(a), 3.7(b), and 3.7(c), which are '*Low Critical*', '*Very Critical*', and '*Moderately Critical*', respectively, with category recovery error rates 13.5%, 4.7%, and 4.81%. Comparing the category recovery error rates with the fixed point values, i.e., 9.5%, 9.2%, and 20%, respectively for the data streams shown in Figures 3.7(a), 3.7(b), and 3.7(c), respectively, the error threshold values for these data streams are 9.5%, 4.7%, and 4.81%, respectively, the lower of their respective fixed points and category recovery error rates.

3.2.5 The Architecture of the Intra-Stream Load Shedding Approach

Figure 3.8 shows the intra-stream load shedding architecture of our DSMS. The current sliding window of a data stream $S$is fed into the (intra-stream) load shedding scheme generation and re-evaluation sub-system, denoted *LSS-RES* (as in Section 3.2.3), which (i) preprocesses the data in the current sliding window, (ii) computes the rankings of the attributes of S, (iii) determines the number of attributes (and their corresponding stream data) to be shed, and (iv) generates the (intra-stream) load shedding scheme on a regular basis. The LSS-RES also re-evaluates the load shedding scheme on a regular basis. After generating (or updating) the load shedding scheme for S, the current sliding window is fed into the *data stream recovery data retainer*, denoted DS-RDR (as discussed in Section 3.2.4.2). Hereafter, the unshed current sliding window from the data stream is shed by the *shedder*, if needed. A data stream with or without undergone shedding by the shedder is referred as an '*intra stream*'.

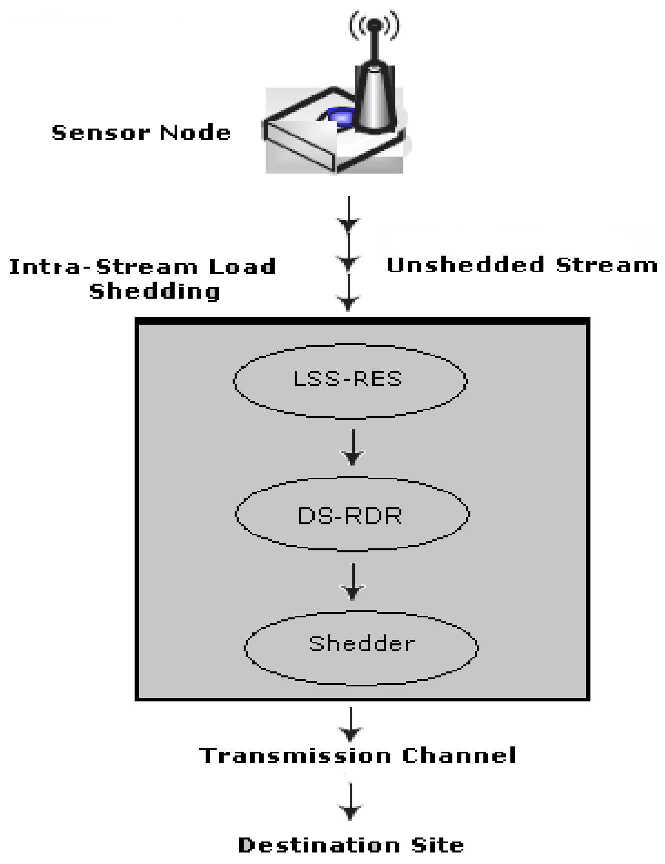


Figure 3.8: The architecture of the intra-stream load shedding and data recovery sub-system in our DSMS

3.3 The Inter-Stream Load Shedding Approach

Our inter-stream load shedding management system, denoted *Inter-LSMS*, which consists of (i) multiple *inter-stream load shedding sub-systems*, denoted *Inter-sub*, one for each intra stream, and (ii) a *central load shedder*, manages the transmission of multiple intra streams over a single multiplexed channel where the cumulative data rate of all the intra streams is sometimes more than the channel capacity. The central load shedder notifies each Inter-sub with the amount of data to be shed from its intra stream. To determine the amount of data to be shed from each intra stream, the central load shedder determines (i) the data transmission rates of all the intra streams, and (ii) the channel capacity of the single multiplex channel, and uses these information to calculate the amount of data to be (further) shed from each intra stream.

The basic architecture of our Inter-LSMS is shown in Figure 3.9. Each Inter-sub consists of the same basic components as in the intra-stream load shedding sub-system. Unshed source data streams, labeled S_1, S_2, …, S_n in Figure 3.9, are originated from different sources and transmitted through different channels, called *simplex channels*, with a single stream transmitted over a single simplex channel, to reach the Inter-LSMS, as intra streams, i.e., S'_1, S'_2, …, S'_n, where they are transmitted on the same channel, called the *multiplexed channel*.

Even though DS-RDR and the shedder in the Inter-sub and each intra-stream load shedding sub-system behave the same, the LSS-RESs in the two sub-systems are different, since the former is notified by the central load shedder the amount of data to be (further) shed from its intra stream. Each LSS-RES of an Inter-sub individually determines the less-informative attribute values in its intra stream to be shed. LSS-RESs in other Inter-subs reduce the combined data rate to be less than the

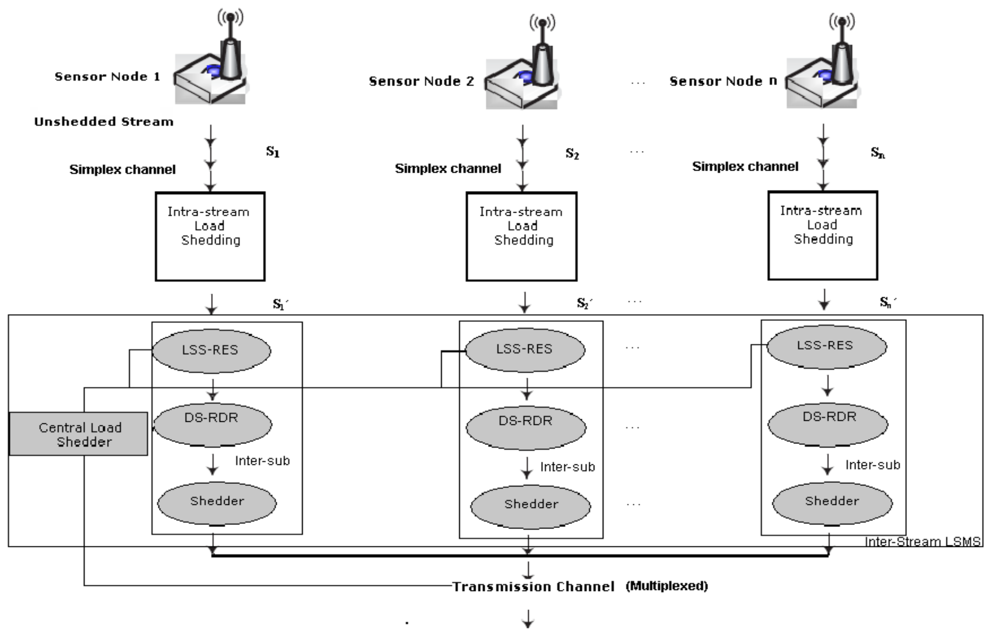


Figure 3.9: Load shedding sub-system architecture in our DSMS

multiplexed channel capacity. After generating (or updating) the inter-stream load shedding scheme at each Inter-sub, the current sliding window for each intra stream is fed into its data stream recovery synopsis generator, which is a separate synopsis recovery matrix compared with the one used at the corresponding intra-stream load shedding sub-system. Hereafter, the current sliding window of an intra stream is processed by its corresponding inter-stream shedder that sheds the data, if needed, according to the inter-stream load shedding scheme. The current (shedded) sliding windows from all the streams are then transmitted over the multiplexed channel to the destination asynchronously, in parallel.

3.3.1 Amount of Data to be shed for Inter-Stream Load Shedding

In our Inter-LSMS, since each LSS-RESs of an Inter-sub cannot directly sense the capacity of the multiplexed channel, the central load shedder must inform the individual LSS-RESs about the *amount of data to shed* from their intra streams. The central load shedder, with two pieces of information, i.e., (i) the data transmission rates of each intra streams and (ii) the channel capacity of the multiplexed channel, computes the amount of data to be shed from each intra stream, and forward this information to its corresponding LSS-RES. Consider R as the *cumulative data rate* of all the data streams, which is computed by the central load shedder and let C be the *channel capacity* of the multiplexed channel, the data transmission rate at which data has to be shed from the accumulation of all intra streams is $R - C$. R must fall to $R^{'}$ ($\leq$ C), if $R > C$, and the cumulative percentage of data to be shed from all the data streams is (R– C)/R. If an intra stream $n(n \geq 1)$ has data tranmission rate R_n, the amount of data to be shed from n, i.e., R_n', is computed as

$$R^{'}_{n} = ((R - C)/R) \times R_n \tag{3.8}$$

where (R - C)/R is the *drop rate.*

The number of attributes to be shed from an individual intra stream by the corresponding inter-stream shedder is defined as:

$$N^{'} = \lceil (R^{'}_{n}/R_n) \times N \rceil \tag{3.9}$$

where $N^{'}$ is the *number of attributes to be shed* from the intra stream n, N is the *number of attributes* in n, R_n is the data transmission rate of n, and ${R_n}^{'}$ is the amount of data to be shed on n.

3.3.2 An Example of Inter-Stream Load Shedding

Example 6 Consider a multiplexed channel with capacity C of 256.6 Kbps with an attempt to transmit four intra streams with 6, 8, 10, and 9 attributes at data transmission rates 64.4 Kbps, 80.6 Kbps, 76.6 Kbps, and 72.2 Kbps, respectively by the Inter-LSMS. Thus, the cumulative data transmission rate R is 293.8 Kbps. Since 293.8 Kbps $>$ 256.6 Kbps, the data transmission would generate errors. To attain error-free transmission, our Inter-LSMS calculates the overall drop rate of the combined intra streams, which is (293.8 - 256.6) / 293.8) = 12.66%. To attain this drop rate, the data transmission rates at which the amount of data has to be shed from the first, second, third, and fourth intra streams (according to Equation 3.8) are $0.1266 \times R_1 = 8.153$ Kbps, 0.1266 x $R_2 = 10.2$ Kbps, $0.1266 \times R_3 = 9.69$ Kbps, and $0.1266 \times R_4 = 9.14$ Kbps. The number of attributes to be shed by the inter-stream LSMS from the first, second, third, and fourth data stream according to Equation 3.9 is $\lceil[(8.153\ /\ 64.4) \times 6]\rceil = 1$,$\lceil(10.2\ /\ 80.6) \times 8]\rceil = 2$, $\lceil(9.69\ /\ 76.6) \times 10]\rceil = 2$, and $\lceil(9.14\ /\ 72.2) \times 9]\rceil = 2$.

Chapter 4

Experimental Results

In order to verify the correctness of the design and justify the merits of the proposed load shedding and data recovery approach, we have conducted experiments using various sets of (test and/or training) data extracted from data streams of different application domains: weather information, stock Exchange information, and Internet traffic information, as shown in Table 4.1. Some data sets are used as training data as needed, details of which are discussed in subsequent sections of this chapter. We performed our experiments in real-time with data sets of the data streams in the three application domains varying in sizes, i.e., 15.6 GB, 21 GB, and 12 GB, respectively. Test and/or training data for each application domain were split into three sets of the same size, with each of the three sets used at the data transmission rates of 60 Kbps, 90 Kbps, and 120 Kbps.

Our load shedding and data recovery design was implemented in Java and tested on a Windows/Linux PC with a 3.2 GHz processor, 3.25 GB RAM memory, and 150 GB of hard disk space with Linux shell scripts running test scripts, whereas the experiments were conducted for our load shedding and data recovery system

Data Stream Source	On Each Data Set				
	Attributes of the Data Stream	Size (GB)	Number of Tuples	Sliding Window Size	Number of Sliding Windows
Weather Information 15.6					
http://weather.yahoo.com - Sets 1, 2, 3	Location, Temperature, Dew Point, Humidity, Sun Rise, Visibility, Sun Set, Barometer, Wind	2	28036790	75	373823
http://www.wunderground .com/ - Sets 1, 2, 3	Location, Temperature, Humidity, Wind, Pressure, Precipitation, Dew Point	1.2	37063068	500	74126
http://www.weather.com - Sets 1, 2, 3	Location, UV Index, Pressure, Dew Point, Wind, Humidity, Visibility	2	32819280	250	131277
Stock Exchange Information 21					
http://quotes.nasdaq.com /quote.dll?page= nasdaq100 - Sets 1, 2, 3	Symbol, Company Name, Last Sale, Net Change, Share Volume, Nasdaq100 , Index, Percentage Change	3	52516326	100	525163
http://finance.indiamart.com /markets/bse/ - Sets 1, 2, 3	Company Name, Last Price, Change, Percentage Change, Market Cap, Weight	3	64860370	488	132910
http://www.channelnewsasia .com/cna/finance /sg/stockmonitor.htm - Sets 1, 2, 3	Stock, Buy, Sell, Last Done, Volume	1	33288126	1235	26953
Internet Traffic Information 12					
http://www.Internettraffic report.com - sets 1, 2, 3	Router Name, Current Index, Response Time, Packet Loss, Maximum Delay, Minimum Delay, Average Delay	2	16332958	96	170134
http://average.miq.net /index.html - Sets 1, 2, 3	Router, Response Time, Packet Loss, Minimum Delay, Average Delay, Maximum Delay	1	11689810	50	233796
http://watt.nlanr.net/ ampmap_active.php - Sets 1, 2, 3	Site Name, Min, Mean, Max, StdDev, Loss	1	10866072	86	126349

Table 4.1: Data sources of test/training stream data collected on March 1,2006 (except for http://www.channelnewsasia.com/cna/finance/sg/ stockmonitor.htm, which was collected on March 29, 2006)

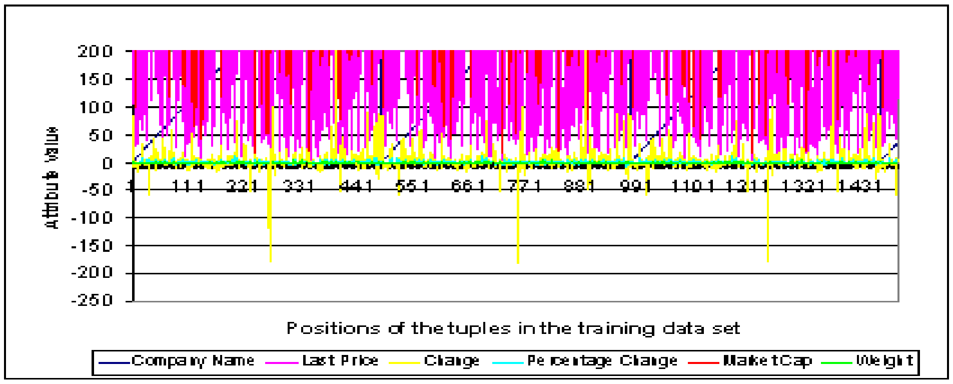
using various data transmission rates and channel capacities. To set various channel capacities of the system, we hard-coded into the load shedders of our implementation various channel capacities, i.e., 72, 98, and 132 Kbps to be tested at various data transmission rates, i.e., 60, 90, and 120 Kbps. These hard-coded values were used to determine the number of attribute to be shed, which directly affect the amount of shed data to be stored for data recovery purpose.

We performed experiments to verify the correctness of the four major modules of our load shedding and data recovery approach: the (i) *CID detection*, which also verified our approach on configuring the sliding window size of a data stream, in Section 4.1, (ii) *error threshold value generation*, for which we verified the accuracy in predicting the error threshold value of a data stream, in Section 4.2, (iii) *load shedding scheme (re-evaluation) and ranking generation approach*, which tested the accuracy of choosing the least-informative attributes in a data stream to be shed, in Section 4.3, and (iv) *shed data recovery*, for which we verified the accuracy of recovering shed data, in Section 4.4. Since the design of these modules is common to both intra-stream and inter-stream load shedding, verification of the design of these four modules verifies the design of both.

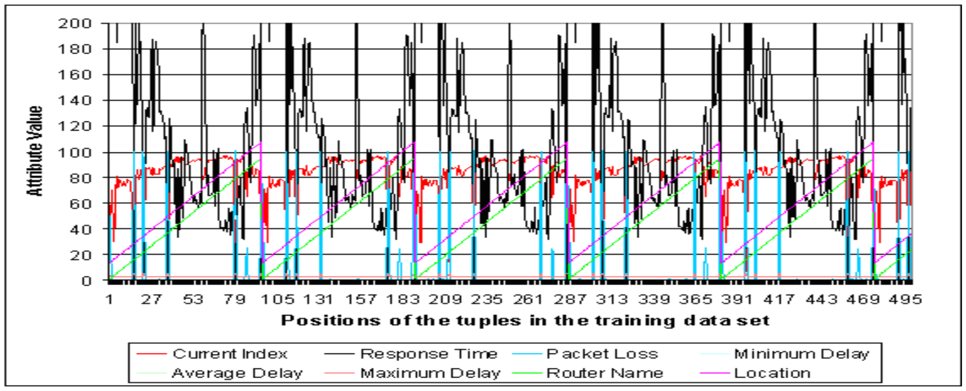
4.1 Experimental results on CID detection

If the attribute(s) of a training data stream S chosen as the CID of S has (have) the same replicated values (in the same sequence) within each sliding window of S, then the accuracy of our CID detection approach is confirmed. We used (i) various sets of training data from each of the three data stream application domains (as shown in Table 4.1) to detect the CID of the corresponding data stream and (ii) different sets of test data from the same source to verify the correctness of each detected CID. Note that the *sliding window size* of each data stream S used in the experiments was also verified along with the detection of the CID of S, since the cycle length of each detected CID yields the corresponding window size. The experimental results of detecting CIDs using the training data in Table 4.1 are partially shown in Figures 4.1(a), 4.1(b), and 4.1(c), whereas the verification results using the test data of the same data sources are shown in Table 4.2. Note that in each figure we have assigned a distinct numerical value to each alphanumerical or alphabetical attribute value (such as company name, location, router name, etc.) for simplicity on representation in a graph.

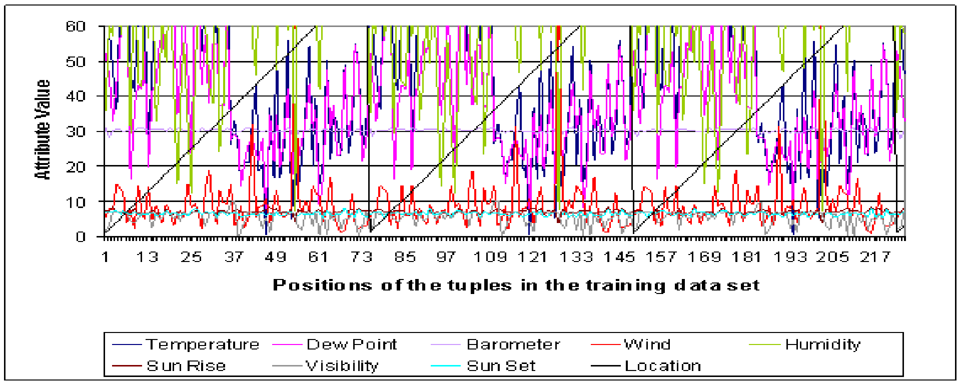
In the training data of Indiamart, a stock Exchange information data stream, the detected CID is '*Company Name*,' as shown in Figure 4.1(a), since it is the attribute having a repetitive curve, whereas '*Router Name*' is the detected CID of Internet Traffic Report, an Internet traffic information data stream, as shown in Figure 4.1(b), in addition to the detected CID '*Location*' of Yahoo, a weather information data stream, as shown in Figure 4.1(c). Table 4.2 shows the verification of the CIDs detected by using the training and test data. The first 10MB of each data set in Table 4.1 were used as training data and the remaining data were used as test data as shown in Table 4.2. Since each test data set was large in size, which made manual examination on the (correctly) detected CIDs infeasible, we verified that the detected CIDs are in fact the CIDs of the corresponding data streams using scripts; in addition, we manually examined a few randomly selected CIDs that were detected automatically. Based on the results as presented in Table 4.2, we conclude that the CID detection approach is 100% accurate.



(a) Experimental results generated by using the training data extracted from http://finance.indiamart.com/markets/bse/, a stock Exchange information data stream, on March 1, 2006 for testing the CID of the data stream



(b) Experimental results generated by using the training data extracted from http://www.Internettrafficreport.com, an Internet traffic information data stream, on March 1, 2006 for testing the CID of the data stream



(c) Experimental results generated by using the training data extracted from http://weather.yahoo.com/, a weather information data stream, on March 1, 2006 for testing the CID of the data stream

Figure 4.1: Experimental results for CID detection

Data Stream Source	**Training Data on Each Set**		**Test Data on Each Set**	
	Size (MB)	**Detected CID**	**Size (GB)**	**Detected CID**
Weather Information	**30**		**15.57**	
http://weather.yahoo.com/ - Set 1, Set 2, Set 3	10	Location	1.99	Location
http://www.wunderground.com/ - Set 1, Set 2, Set 3	...	...	1.19	...
http://www.weather.com/ - Set 1, Set 2, Set 3	...	...	1.99	...
Stock Exchange Information	**30**		**20.97**	
http://quotes.nasdaq.com/ quote.dll?page =nasdaq100 - Set 1, Set 2, Set 3	...	Company Name	2.99	Company Name
http://finance.india mart.com/markets/bse/ - Set 1, Set 2, Set 3	...	...	...	...
http://www.channelnewsasia.com/ cna/finance/sg/stock monitor.htm - Set 1, Set 2, Set 3	...	Stock	0.99	Stock
Internet Traffic Information	**30**		**11.97**	
http://www.Internettrafficreport.com - Set 1, Set 2, Set 3	...	Router Name	1.99	Router Name
http://average.miq.net/index.html - Set 1, Set 2, Set 3	...	...	0.99	...
http://watt.nlanr.net/active/maps/ ampmap_active.php - Set 1, Set 2, Set 3	...	...	...	...

*For each Set i(i = 1, 2, 3), the training data and test data are disjoint

Table 4.2: Training and test data results for CID detection

4.2 Experimental results on error threshold value detection

In this section, we present the verification of the correctness of our error threshold value detection approach, which include the (i) verification of the permissible and variation threshold values used in the EQ function that assigns a new data stream S to a criticality category, and (ii) the determination of the fixed point of S. In Section 4.2.1, we show how to determine the permissible and variation threshold values and introduce the pre-defined lists of attribute names for the three application domains, i.e., weather information, Internet traffic information, and stock Exchange information. According to the threshold values detected in Section 4.2.1, we verified the accuracy of our fixed point detection approach and the correctness of our approach in determining the category recovery error rate, which dictates the error threshold value, of a data stream S for assigning the criticality category to S in Section 4.2.2.

4.2.1 Verification of the permissible and variation threshold values in the EQ function

In defining the threshold values in the EQ function, we empirically adjust the permissible threshold value, which defines the *minimum similarity* between two lists of attribute names S_1 and S_2, i.e., $MIN(Sim(S_1,\ S_2),\ Sim(S_2,\ S_1))$, and the variation threshold value, which defines the *maximum dissimilarity* betweenS_1 and S_2, i.e., $|Sim(S_1,\ S_2)$ - $Sim(S_2,\ S_1)|$. The threshold values, along with the similarity values among the attribute names of S_1 and S_2, are used by the EQ function, which decides if S_1 and S_2 should be treated as equal. The EQ function determines to which category

C a new data stream S_2 should be assigned, which in turn determines the *category recovery error rate* CR for S_2, using the list of attribute names S_1 for C. The smaller of the CR and the fixed-point value of S_2 yields the error threshold value of S_2 (see Section 3.2.4.3 for details).

Using forty randomly chosen training data stream Websites with the Internet traffic, weather, and stock Exchange information application domains in mind (as shown in Table C.1), which were collected on April 8, 2006 and included data streams in the Internet traffic information (8), weather information (12), stock Exchange information (10), financial information (3), network data loss information (3), and chemical properties information (4), we determined the permissible and variation threshold values of the EQ function. The last three application domains in Table C.1, i.e., financial information, network data loss information, and chemical properties information, have attributes closely related to, but not the same as, stock Exchange information, Internet traffic information, and weather information, respectively, and were included in training our category assignment approach to demonstrate its precision in distinguishing the correct categories against categories that have closely related, but different, attribute names.

Suppose L_1 is a set of lists of attribute names, with one list of attribute names for each randomly selected data stream Website in Table C.1. Further assume that L_2 is a set containing three pre-defined lists of attribute names, one for each of the three application domains, i.e., weather information, stock Exchange information, and Internet traffic information. Each list in $L_1$was compared with each list in L_2 to determine the permissible and variation threshold values that yield the least total number of false positives and false negatives in matching the lists[1]. The false

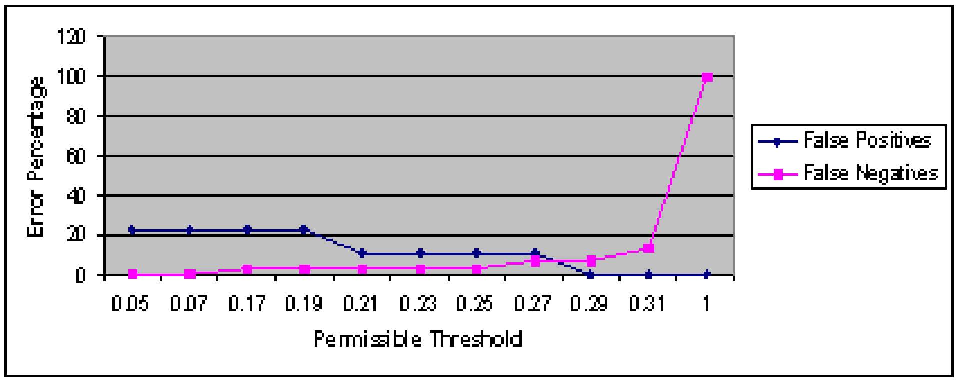
[1] *A false positive* (*false negative*, respectively) occurs when two lists of attribute names that are different but are termed as equal (are equal but are termed as different, respectively).

positives and false negatives were determined manually for each enumerated pair of lists of attribute names, with one list from L_1 and another one from L_2, which is in turn categorized as *equal* or *different* according to the EQ function computed automatically. The total number of detected false positives and false negatives according to various permissible threshold values are plotted in the graph as shown in Figure 4.2(a), which indicates that the ideal permissible threshold value is 0.29, which is the point with the least total number of false positives and false negatives[2], instead of 0.28, which although is the "intersected" point, but has a greater number of total false positives and false negatives.

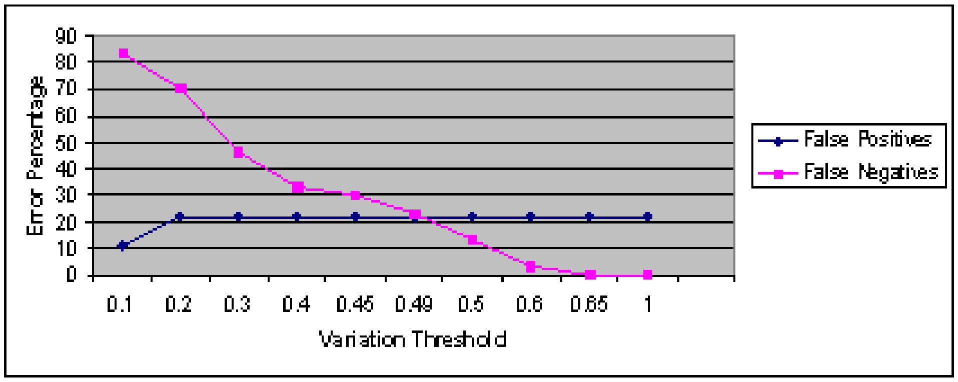
To obtain the variation threshold value of the EQ function, we manually categorized each pair of lists of attribute names from S_1 and S_2 as equal or different, where S_1 is the set of attribute names of a new data streams and S_2 is the set of attribute names of a pre-defined data stream. The manual categorization was followed by automatically categorizing S_1 and S_2 as equal or different according to different variation threshold values in the EQ function. The detected number of false positives and false negatives for the variation threshold value are plotted in the graph as shown in Figure 4.2(b), which indicates that the ideal variation threshold is 0.65, which is the point with the least total number of false positives and false negatives, instead of 0.49, which although is the "intersected" point, but has a greater number of total false positives and false negatives.

The list of pre-defined attribute names for each of the three application domains, i.e., weather information, stock Exchange information, and Internet traffic, were generated by including all the attribute names that occur in more than 80% of the at-

[2] As the permissible threshold value increases, the total number of attribute names in two different data stream treated as equal decreases, since more attribute names are treated as different in their corresponding data streams, and as a result, the false negatives increase, which is opposite in the case of false positives [YN05].



(a) Permissible threshold values



(b) Variation threshold values

Figure 4.2: Determining the permissible and variation threshold values in the EQ function using (training) data in forty randomly chosen data stream Websites in Table C.1

tribute lists of the data streams in the same application domain in Table C.1. The list of pre-defined attribute names for weather information are {*Location, Temperature, Humidity, Wind, Barometer*}, for stock Exchange information {*Company, Change, Volume, Percentage, High, Low*}, and for Internet traffic information {*Router, Packet, Loss, Delay, Average*}. We verified the accuracy of the permissible and variation threshold values detected in Figure 4.2(a) and 4.2(b) using a new set of forty randomly chosen Websites, with the three application domains that we have been considering in mind. The results on adopting the chosen threshold values (i.e., 0.29 for permissible, and 0.65 for variation) and the three pre-defined attribute lists for determining the application domain of a data stream are shown in Table C.2.

As shown in Table C.2, out of the forty randomly selected test Websites, thirty-seven were correctly categorized with 0 false positive and 3 false negatives, an accuracy rate of 92.5%, which justifies the accuracy of the EQ threshold values and the comprehensiveness of the lists of pre-defined attribute names for their corresponding attribute domains. Although we consider only three application domains for generating our pre-defined lists of attribute names and the EQ threshold values, our application domain detection method can be applied to any application domain D by determining the pre-defined list of attribute names of D by choosing the most commonly used attribute names for D from multiple data streams of D. Furthermore, the threshold value generation approach is applicable to other application domains as well.

We observed that the attributes that often cause false negatives are the attribute names that are not in any pre-defined list of attribute names, and are not closely related or synonymous to any of the attribute names in the pre-defined lists. For example, the attribute name '*StdDev*', an attribute in the second test data stream belonged to the Internet traffic information shown in Table C.2, is not in any list of pre-defined attribute names.

4.2.2 Verification of the error threshold value generation method

To verify our error threshold value generation module using the permissible and variation threshold values computed for the data streams in Table C.1 and set in Section 4.2.1, we conducted experiments using (i) training data, which were used to determine the fixed point values of the three different application domains considered in this thesis, along with the new *population information* application domain, which were collected on May 12, 2006, and (ii) the randomly chosen consecutive sliding windows from the test data, which evaluated the accuracy of the fixed point values of the corresponding data streams generated by the training data. The training data shown in Table 4.3 for each data stream was extracted from the corresponding data in Table 4.1 (and the remaining data from Table 4.1 were used as the test data as shown in Table 4.3), excluding the population information data. Since we use the population information data stream only for the error threshold generation experiments, it is only presented in Table 4.3, and no where else. The experimental results from the three sets of training data for each source data stream, which are followed by the results from the three sets of test data for the corresponding data stream, are shown in Table 4.3, which verified the accuracy of our fixed points generation method.

	Training Data		Randomly Chosen Test Data			
Data Stream Source	Data Size (MB)	Fixed Point (F)%	Number of Windows Sliding in the Randomly Chosen Data	Fixed Point (F)%	Actual Deviation = ($\mathbf{F}_{testdata}$ -$\mathbf{F}_{training\ data}$)	Deviation %
Stock Exchange Information		**4.7** (average)				
http://quotes.nasdaq.com/quote .dll?page=nasdaq100 - Set 1	100	1.3	25000	1.2	0.1	8.32
Set 2	...	1.25	...	1.29	0.04	3.10
Set 3	...	1.35	...	1.45	0.1	6.88
http://finance.indiamart .com/markets/bse/ - Set 1	...	5.6	...	5.8	0.2	3.44
Set 2	...	5.25	...	5.34	0.09	1.68
Set 3	...	5.5	...	5.75	0.25	4.34
http://www.channelnewsasia .com/cna/finance/sg/ stockmonitor.htm - Set 1	...	7.5	...	7.57	0.07	0.92
Set 2	...	7.65	...	7.73	0.08	1.02
Set 3	...	7.43	...	7.57	0.14	1.84
Weather Information		**4.81** (average)				
http://weather.yahoo.com/ - Set 1	200	4.5	...	4.55	0.05	1.08
Set 2	...	4.67	...	4.72	0.05	1.04
Set 3	...	4.4	...	4.61	0.21	4.54
http://www.wunder ground.com/ - Set 1	...	4.8	...	4.95	0.15	3.02
Set 2	...	4.85	...	4.67	0.18	3.84
Set 3	...	4.75	...	4.58	0.17	3.70
http://www.weather.com/ - Set 1	...	5	...	5.2	0.2	3.84
Set 2	...	5.2	...	5.05	0.15	2.96
Set 3	...	5.15	...	5.3	0.15	2.82
Internet Traffic Information		**13.5** (average)				
http://www.Internet trafficreport.com - Set 1	200	20	...	21.2	1.2	5.66
Set 2	...	20.4	...	19.6	0.8	4.08
Set 3	...	21.2	...	20.5	0.7	3.40
http://average.miq. net/index.html - Set 1	...	11.4	...	12.5	1.1	8.80
Set 2	...	11.2	...	12.3	1.1	8.94
Set 3	...	11.9	...	11.5	0.4	3.46
http://watt.nlanr.net/active/maps/ ampmap_active.php - Set 1	...	8	...	7.4	0.6	8.10
Set 2	...	8.5	...	9.2	0.7	7.60
Set 3	...	8.9	...	9.1	0.2	2.18
Population Information		**20.33** (average)				
http://www.mnsu.edu/emuseum/ country.php?FILE=INNAME	100	20	...	20.5	0.5	2.44

Continued on Next Page...

Table 4.3 – Continued

=India3C2Foptn3E - Set 1						
Set 2	...	21	...	21	0	0
Set 3	...	20	...	19.6	0.4	2.04
Average		**7.69**		**7.8**	**0.34**	**3.84**

Table 4.3: Experimental results of detecting various fixed points in data streams using training and test data in Table 4.1, in addition to the new population information stream data

We observed from Table 4.3 that the fixed points determined by using the training data set are very close to the fixed points determined by using the test data sets. The *deviation*, which determines the differences between the fixed point values computed by using the training data and the randomly chosen test data, calculated as |(fixed point$_{testdata}$ – fixed point$_{trainingdata}$)|/fixed point$_{testdata}$, ranges from *0%* to *8.94%*, as shown in Table 4.3. We explain the differences below.

Different sliding windows of a data stream S may contain different values for the same attributes. For some (current) sliding windows, if the value for an attribute is the same or its difference does not exceed error threshold compared with as the attribute value in the previous sliding window, then such a value is not stored in the synopsis recovery matrix of S; otherwise, an error occurs. Recall that the fixed point value of data stream S achieves the balance between the data recovery accuracy and the amount of recovery information to be stored in the synopsis matrix of S. On the basis of the low *deviation*, i.e., between 0% to 8.94%, with an average of 3.84%, as shown in Table 4.3, we draw the conclusion that our fixed point value detection approach works adequately.

Recall that the *minimal* of the fixed point value of a data stream S and the category recovery error rate of S yields the error threshold value of S. As the last step of verifying our error threshold value generation process, we empirically determined the category recovery error rate for the category of a new data stream. We set the values for the category recovery error rates for the stock Exchange information, weather information, Internet traffic information, and population application domains using the average of the fixed point values of the corresponding test and training data streams, as shown in Table 4.3, and used them to interpolate the category recovery error rate for the '*Extremely Critical*'[3]. The average of the fixed point values in the randomly chosen data streams for stock Exchange information, weather information, Internet traffic, and population information data domains are 4.7%, 4.81%, 13.5%, and 20.33%, as shown in Table 4.3, which are set as the category recovery error rates for the corresponding '*Very Critical*', '*Moderately Critical*', '*Low Critical*', and '*Not Critical*' predefined categories. The average fixed point value of each of these application domains turns out to be in the ascending order because of the nature (i.e., values) of the data. Using interpolation and considering the *ratios* of the category error rates of every other category as *equal*, a pattern that holds for the four categories, the category recovery error rate for the '*Extremely Critical*' category, denoted CRER(1), is calculated by using the formula CRER(1)/CRER(3) = CRER(2)/CRER(4). Using this formula, we interpolated the values for the '*Extremely Critical*' category, which is 1.67%. Thus, the category recovery error rates are 1.67%, 4.7%, 4.81%, 13.5%, and 20.33% for the '*Extremely Critical*', '*Very Critical*', '*Moderately Critical*', '*Low Critical*', and '*Not Critical*' categories, respectively. A new data stream S is assigned its error threshold value using its category recovery error rate if S belongs to one of the five pre-defined categories, assuming that at least one data stream is available in the '*Extremely Critical*' category; otherwise, it belongs to the '*others*' category, which has a category recovery error rate of "infinite", i.e., ∞ (as discussed in Section

[3] The category recovery error rate for the '*Extremely Critical*' category is interpolated due to the unavailability of any training or test data of the corresponding data stream, such as medical information, which includes private data that are unavailable on the Internet.

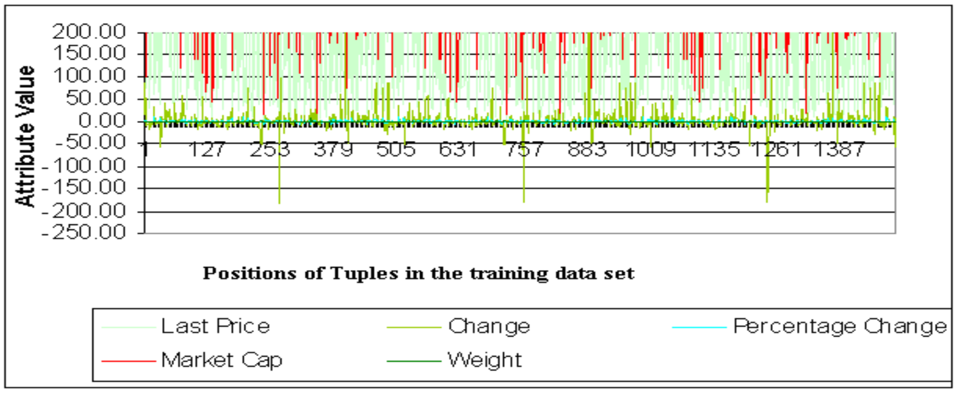
3.2.4.3), and the fixed point value of S is used as the error threshold value of S.

4.3 Experimental results on the load shedding scheme generation approach

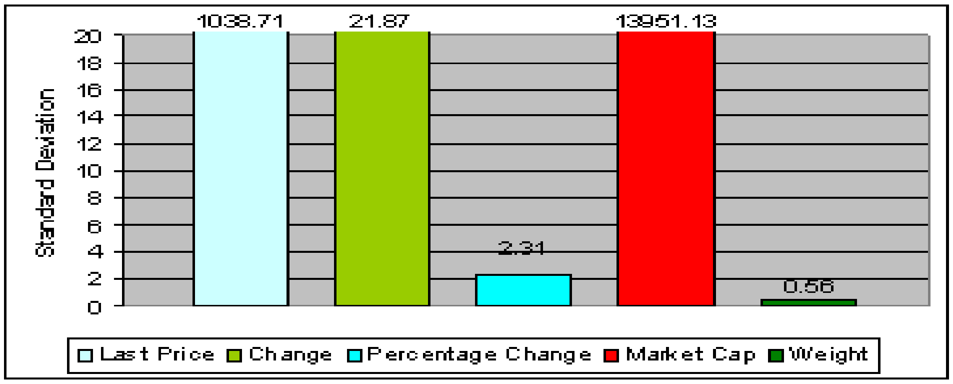
In this section, we evaluate our load shedding scheme generation and re-evaluation approach. In Section 4.3.1, we justify the accuracy of our least-informative attribute detection approach, whereas in Section 4.3.2, we assert the correctness of our load shedding scheme generation and re-evaluation approach. The least-informative attribute detection approach determines the attribute(s) to be shed, i.e., the attribute(s) composing the load shedding scheme, whereas the re-evaluation approach reconsiders the load shedding scheme for necessary modification.

4.3.1 Verifying the accuracy of detecting least-informative attributes

To verify the accuracy of our approach in determining the informativeness of an attribute, i.e., the ranking of attributes in each sliding window, of a data stream, which determines the attributes to be shed, we performed experiments on the test data of various data streams using standard deviations (since the informativeness is always computed in real-time using real stream data, and not once using training data). We captured the standard deviations of different attributes in a data stream using bar charts and verified that the attributes detected as least-informative are indeed less varying as compared to the attributes detected as more-informative. Figure 4.3(a) shows the variations in the values for different attributes of a stock Exchange information data stream, except the CID attribute, i.e., '*Company Name*', which cannot



(a) Experimental results generated by using test data retrieved from http://finance.indiamart.com/markets/bse/, a stock Exchange information data stream, on March 1, 2006 for detecting the least-informative attribute(s) in the data stream

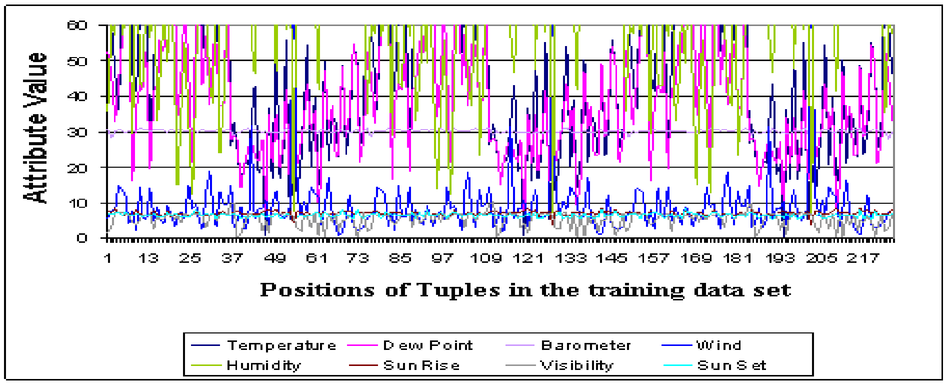


(b) Bar chart representation of the experimental results in Figure 4.3(a)

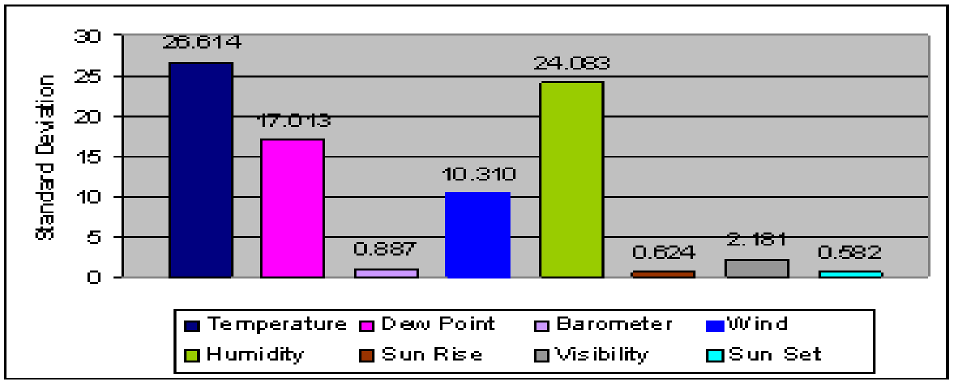
Figure 4.3: Least-informative attribute detection for Stock Exchange information data stream

be shed for data recovery purpose, whereas Figure 4.3(b) shows the standard deviation (rankings) for different attributes of the test data in Figure 4.3(a). According to Figure 4.3(b), the attribute with the highest ranking (i.e., the lowest standard deviation) is '*weight*', which is also the attribute with the least varying curve for its values as shown in Figure 4.3(a).

Figure 4.3(c) shows the variations in the values of different weather information data stream attributes extracted from Yahoo, except the CID attribute, i.e., '*Loca-*



(c) Experimental results generated by using test data retrieved from http://weather.yahoo.com/, a weather information data stream, on March 1, 2006 for detecting the least-informative attribute(s) in the data stream

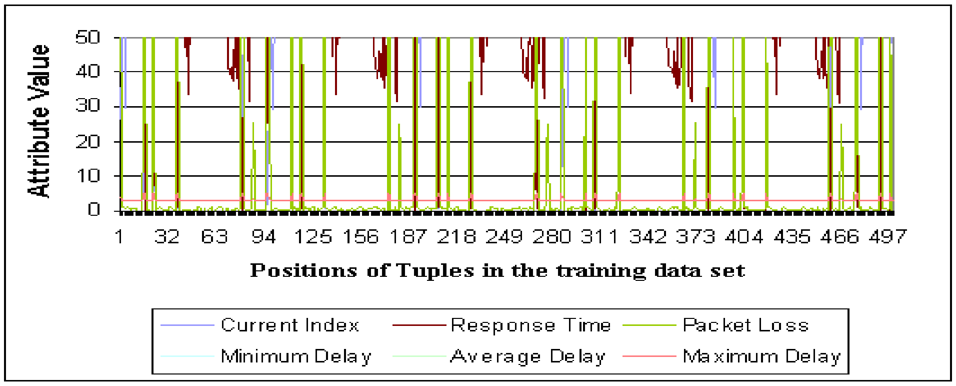


(d) Bar chart representation of the experimental results in Figure 4.3(c)

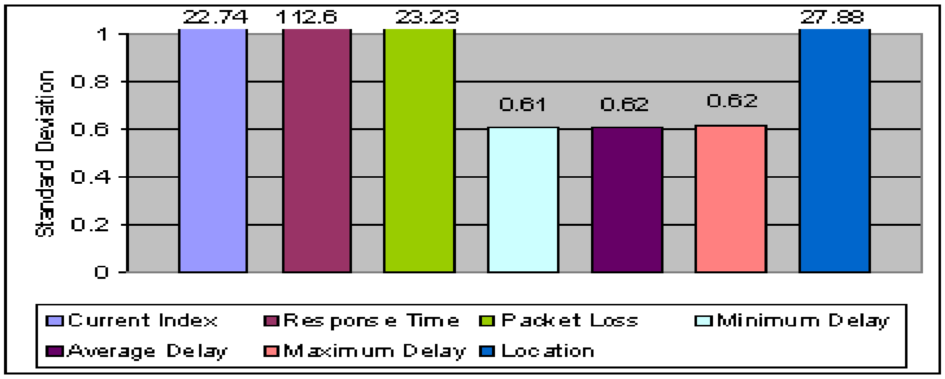
Figure 4.3: Least-informative attribute detection for Weather information data stream

tion', and Figure 4.3(d) shows the standard deviation (rankings) of the attributes. According to Figure 4.3(d), the attribute with the lowest standard deviation is '*Sun Set*', which is also observed in Figure 4.3(c) to be the attribute with the least varying curve for its values.

Figure 4.3(e) shows the variations in the values of different attributes of the Internet traffic information data stream extracted from Internet Traffic Report, except the CID attribute, i.e., '*Router Name*', whereas Figure 4.3(f) shows the standard deviation (rankings) of the attributes. According to Figure 4.3(f), the attribute with the lowest standard deviation is '*minimum delay*', which is also observed in Figure 4.3(e)



(e) Experimental results generated by using test data retrieved from http://www.internettrafficreport.com, a Internet traffic information data stream, on March 1, 2006 for detecting the least-informative attribute(s) in the data stream



(f) Bar chart representation of the experimental results in Figure 4.3(e)

Figure 4.3: Least-informative attribute detection for Internet traffic information data stream

to be the attribute with the least varying curve for its values. Note that '*minimum delay*', '*average delay*', and '*maximum delay*' are indistinguishable in Figure 4.3(e) due to their extremely small and overlapping values.

According to the experimental results on the three different data stream application domains, we conclude that our least-informative attribute detection approach determines the least-informative attribute(s) in a data stream with 100% accuracy. Such high accuracy is achieved because our least-informative attribute detection approach is based on calculating *standard deviation* on attribute values, which is a simple, mathematically sound, and widely accepted concept in statistics for detecting the variations in the values of a data set.

4.3.2 Verification of the correctness of the load shedding scheme generation and re-evaluation approach

As discussed in Section 3.2.3.3, our load shedding approach utilizes an *adaptive* load shedding scheme (which is re-evaluated at various time intervals) that defines the least-informative attribute(s) to be shed, and the load shedding scheme is generated in real-time on real stream data, and not on training data. The verification of the load shedding scheme involves verifying the correctness of the re-evaluation of the load shedding scheme in between the pre-defined time intervals. We manually computed each least-informative attribute A for each randomly chosen sliding window of a data stream (100 sliding windows for each application domain chosen from the data shown in Table 4.1) and compare A to the automatically detected least-informative attribute of various sliding windows of the same data stream generated by our load shedding approach. Each match of the manually determined least-informative attribute and the automatically generated least-informative attribute in each sliding window is called a *hit*, whereas each mismatch is called a *miss*, as shown in Table 4.4, with the same test data used in Table 4.1. The *misses* occur when the ranking of the attributes of a data stream changes in between two consecutive re-evaluations of the load shedding scheme, which is not reflected in (i.e., integrated into) the currently adopted load shedding scheme after the change has occurred and before the load shedding scheme is re-evaluated. This scenario occurs when the time interval between two subsequent re-evaluations of the load shedding scheme is sometimes larger than the time interval

Data Stream Source (Sets S_1, S_2, S_3)	**# Hits** S_1	S_2	S_3	**# Misses** S_1	S_2	S_3
Weather Information						
weather.yahoo.com/	94	94	93	6	6	7
www.wunderground.com/	95	94	95	5	6	5
www.weather.com	96	96	96	4	4	4
Stock Exchange Information						
quotes.nasdaq.com/quote.dll? page=nasdaq100	94	95	95	6	5	5
finance.indiamart. com/markets/bse/	95	95	95	5	5	5
www.channelnewsasia.com/cna/ finance/sg/stockmonitor.htm	93	94	93	7	6	7
Internet Traffic Information						
www.Internettrafficreport.com	94	95	95	6	5	5
average.miq.net/index.html	92	93	93	8	7	7
watt.nlanr.net/active/maps/ ampmap_active.php	95	94	94	5	6	6
Average	**94.3**			**5.7**		

Hit: a match of manually & automatically generated least-informative attribute(s). *Miss*: a mismatch. *Number of Randomly Chosen Sliding Windows* for each test data set: 100

Table 4.4: Experimental results of test data for load shedding scheme generation and re-evaluation approach with an average number of *hits* of *94.3%* and an average number of *misses* of *5.7%*

between changes in the ranking of the attributes of a data stream. These misses could be minimized by decreasing the time interval between two subsequent re-evaluations of the load shedding scheme, which would in turn increase the workload on the system. The decrease in the time interval can be achieved by replacing '2' in step 2(i) of Algorithm 2 (as given in Section 3.2.3.3) with 'x', where $x < 2$.

The number of calculated hits and misses show the accuracy of the load shedding scheme at any moment. According to Table 4.4, the load shedding scheme has an average accuracy, or average number of hits, of 94.3%, whereas the average number of misses is 5.7%, which achieve a relatively high accuracy. In Table 4.5, we show some of the hits and misses presented in Table 4.4.

We have also conducted experiments to verify the ability of our load shedding approach in calculating the number of attributes to be shed at each of the different

Data Stream Source	**Detected Attribute to be Shed**	**Attribute Should be Shed**
weather.yahoo.com	Sunrise	Sunset
	Sunset	Sunset
finance.indiamart. com/markets/bse	Percentage Change	Weight
	Weight	Weight
www.internettraffic-report.com/	Average Delay	Minimum Delay
	Minimum Delay	Minimum Delay

Table 4.5: Some of the *hits* and *misses* generated by the test data presented in Table 4.4

data rates, i.e., 60, 90, and 120 Kbps, with three different channel capacities, i.e., 72, 98, and 132 Kbps. According to Table 4.6[4], the number of attributes to be shed decreases with an increase in the channel capacity, and vice versa, when the data (transmission) rate remains constant. Also, we notice that no attributes are dropped when the data (transmission) rate is lower than the channel capacity, as expected.

4.4 Experimental results on recovering shed data

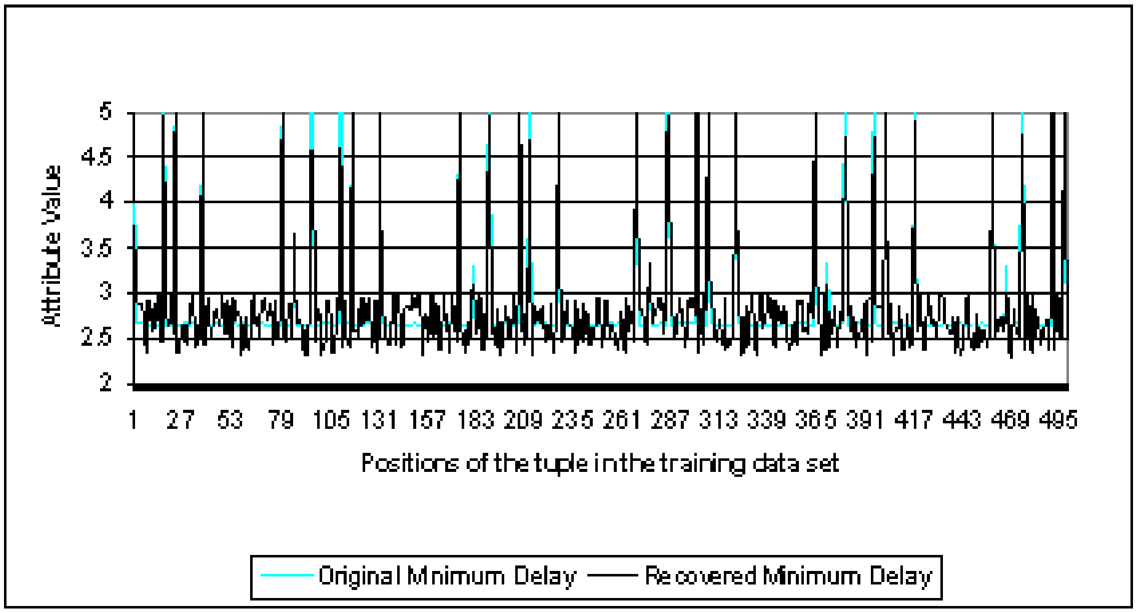
The purpose of verifying the design of our data recovery approach is to measure the accuracy in recovering shed data. Recall that in our shed data recovery approach, the value of an attribute A in a sliding window of a data stream S to be shed is stored in the synopsis recovery matrix of S whenever the difference in the value of A to be shed and a previously stored value of A in the synopsis recovery matrix exceeds the *error threshold* value of S. We have verified the correctness of our data recovery approach by (i) shedding data from test data (since data recovery is carried out in real-time on stream data), (ii) recovering the shed data values, and (iii) graphically comparing the recovered data values to the original data values. The results of the experiments on data recovery applied to the Internet traffic information data stream retrieved from

[4] The test data used in Table 4.6 are the same as the test data shown in Table 4.1, and no training data is used for this experiment, since this experiment does not require any training data.

Data Stream Source	**D**	**C**	**N**	**C**	**N**	**C**	**N**
Weather Information							
http://weather.yahoo.com/ - Set 1	60	72	0	98	0	132	0
Set 2	90	...	2	...	0	...	...
Set 3	120	...	4	...	2	...	...
http://www.wunderground.com/ - Set 1	60	...	0	...	0	...	...
Set 2	90	...	2	...	0	...	...
Set 3	120	...	3	...	2	...	...
http://www.weather.com/ - Set 1	60	...	0	...	0	...	...
Set 2	90	...	2	...	0	...	...
Set 3	120	...	3	...	2	...	...
Stock Exchange Information							
http://quotes.nasdaq.com/quote.dll?page= nasdaq100 - Set 1	60	...	0	...	0	...	...
Set 2	90	...	2	...	0	...	...
Set 3	120	...	3	...	2	...	...
http://finance.indiamart.com/markets/bse/ - Set 1	60	...	0	...	0	...	...
Set 2	90	...	2	...	0	...	...
Set 3	120	...	3	...	2	...	...
http://www.channelnewsasia.com/ cna/finance/sg/stockmonitor.htm - Set 1	60	...	0	...	0	...	...
Set 2	90	...	1	...	0	...	...
Set 3	120	...	2	...	1	...	...
Internet Traffic Information							
http://www.Internettrafficreport.com/ - Set 1	60	...	0	...	0	...	...
Set 2	90	...	2	...	0	...	...
Set 3	120	...	3	...	2	...	...
http://average.miq.net/index.html - Set 1	60	...	0	...	0	...	...
Set 2	90	...	2	...	0	...	...
Set 3	120	...	3	...	2	...	...
http://watt.nlanr.net/active/ maps/ampmapactive.php - Set 1	60	...	0	...	0	...	...
Set 2	90	...	2	...	0	...	...
Set 3	120	...	3	...	2	...	...

N : Number of Attributes to be Shed
C : Channel Capacity (Kbps) **D** : Data Rate (Kbps)

Table 4.6: Experimental results for the number of attributes to be shed with changes in *data* (*transmission*) *rate* and *channel capacity*



(a) Experimental results for the test data extracted from http://www. Internettrafficreport.com , an *Internet traffic information* data stream, on March 1, 2006 with error threshold value 13.5%, data recovery accuracy 90.17%, and error percentage 9.83%

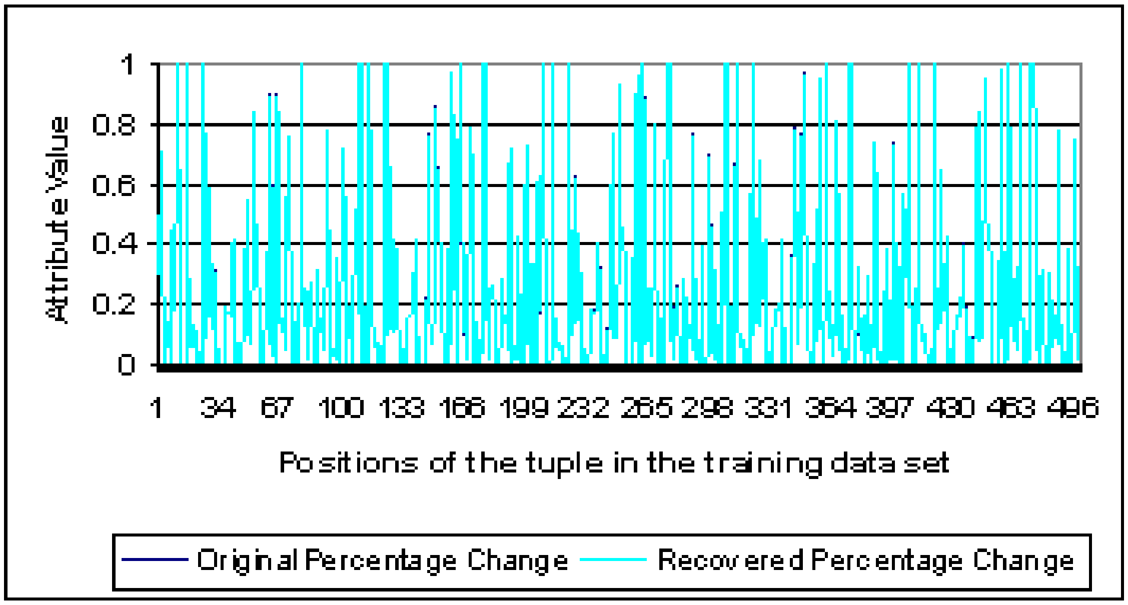
Figure 4.4: Shed data recovery for Internet traffic information data stream

Internet Traffic Report, the stock Exchange information data stream retrieved from NASDAQ, and the weather information data stream retrieved from Yahoo using 2 GB, 3 GB, and 2 GB of data, respectively, which are portions of the data in Table 4.1 as shown in Figures 4.4(a), 4.4(b), and 4.4(c), respectively. The error threshold values used in Figure 4.4(a), 4.4(b), and 4.4(c) is the *lower* of the category recovery error rate and the fixed point for the data stream, as discussed in Section 3.2.4.3. An extract of the values that were not recovered from the synopsis matrix of the three data streams are shown in Tables 4.7, 4.9, and 4.10, respectively. The shedding schemes for the three source Websites are {*Minimum Delay*}, {*Percentage Change*}, and {*Sun Set*}[5], respectively. Shown in Table 4.10 is the summary of the experiments conducted on our shed data recovery approach.

[5] Note that there is very high degree of overlapping between the *original percentage change* and the *recovered percentage change* in Figure 4.4(b), due to high data recovery accuracy.

Original Values	**Recovered Values**
3.979	4.761
2.666	2.382
2.659	3.148
2.648	3
2.663	2.97
2.655	3.123
2.65	2.757
2.663	2.871
2.648	3.002
2.662	2.9
2.649	2.386
2.641	2.402

Table 4.7: Shed data values that were not stored in the synopsis matrix for the data stream shown in Figure 4.4(a)

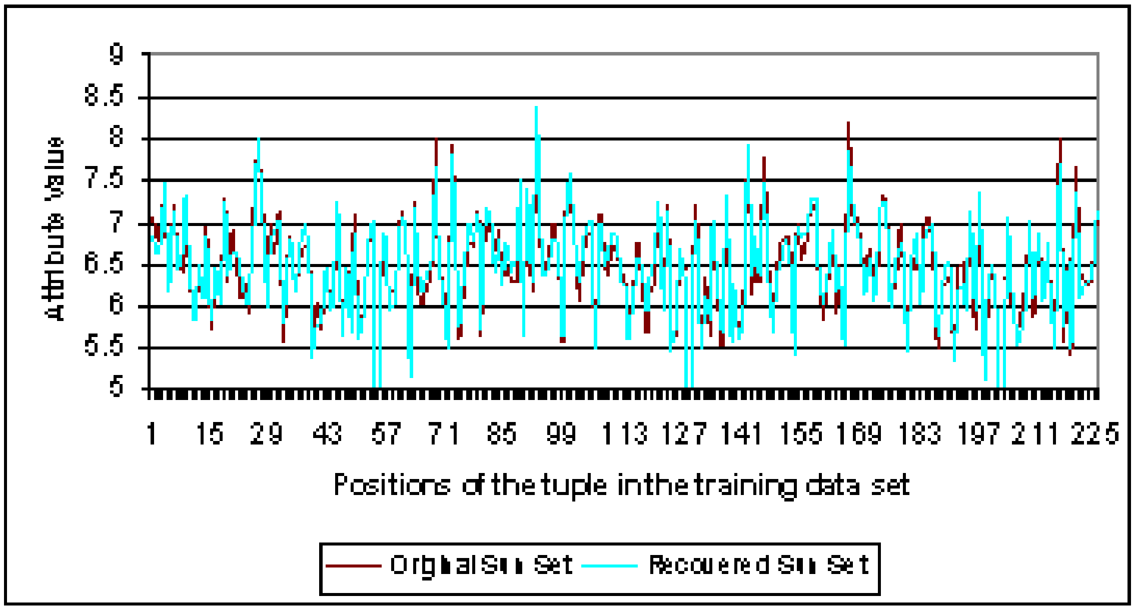


(b) Experimental results for the test data extracted from http://quotes. nasdaq.com/quote.dll?page=nasdaq100 , a *stock Exchange information* data stream, on March 1, 2006 with error threshold value 1.3%, data recovery accuracy 99.63%, and error percentage 0.37%

Figure 4.4: Shed data recovery for Stock exchange information data stream

Original Values	**Recovered Values**
0.3	0.297
0.71	0.716
-0.04	-0.04
0.02	0.02
0.14	0.141
0.04	0.04
-0.1	-0.101
0.45	0.452
0.37	0.372
0.18	0.181
1.19	1.202
-0.06	-0.061

Table 4.8: Shed data values that were not stored in the synopsis matrix for the data stream shown in Figure 4.4(b)



(c) Experimental results for test data extracted from http://weather. yahoo.com/, a *weather information* data stream, on March 1, 2006 with error threshold value 4.5%, data recovery accuracy 97.12%, and error percentage 2.88%

Figure 4.4: Shed data recovery for Weather information data stream

Original Values	**Recovered Values**
7.06	6.991
6.79	7.004
7.06	7.243
7.32	7.539
6.22	6.398
7.22	7.187
6.49	6.595
6.41	6.373
7.16	7.435
6.22	6.476
6.03	5.927
6.24	6.366

Table 4.9: Shed data values that were not stored in the synopsis matrix for the data stream shown in Figure 4.4(c)

Data Stream Source	**Data Size (GB)**	**Error Threshold Value Accuracy**	**Data recovery (%)**	**Error (%)**	**% of Shed data Stored**
http://www.internet	2	20	90.17	9.93	1.7110^{-4}
trafficreport.com	2	20	90.17	9.93	1.7110^{-4}
http://quotes.nasdaq	3	1.3	99.63	0.37	1.7310^{-4}
.com/quote.dll?page=nasdaq100	3	1.3	99.63	0.37	1.7310^{-4}
http://weather.	2	4.5	97.12	2.88	3.210^{-4}
yahoo.com/	2	4.5	97.12	2.88	3.210^{-4}
Average	**2.3**	**8.6**	**95.64**	**4.36**	2.210^{-4}

Table 4.10: Summary of the experiments conducted on the shed data recovery approach

Recall that when there is an attempt to recover a shed value that was stored in the corresponding synopsis recovery matrix, the recovered value has no error; otherwise, an error occurs, which has the error rate value less than the error threshold value, since if this is not the case, then the shed value would have been stored in the synopsis recovery matrix. Hence, the shed data recovery module would have low error value in the recovered data if the corresponding error threshold value is low. According to the experimental results as shown in Table 4.10, we claim that our shed data recovery method achieves high accuracy in recovering shed data, i.e., low error percentage in the recovered data, which is in the range of 0.37% and 9.93% with an average of 4.36%[6], at low information storage cost, i.e., percentage of the shed data to be stored in the synopsis matrix, which is in the range of $1.71\times10^{-4}\%$ and $3.2\times10^{-4}\%$ with an average of $2.2\times10^{-4}\%$. The processing times and memory usage for the synopsis matrix have been found to be insignificant.

[6] Note that the error in recovered data is the complement of recovery accuracy.

Chapter 5

Conclusions

In this thesis, we propose a dynamic load shedding and data recovery approach for solving the network channel overload problem in transmitting stream data with high data transmission rates over low capacity channels. Our load shedding and data recovery approach (i) detects and sheds least-informative attribute(s) of a data stream, which reduces the information loss by retaining more-informative attributes in the data stream, i.e., intra-stream load shedding, the first attribute-based load shedding approach in the literature, (ii) simultaneously sheds data from multiple data streams according to their data transmission rates and the channel capacity of the single channel over where multiple data streams are to be transmitted, i.e., inter-stream load shedding, and (iii) includes a unique data recovery method with low storage overhead and high accuracy in recovering shed stream data.

The uniqueness of our load shedding approach, as compared with existing load shedding approaches, is its ability in (i) detecting the least-informative attribute(s) of a data stream S to form the load shedding scheme of S, and (ii) updating the load shedding scheme in real-time according to the patterns in which rankings amongst attributes of a data stream change. The advantage of re-evaluating load shedding schemes in real-time is that it minimizes the chances of using an obsolete load shedding scheme. Furthermore, our shed data recovery approach is also unique, since it considers the category recovery error rate of a data stream, which is based on the criticality of the data in a data stream, in determining the error threshold, which is in turn used for deciding when shed data has to be stored in the synopsis matrix depending on the degree of changes in the shed attribute values.

We have conducted experiments to verify (i) the correctness of our least-informative attribute load shedding approach, with a 100% accuracy in choosing the least-informative attributes of a data stream to be shed, (ii) the correctness of our load shedding scheme generation and re-evaluation with 94.3% accuracy rate in generating and re-evaluating a load shedding scheme, and (iii) the accuracy of our shed data recovery approach with 90.2%, 99.6%, and 97.1% success rates in recovering shed data in Internet traffic information, stock exchange information, and weather information data streams, respectively, with an average shed data recovery accuracy of 95.6%.

Our load shedding approach is dynamic in nature, since it is re-evaluated in real-time, and is *adoptive*, since it selects the least-informative attributes in a data stream to be shed based on the standard deviations of various attributes of the data stream, which is applicable to any kinds of data streams. Our load shedding approach also considers the criticality of the data in a data stream and sets lower error threshold value for critical data streams, such as the medical monitoring data stream, which reduces the error in recovering shed data that require a high degree of accuracy in their stream data. Furthermore, our data recovery approach is also *adoptive*, since it stores shed data in a synopsis matrix of a data stream S whenever the change in the value to be shed and the previous value stored in the synopsis matrix is greater than the pre-defined recovery *error threshold*, irrespective of the application domain of S. In order to evaluate the correctness of the category assignment module of our load shedding and data recovery approach, we have verified the correctness in assigning a new data stream to its corresponding category, which is either '*Extremely Critical*', '*Very Critical*', '*Moderately Critical*', '*Low Critical*', '*Not Critical*', or '*Others*', with 93% accuracy, using the Fuzzy set IR model, which calculates the similarity between the lists of pre-defined attribute names and the list of attribute names for the new data stream to determine the criticality category of the new data stream.

Our shed data recovery method achieves high accuracy in recovering shed data at low information storage cost, i.e., percentage of the shed data to be stored in the synopsis matrix, which is in the range of 0.000171% and 0.00032% with an average of 0.00022%. The processing times and memory usage for the synopsis matrix have been found to be insignificant.

Bibliography

[AA03] A.K. Al-Asmari. A New Video Compression Algorithm for Different Videoconferencing Standards. *International Journal of Network Management*, 1:3–10, January 2003.

[ABC+05] Y. Ahmad, B. Berg, U. Cetintemel, M. Humphrey, J.H. Hwang, A. Jhingran, A. Maskey, O. Papaemmanouil, A. Rasin, N. Tatbul, W. Xing, Y. Xing, and S. Zdonik. Distributed Operation in the Borealis Stream Processing Engine. In *Proceedings of the 25th ACM SIGMOD Conference (SIGMOD'05)*, pages 882–884, 2005.

[ABW03] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Technical Report 2003-67, Computer Science Department, Stanford University, 2003.

[ACE+03] D.J. Abadi, D. Carney, U. Etintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A New Model and Architecture for Data Stream Management. *VLDB*, 12(2):120–139, 2003.

[BDM04] B. Babcock, M. Datar, and R. Motwani. Load Shedding for Aggregation Queries over Data Streams. In *Proceedings of the 20th Inter-*

national Conference on Data Engineering (ICDE'04), pages 350–361, 2004.

[CLL$^+$04] M. Cohen, Z.C. Liu, J. Li, Z. Wen, K. Zheng, and T. Huang. Low Bit-Rate Video Streaming for Face-to-Face Teleconference. In *Proceedings of International Conference on Multimedia and Expo*, pages 1631–1634, 2004.

[CWY05] Y. Chi, H. Wang, and P.S. Yu. Loadstar: Load Shedding in Data Stream Mining. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 1302–1305, 2005.

[Gar06] I. Garcia. Load Shedding on RSS News Feeds Data Streams. Master's thesis, Computer Science Dept., Brigham Young University, August 2006.

[Gol49] M.J.E. Golay. Notes on Digital Doding. In *Proceedings of the IRE, Vol. 37*, pages 657–664, June 1949.

[Gol04] L. Golab. Querying Sliding Windows over On-Line Data Streams. In *Proceedings of ICDE/EDBT Ph.D. Workshop*, pages 1–10, March 2004.

[Gum98] C. Gumas. *Turbo Codes Build on Classic Error-Correcting Codes and Boost Performance.* Personal Engineering & Instrumentation News (PE&IN), February 1998.

[Ham50] R.W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29:147–160, 1950.

[HIN] http://www.investopedia.com/university/movingaverage.

[HIT] http://www.its.bldrdoc.gov/projects/devglossary/datastream.html.

[HST] http://www.stockcharts.com/education/IndicatorAnalysis/indic_movingAvg.html.

[MMdEdT03] M.E. Mancin, M. Merello, L. Dipt. di Elettronica, and P. di Torino. Low-Complexity Video Compression for Wireless Networks. In *Proceedings of International Conference on Multimedia and Expo, Vol. 3*, pages 585–588, 2003.

[MQ02] J.C. De Martin and D. Quaglia. Delivery of MPEG Video Streams with Constant Perceptual Quality of Service. In *Proceedings of International Conference on Multimedia and Exhibition, Vol. 2*, pages 85–88, 2002.

[Mut03] S. Muthukrishnan. Data Streams: Algorithms and Applications. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 413–413, 2003.

[MWA$^+$03] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datarand, G.S. Manku, C. Olston, J. Rosenstein, and R. Varma. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, pages 1–16, January 2003.

[RH05] F. Reiss and J.M. Hellerstein. Data Triage: An Adaptive Architecture for Load Shedding in TelegraphCQ. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 155–156, 2005.

[RS60] I. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *J. Soc. Indus. Applied Math*, 8:300–304, 1960.

[Sha49] C.E. Shannon. *The Mathematical Theory of Information.* Urbana, University of Illinois Press, 1949. (reprinted 1998).

[TRL01] K. Tan, R. Ribier, and S. Liou. Content-Sensitive Video Streaming over Low Bitrate and Lossy Wireless Network. In *Proceedings of the Ninth ACM International Conference on Multimedia (MULTIMEDIA'01), Vol. 9*, pages 512–515, 2001.

[TSZS01] A. Ta-Shma, D. Zuckerman, and S. Safra. Extractors from Reed-Muller Codes. In 42nd *IEEE symposium on Foundations of Computer Science (FOCS'01)*, pages 638–647, 2001.

[Vit67] A.J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, IT-13:260–269, April 1967.

[WIK] http://wikipedia.org/.

[YN05] R. Yerra and Y.K. Ng. Detecting Similar HTML Documents Using a Fuzzy Set Information Retrieval Approach. In *Proceedings of the IEEE International Conference on Granular Computing (GrC'05)*, pages 693–699, July 2005.

[ZCC$^+$02] S. Zdonik, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and D. Carney. Monitoring Streams - A New Class of Data Management Applications. In *Proceedings of* *the 28th International Conference on Very Large Data Bases*, pages 215–226, 2002.

Appendix A

An example on the use of Hamming code

Consider the 7-bit data word "0110101". We demonstrate using Table A.1 how the Hamming code of the 7-bit data word "0110101" is calculated and used to detect an error, where $\boldsymbol{D}_i$ $(1 \leq \quad i \quad \leq 7)$ denotes a data bit and $\boldsymbol{P}_j$ $(1 \leq \quad j \quad \leq 4)$ denotes a parity bit in Table A.1.

First of all, all the data bits of the data word "0110101" are inserted into their corresponding positions in the bit string and the parity bits are calculated in each case using *even* parity.

According to the steps in the Hamming code formation, P_1 considers all the data bits at positions having the 1^{st} bit (in their binary representation) being a '1', which are D_1, D_2, D_4, D_5, and D_7 in this example. Amongst these data bits in the data word, three of them, i.e., D_2, D_5, and $D_{7,}$ are set as '1', thus the parity bit P_1 ofD_1, D_2, D_4, D_5, and D_7 is set as '1' to ensure even parity. P_2 considers all the data bits at positions having the 2^{nd} bit (in their binary representation) being a '1', which are

Parity/Data	**P$_1$**	**P$_2$**	**D$_1$**	**P$_3$**	**D$_2$**	**D$_3$**	**D$_4$**	**P$_4$**	**D$_5$**	**D$_6$**	**D$_7$**
Bit Position	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Bit Position in Binary	**1**	**10**	**11**	**100**	**101**	**110**	**111**	**1000**	**1001**	**1010**	**1011**
Data Word without Parity:			**0**		**1**	**1**	**0**		**1**	**0**	**1**
P$_1$	**1**		0		1		0		1		1
P$_2$		**0**	0			1	0			0	1
P$_3$				**0**	1	1	0				
P$_4$								**0**	1	0	1
Data Word with Parity:	**1**	**0**	0	**0**	1	1	0	**0**	1	0	1

Table A.1: Calculation of Hamming code parity bits for the data word “0110101”

D_1, D_3, D_4, D_6, and D_7. Amongst these data bits in the data word, two of them, i.e., D_3, and D_7, are set as ‘1’, thus the parity bit P_2 of D_1, D_3, D_4, D_6, and D_7 is set to 0. P_3 considers all the data bits at positions having the 3^{rd} bit (in their binary representation) being a ‘1’, which are D_2, D_3, and D_4. Amongst these bits in the data word, two of them, i.e. D_2, and D_3, are set as ‘1’, thus the parity bit P_3 of D_2, D_3, and D_4 is unset, i.e., set to 0. P_4 considers all the data bits at positions having the 4^{th} bit (in their binary representation) being a ‘1’, which are D_5, D_6, and D_7. Amongst these bits in the data word, two of them, i.e., D_5, and D_7, are set as ‘1’, thus the parity bit P_4 of D_5, D_6, and D_7 is unset.

The new data word (with the inserted parity bits) is now ”10001100101”. We assume the last bit is corrupted during transmission and is changed from ‘1’ to ‘0’, which yields the new (flawed) data word ”10001100100”, as shown in Table A.2, which demonstrates the checking of parity bits to detect and correct errors.

According to the Hamming code formation, the algorithm checks the data bits arrived at the receiver site at positions having the 1^{st} bit (in their binary representation) being set, which are D_1, D_2, D_4, D_5, and D_7 in this example. Amongst these data bits in the flawed data word, two of them, i.e., D_2 and D_5 are set as '1', and thus the calculated parity bit ofD_1, D_2, D_4, D_5, and D_7 is set as '0' to ensure even parity. Since this calculated parity bit does not match the value of parity bit P_1 in the new flawed data word, which is a '1', the parity check is set to *fail*, and the corresponding parity bit check flag is set to '1'. Next, the algorithm checks the data bits at positions having the 2^{nd} bit (in their binary representation) being set, which are D_1, D_3, D_4, D_6, and D_7. Amongst these data bits in the flawed data word, only one bit, i.e., D_3 is set as '1', and thus the calculated parity bit P_2 is set to '1'. Since this calculated parity bit does not match the parity bit P_2 in the new flawed data word, which is a '0', the parity check is set to *fail*, and the corresponding parity bit check flag is set to '1'. Hereafter, the algorithm checks the data bits at positions having the 3^{rd} bit (in their binary representation) being set, which are D_2, D_3, and D_4. Amongst these data bits in the flawed data word, two of them, i.e. D_2, and D_3, are set as '1', and thus the calculated parity bit P_3 is unset, i.e., set to 0. Since this calculated parity bit matches the parity bit P_3 in the new flawed data word, which is a '0', the parity check is set to *pass*, and the corresponding parity bit check flag is set to 0. Last, the algorithm checks the data bits at positions having the 4^{th} bit (in their binary representation) being set, which are D_5, D_6, and D_7. Amongst these data bits in the flawed data word, only one bit, i.e., D_5, is set as '1', and thus the calculated parity bit P_4 is set to '1'. Since this calculated parity bit does not match the parity bit P_4 in the new flawed data word, which is a '0', the parity check is set to *fail*, and the corresponding parity bit check flag is set to '1'.

The parity bit check flag (read in the reverse order) is 1011 for the flawed data word (with parity bits), which is "10001100100". The decimal value of the parity bit check flag is 11, which signifies that the 11^{th} bit, which is D_7, in the flawed data word

Parity/ Data Bit	**P$_1$**	**P$_2$**	**D$_1$**	**P$_3$**	**D$_2$**	**D$_3$**	**D$_4$**	**P$_4$**	**D$_5$**	**D$_6$**	**D$_7$**	**Calculated Parity Bit Based on Transmitted Data Bits**	**Original Parity Bits**	**Parity Check**	**F**
Received Data Word:	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**				
P$_1$	**1**		0		1		0		1		0	**0**	**1**	**Fail**	**1**
P$_2$		**0**	0			1	0			0	0	**1**	**0**	**Fail**	**1**
P$_3$				**0**	1	1	0					**0**	**0**	**Pass**	**0**
P$_4$								**0**	1	0	0	**1**	**0**	**Fail**	**1**

F: Parity Bit Check Flag

Table A.2: Checking of parity bits for the flawed data word "0110101"

(including the parity bits and the error bit) is incorrect and must be flipped.

Flipping the 11th bit, which is '0', in the flawed data word (with parity bits), which is "1000110010**0**", causes the binary representation of the flawed word to change back into "1000110010**1**", which is the unflawed data word with parity bits, and removing the parity bits from the unflawed data word "1000110010**1**" yields the original data word "0110101".

Consider the 7-bit data word "0110101" again and assume that the second bit, which is the parity bit P$_2$, is corrupted during transmission and is changed from '0' to '1', which yields the new flawed data word "11001100101", as shown in Table A.2, which demonstrates yet another checking of parity bits to detect and correct errors.

According to the Hamming code formation, the algorithm checks the data bits at positions having the 1st bit (in their binary representation) being set, which are D_1, D_2, D_4, D_5, and D_7. Amongst these data bits in the flawed data word, three of them, i.e., D_2, D_5, and D_7, are set as '1', and thus the calculated parity bit ofD_1, D_2, D_4,

D_5, and D_7 is set as '1' to ensure even parity. Since this calculated parity bit matches the value of parity bit P_1 in the new flawed data word, which is also a '1', the parity check is set to *pass*, and the corresponding parity bit check flag is set to '0'. The algorithm next checks the data bits at positions having the 2^{nd} bit (in their binary representation) being set, which are D_1, D_3, D_4, D_6, and D_7. Amongst these data bits in the flawed data word, two of them, i.e., D_3, and D_7 are set as '1', and thus the calculated parity bit P_2 is set to '0'. Since this calculated parity bit does not match the parity bit P_2 in the new flawed data word, which is a '1', the parity check is set to *fail*, and the corresponding parity bit check flag is set to '1'. Hereafter, the algorithm checks the data bits at positions having the 3^{rd} bit (in their binary representation) being set, which are D_2, D_3, and D_4. Amongst these data bits in the flawed data word, two of them, i.e. D_2, and D_3, are set as '1', and thus the calculated parity bit P_3 is unset, i.e., set to 0. Since this calculated parity bit matches the parity bit P_3 in the new (flawed) data word, which is a '0', the parity check is set to *pass*, and the corresponding parity bit check flag is set to 0. As the last step, the algorithmchecks the data bits at positions having the 4^{th} bit (in their binary representation) being set, which are D_5, D_6, and D_7. Amongst these data bits in the flawed data word, two of them, i.e., D_5, and D_7 are set as '1', and thus the calculated parity bit P_4 is set to '0'. Since this calculated parity bit matches the parity bit P_4 in the new flawed data word, the parity check is set to *pass*, and the corresponding parity bit check flag is set to '0'.

As the parity bit check flag (read in the reverse order) is 0010 for the flawed data word (with parity bits), which is "11001100101", the decimal value of the parity bit check flag is 2, which signifies that the 2^{nd} bit in the flawed data word (including the parity bits and the error bit), that turns out to be the parity bit P_2 is incorrect and

Parity/ Data Bit	**P$_1$**	**P$_2$**	**D$_1$**	**P$_3$**	**D$_2$**	**D$_3$**	**D$_4$**	**P$_4$**	**D$_5$**	**D$_6$**	**D$_7$**	**Calculated Parity Bit Based on Transmitted Data Bits**	**Original Parity Bits**	**Parity Check**	**F**
Received Data Word:	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**				
P$_1$	**1**		0		1		0		1		1	**1**	**1**	**Pass**	**0**
P$_2$		**1**	0			1	0			0	1	**0**	**1**	**Fail**	**1**
P$_3$				**0**	1	1	0					**0**	**0**	**Pass**	**0**
P$_4$								**0**	1	0	1	**0**	**0**	**Pass**	**0**

F: Parity Bit Check Flag

Table A.3: Checking of parity bits for the flawed data word "0110101"

must be flipped.

Flipping the 2nd bit, which is '1', in the flawed data word (with parity bits), which is "1**1**001100101," causes the binary representation of the flawed word to change back into "1**0**001100101" (which is the unflawed data word with parity bits), and removing the parity bits from the unflawed data word "1**0**001100101" yields the original data word "0110101".

Appendix B

An example on movings averages

We consider arbitrary stock data to demonstrate the usage of the two most popular types of MA, i.e., SMA and EMA, to tone down the fluctuations. A SMA for the most recent values in a period is formed by computing the average (mean) of the data over a specified number of periods. For example, a 10-day simple moving average for a particular stock is calculated by adding the closing prices for the last 10 days and dividing the total by 10. To explain the concept even better, we quote examples hosted by [HST].

Table B.1 shows the closing prices for stocks of East man Kodak (EK), which are used for calculating a 10-day SMA, for which Day 10 is the first day possible to calculate a SMA. As the closing price for a new day is added, it is included in the SMA calculation and the closing price for the oldest day is excluded. For example, the 10-day SMA for Day 11 is calculated by adding the closing prices of Day 2 through Day 11 and dividing the sum by 10. The averaging process is then moved on to the next day where the 10-day SMA for Day 12 is calculated by adding the prices of Day 3 through Day 12 and dividing the sum by 10.

	SMA	EMA Period (N):		10
Period	**Date**		**Close (C)**	**10-Day EMA (X)**
1	09-Nov-99		$ 64.75	
2	10-Nov-99		$ 63.79	
3	11-Nov-99		$ 63.73	
4	12-Nov-99		$ 63.73	
5	15-Nov-99		$ 63.55	
6	16-Nov-99		$ 63.19	
7	17-Nov-99		$ 63.91	
8	18-Nov-99		$ 63.85	
9	19-Nov-99		$ 62.95	
10	22-Nov-99		$ 63.37	$ 63.682
11	23-Nov-99		$ 61.33	$ 63.340
12	24-Nov-99		$ 61.51	$ 63.112
13	26-Nov-99		$ 61.87	$ 62.926
14	29-Nov-99		$ 60.25	$ 62.578
15	30-Nov-99		$ 59.35	$ 62.158
16	01-Dec-99		$ 59.95	$ 61.834
17	02-Dec-99		$ 58.93	$ 61.336
18	03-Dec-99		$ 57.68	$ 60.719
19	06-Dec-99		$ 58.82	$ 60.306
20	07-Dec-99		$ 58.87	$ 59.856

Table B.1: The sample closing prices for stocks of East man Kodak (EK) for 20 days

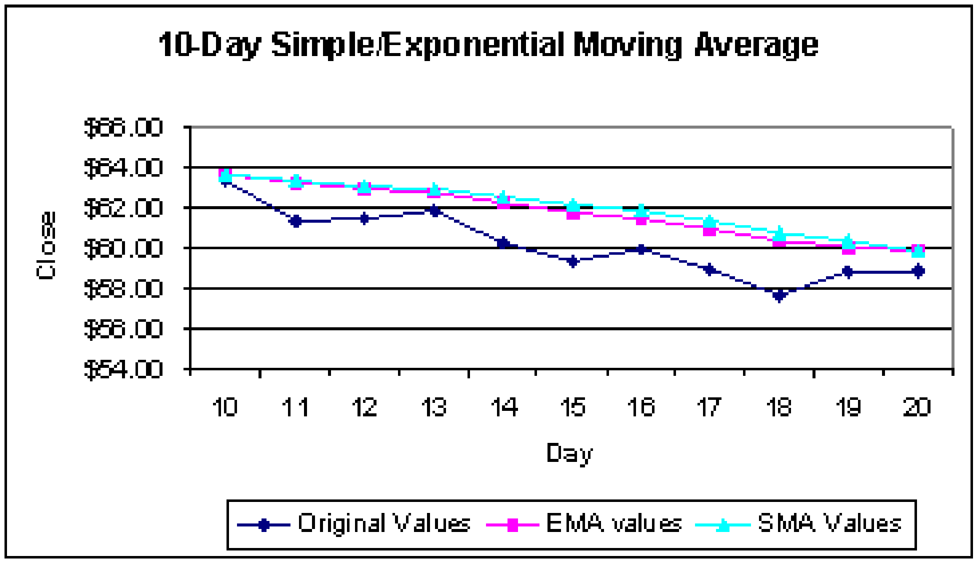


Figure B.1: SMAs and EMAs for the data from Table B.1 and Table B.2

The chart in Figure B.1 shows the SMAs (along with the EMAs) for the data from Table B.1. The SMAs are calculated starting on Day 10 and continue from there as explained before. The simple illustration highlights the fact that all moving averages are lagging indicators and will always be "behind" the closing price, i.e., stay above the original value when the value curve is curving down (i.e., when the values are decreasing) and stay below the original value when the value curve is curving up (i.e., when the values are increasing).

Clearly, SMA lacks the ability of controlling the lag according to the user's need. This problem can be overcome by using EMA, which is also called *Exponential Weighted Moving Averages* (EWMAs)[1] [HIN]. EMA reduces the lag by applying more weight to recent values relative to older values[2]. Since EMA puts more weight on recent values, which allows the EMA to react quicker to recent changes in values than the SMA.

Table B.2 includes the EMAs for the same set of closing prices of East man Kodak at the end of different days as shown in Table B.1. For the EMA of the first period, the SMA over the first period was used as the EMA, which is shown as darkened in Table B.2. From the 2^{nd} period onwards, to calculate the EMA of n^{th} period, i.e., EMA(a_n), n $\geq$ 1, the EMA of the previous period, i.e., EMA(a_{n-1}) was used. Assume that C in Table B.2 denotes the current price, P denotes the EMA at the close of the previous day, and K, which is the multiplier used to control the impact of older data points, is 0.18 (refer to Equation 2 in chapter 3). Using Equation 2 (in chapter 3), the 11^{th} period is calculated as:

(61.33 - 63.68) x 0.18 + 63.68 = 63.25

[1] http://www.stockcharts.com/education/IndicatorAnalysis/indic_movingAvg.html

[2] The *recent* (*older*, respectively) value of the closing price is the value which has not been (has been, respectively) used to calculate the EMA.

		EMA Period (N):	10		
		Smoothing Constant (K):	0.181818		
Period	**Date**	**Close (C)**	**Previous Period's EMA (P)**	**10-Day EMA (X)**	
1	09-Nov-99	$ 64.75			
2	10-Nov-99	$ 63.79			
3	11-Nov-99	$ 63.73			
4	12-Nov-99	$ 63.73			
5	15-Nov-99	$ 63.55			
6	16-Nov-99	$ 63.19			
7	17-Nov-99	$ 63.91			
8	18-Nov-99	$ 63.85			
9	19-Nov-99	$ 62.95			
10	22-Nov-99	$ 63.37		$ 63.682	- *
11	23-Nov-99	$ 61.33	$ 63.682	$ 63.254	
12	24-Nov-99	$ 61.51	$ 63.254	$ 62.937	
13	26-Nov-99	$ 61.87	$ 62.937	$ 62.743	
14	29-Nov-99	$ 60.25	$ 62.743	$ 62.290	
15	30-Nov-99	$ 59.35	$ 62.290	$ 61.755	
16	01-Dec-99	$ 59.95	$ 61.755	$ 61.427	
17	02-Dec-99	$ 58.93	$ 61.427	$ 60.973	
18	03-Dec-99	$ 57.68	$ 60.973	$ 60.374	
19	06-Dec-99	$ 58.82	$ 60.374	$ 60.092	
20	07-Dec-99	$ 58.87	$ 60.092	$ 59.870	

Table B.2: EMAs for the closing prices of East man Kodak at the end of different days

The chart in Figure B.1 also shows the EMAs (along with SMAs) for the data from Table B.2, which shows that the EMA curve is always closer to the East man Kodak curve than the SMA curve.

Appendix C

Data sets used for EQ Threshold values detection and verification

The data set used to detect the EQ threshold values used in this thesis is shown in Table C.1.

Data Stream Source (i)	**List of Attribute Names**	**Sim (i,j)**	**Sim (j,i)**	**EQ**	**Detected Category**
	Internet Traffic Information (j)				
http://www.internet trafficreport.com	Router Name, Current Index, Response Time, Packet Loss, Minimum Delay, Average Delay, Maximum Delay	0.5	1	1	Internet Traffic Information
http://average.miq. net/index.html	Router, Time, Response Packet Loss, Minimum Delay, Average Delay, Maximum Delay,	0.63	1	1	Internet Traffic Information
http://watt.nlanr.net /active/maps/amp mapi_active.php	Site Name, Min, Mean, Max, StdDev,	0.166	0.2	0	Others (False Negative)

Continued on Next Page...

Table C.1 – Continued

	Loss				
http://www.jisctau.ac.uk/usa-access.html	Router Name, Average delay, Max Delay, Min delay, Loss	0.666	0.8	1	Internet Traffic Information
http://www.ratings .matrix.net/	Router, Packet Loss, Min Loss, Max Loss, Average Loss	0.777	0.8	1	Internet Traffic Information
http://www.cns.ucla .edu/traffic.html	Router Name, Current Index, Response Time, Packet Loss, Minimum Delay, Average Delay, Maximum Delay	0.5	1	1	Internet Traffic Information
http://www1.world com.com/global/ about/network/latency/	Router Name, Latency, Packet Loss	0.6	0.6	1	Internet Traffic Information
http://stats.dante.org.uk/nep/routermap.html	Router Name, Average Delay, Max Delay, Min delay, Loss	0.666	0.8	1	Internet Traffic Information
Weather Information (j)					
http://www.metservice.co.nz/default/ index.php ?alias=Auckland	Location, Temperature, Humidity, Wind Direction, Wind Speed, Pressure, Rainfall	0.55	0.8	1	Weather Information
http://wwwa.accuweather.com/worldforecastcurrent conditions.asp?partner=a ccuweather&myadc=0&tr aveler=0&zipcode=SAM—AR—AR002—SAN%20FRA NCISCO—&metric=1	Location, Temperature, Humidity, Visibilty Ceiling, DewPoint Apparent Temperature Wind Chill, Wind Direction, Wind Speed, Pressure, Wind Gusts	0.47	0.8	1	Weather Information
http://wwwagwx.ca.uky.edu/	Location, Air Temperature, Relative Humidity, Dewpoint, Wind Direction, Barometer Pressure	0.5	1	1	Weather Information
http://weather.msn.com/lo cal.aspx?wealocations=w c:USAZ0105	Location, Temperature, Humidity, Visibility, Dewpoint, Wind,	0.45	1	1	Weather Information

Continued on Next Page...

Table C.1 – Continued

	UV Index, Sunrise, Sunset, Barometer				
http://autofeed.msn.co.in/weather/details/Delhi.asp	Location, Temperature, Wind, Relative Humidity, Barometer, Sunrise, Sunset	0.625	1	1	Weather Information
http://www.bbc.co.uk/weather/5day.shtml?world=0151	Location, Temperature, Wind, Relative Humidity, Pressure, Visibility	0.57	0.8	1	Weather Information
http://www.findlocalweather.com/forecast.php?icao #NAME?	Location, Humidity, Wind Speed, Barometer, Dewpoint, Heat Index, Wind Chill, Civil Twilight, Sunrise, Sunset, Civil Twilight	0.312	0.8	1	Weather Information
http://www.news8austin.com/content/Weather/weather%5Fstations/Station1/	Location, High Today, Low Today, Humidity, Wind Speed, Barometer, DewPoint Heat Index, Gust, Daily Rain, Weekly Rain, Monthly Rain	0.25	0.8	0	Weather Information (False Negative)
http://weather.yahoo.com/	Location, Temperature Humidity, Wind, Visibility Barometer, DewPoint Heat Index, Sun Rise, Sun Set	0.45	1	1	Weather Information
http://www.wunderground.com	Location, Temperature Humidity, Wind, Visibility Pressure, DewPoint Precipitation	0.57	0.8	1	Weather Information
http://www.weather.com	Location, UV Index Humidity, Wind, Visibility Pressure, DewPoint	0.375	0.6	1	Weather Information
http://weather.boston.com	Location, Temperature Humidity, Wind Direction, Visibility Precipitation, Barometer Wind Speed Tanning Index	0.54	1	1	Weather Information
Stock Exchange Information (j)					
http://quotes.nasdaq.com/quote.dll?	Company Name, Last Sale, Net Change	0.5	0.8	1	Stock Exchange

Continued on Next Page...

Table C.1 – Continued

	Share Volume, Nasdaq 100 Index, Percentage Change				Information
http://finance.indiamart.com/markets/bse/	Company Name, Last Price, Change, Percentage Change, Market Cap, Weight	0.44	0.6	1	Stock Exchange Information
http://www.channelnewsasia.com/cna/finance/	Stock, Buy, Sell, Last Done, Volume	0.33	0.2	1	Stock Exchange Information
http://www.ihs.com/Investor-Relations/stock-information.htm	Company Name, Last Price, Change, Percentage Change, Market Cap, Weight	0.44	0.6	1	Stock Exchange Information
http://www.stockwatch.com/swnet/utilit/utilit_snapsh_result.aspx?action=go&symbol=AM&regionn=C&lookup=symbol&snapshot=default	Symbol, Company Name, Price, Last Change, Volume, Open High, Year High, Low, Percentage Change	0.66	0.6	1	Stock Exchange Information
http://www.newsx.com.au/prices_alpha.asp?nsxcode=APO	Symbol, Company Name, Last Price, Offer, Volume, Open, High, Low, Percentage Change	0.55	1	1	Stock Exchange Information
http://www.tsx.com/HttpController?GetPage=QuotesViewPage&DetailedView=DetailedPrices&Market=T&Language=en&QuoteSymbol_1=am&x=29&y=1	Symbol, Price, Change, Percentage Change, Volume, Exchange	0.57	0.6	1	Stock Exchange Information
http://phx.corporateir.net/phoenix.zhtml?c=84204&p=irol-stockquotechart	Company Name, Price, Volume, Intraday High, Intraday Low, Todays Open, Previous Close, Change, Percentage Change, 52 Week High, 52 Week Low, Currency, Exchange	0.388	1	1	Stock Exchange Information
http://www.google.com/help/features.html#stock	Company Name, Open, High, Low, Volume, Average Volume, Market Cap	0.5	0.6	1	Stock Exchange Information
http://moneycentral.msn.com/detail/stock_quote?Symbol=MSFT	Company Name, Average Daily Volume, High, Low, Previous Close, 52 Week High, 52 Week Low, Market Cap	0.3	0.6	1	Stock Exchange Information
Others (j)					
Financial Information					
http://www.deloitte.com/d	Revenues, Year,	0	0	0	Others

Continued on Next Page...

Table C.1 – Continued

tt/leadership/0,1045,sid%253D2282,00.html	Tax, US Offices, CPA, Others				
https://online.wellsfargo.com/mn1_ea4_on/cgibin/session.cgi?sessargs=H_sGNvbD35koY_7IUSgvMni9RZ5kVJqN	Date, Description, Amount, Balance	0	0	0	Others
https://securekeypointcu.com/Common/Register/Register.asp?Account=1&Pending=1	Date, Num, Description, Amount, Clear, Balance Action	0	0	0	Others
Network Data Loss Information					
http://members.toast.net/leslie/testpage/testinfo.htm	Line Rate, Compression Off, Compression On	0	0	0	Others
http://members.toast.net/leslie/testpage/testinfo.htm	Line Rate, Transfer Rate, Estimated download, Typical download	0	0	0	Others
http://www.cisco.com/en/US/tech/tk39/tk51/packtdrops.shtml	Throughput, Transfer Rate, Bit Rate, Drops	0	0	0	Others
Chemical Properties Information					
http://chemlab.pc.maricopa.edu/PERIODIC/Pb.html	Atomic number, Atomic weight, Melting point, Bonding radius, Boiling point, Atomic radius, Heat of vaporization, Specific heat	0	0	0	Others
http://en.wikipedia.org/wiki/Table_of_chemical_elements	Name, Symbol, Discoverer, Period Group, Mass, Density, Melting, Boiling, Year of discovery	0	0	0	Others
http://ecophys.biology.utah.edu/Labfolks/domingues/	Leaf temperature, Relative Humidity, CO2 Concentration, Staurating level	0.28	0.4	0	Others
http://72.14.207.104/search?q=cache:LeSc6K0sZ0kJ:www.freepatentsonline.com/4956582.html+temperature+%22conditions+inside+chamber%22+monitoring&hl=en&gl=us&ct=clnk&cd=4	Temperature, Pressure, O2 Sub Flow Rate	0.2	0.2	0	Others

Table C.1: Test data used for detecting the permissible and variation threshold values detected in Figure 4.2(a) and 4.2(b)

The data set used to verify the EQ threshold values used in this thesis is shown in Table C.2.

Data Stream Source (i)	**List of Attribute Names**	**Sim (i,j) (j)**	**Sim (j,i)**	**EQ**	**Detected Category**
	Internet Traffic Information (j)				
http://www.cyberlynk.net/ customercare_techsupport /traffic/	Max in, Average in, Current in, Max out, Average out, Current out	0.33	0.2	0	Others (False Negative)
http://watt.nlanr.net /active/amp-apantyo/ international/body.html	Site Name, Min, Mean, Max, StdDev, Loss	0.166	0.2	0	Others (False Negative)
http://www.nlanr.net/ Presentations/ Montreal.jun96/	Router, Packet Loss, Min Loss, Max Loss, Average Loss	0.66	0.8	1	Internet Traffic information
http://plumeria.vmth. ucdavis.edu/ java/ cgi-bin/netstat.cgi	Router, Delay, Loss, Latency	0.75	0.6	1	Internet Traffic information
http://ipnetwork.bgtmo. ip.att.net/pws/current _network_performance.shtml	Destination, Latency, Packet Loss	0.5	0.4	1 1	Internet Traffic information
http://traffic. cwix.com/	Name, Average Delay, Max Delay, Min delay, Loss	0.625	0.6	1	Internet Traffic information
	Weather Information (j)				
http://www.bbc.co uk/weather/5day .shtml?world=0100	Location, Max Day, Min Night, Wind, Visibility, Pressure, Relative Humidity, Sun Index, Pollution	0.3	0.6	1	Weather information
http://asp.usatoday.com/ weather/CityForecast. t.aspx?LocationID=USCA 0993&ps=L1	Location, Temperature, Sunrise, Sunset, Heat Index, Humidity, Wind, Visibility, Dewpoint, Barometer	0.45	1	1	Weather information
http://www.theweather network.com/weath er/cities/can/Pages/ CAAB0049.htm	Location, Temperature, Wind Gusts, Relative Humidity, Dewpoint, Pressure, Visibility, Ceiling	0.4	0.8	1	Weather information
http://www.washington post.com/ac2/wpdyn/ weather?local=true& zipcode=95111&go.x=0	Condition, Temperature, Wind, Barometer, Humidity, Sunrise, Sunset	0.57	0.8	1	Weather information
http://www.weather online.co.uk/North	Location, Temperature, Humidity, Visibility,	0.8	0.44	1	Weather information

Continued on Next Page…

Table C.2 – Continued

America.htm	Precipitation, Clouds, Wind, Pressure, Snow				
http://www.weather reports.com/United_ States/CA/ Los_Angeles	Location, Temperature, Wind, Direction, Pressure, Dewpoint, Visibility, Feelslike	0.37	0.6	1	Weather information
http://www.chicago tribune.com/news/ weather/	Location, Temperature, Windchill, Wind, Dewpoint, Humidity, Visibility, Barometer	0.62	1	1	Weather information
http://weather.cbs2. com/auto/kcbsV2/CA/ Sun_Valley.html	Location, Wind, Humidity, Dewpoint, Visibility, Barometer	0.66	0.8	1	Weather information
http://www.thunder headtech.com/Taunton2W/ Current.html	Location, Temperature, Dewpoint, Humidity, Wind Gusts, Wind, SunShine, Precipitation, Barometer, Heat Index	0.45	1	1	Weather information
http://weather.philly .com/auto/philly/PA/ Philadelphia.html?map =Temperature	Location, Temperature, Wind, Dewpoint, Pressure, humidity, Visibility	0.57	0.8	1	Weather information
http://weather.wcco.com/ US/MN/Minneapolis.html US/MN/Minneapolis.html	Location, Wind, Humidity, Dewpoint, Visibility, Barometer	0.66	0.8	1	Weather information
http://weather.calendar live.com/US/CA/ KBUR.html	Location, Temperature, Wind, Dewpoint, humidity Visibility	0.66	0.8	1	Weather information
Stock Exchange Information (j)					
http://www.nyse.com/about/ listed/lc_C.html?Listed Comp=All&start=121& startlist=1&itemNAME?	Name, Symbol, Last Trade, Volume, Change	0.4	0.4	1	Stock exchange information
http://www.amex.com/	Product, Symbol, Last Sale, Volume	0.833	0.8	1	Stock exchange information
http://www.londonstock exchange.com/en- GB /pricesnews/prices/ system/detailed prices.htm?ti=ANTO	Name, Price, High, Low, Last Close, Volume, Offer	0.75	1	1	Stock exchange information
http://www.asx.com.au /asx/research/CompanyInfo SearchResults.jsp?search By=asxCode&allinfo=on& asxCode=ANZ&company Name=&principalActivity =&industryGroup=NO	Code, Last Change, Offer, Open, High, Low, Volume	0.5	0.4	1	Stock exchange information
http://www.tsx.com/Http Controller?GetPage= ListedCompaniesView	Company Name, Last Value, Net Change, Percentage Change	0.375	0.6	1	Stock exchange information

Continued on Next Page…

Table C.2 – Continued

Page&Page=1&Search IsMarket=Yes&Market= T&Language=en&Search Criteria=Name&Search Type=Contain&Search Keyword=cisco+systems &SUBMIT=Submit					
http://www.swx.com/ market/quote_chart _en.html?id=CH 0012138605CHF1	Company Name, Last Trade, Change, Market, Previous Close, Open	0.6	0.6	1	Stock exchange information
http://www.set.or.th /set/stockquotation. do?languageNAME?	Company Name, Last Change, Percentage Change, High, Low, Volume, Value	0.75	1	1	Stock exchange information
http://www.jamstockex .com/controller.php? action=view_summary	Company Name, Volume, Last Sale, Change	0.295	0.4	1	Stock exchange information
http://www.kse.com.pk/	Company Name, Symbol, Volume, High Rate, Price Change, Shares				Stock exchange information
http://www.nyse.com/ home.html	Symbol, Volume, Value, Change	0.6	0.6	1	Stock exchange information
http://esite.sgx.com/ live/st/STTop20. asp?top=Gain&cnt=20	Company Name, Last, Change, Percentage, Volume, Buy, Sell, Volume, High, Low	0.333	0.4	1	Stock exchange information
http://www.stockhouse. ca/quote/index.asp	Company Name, Last Change, Percentage Change, High, Low, Open	0.255	0.2	0	Others (False Negative)
Others (j)					
Financial Information					
https://cards.chase. com/Account/Account Activity.aspx?AI=1	Trans Date, Post Date, Type, Description, Transaction Number, Amount	0	0	0	Others
https://www99.american express.com/myca/ estatement/us/action?	Date, Description, Amount	0	0	0	Others
https://www4.usbank.com/ internetBanking/ RequestRouter?reques tCmdId=AccountDetails &ACCOUNTLISTITEM =IB7rtb8XZhcCl7f 7n5JKRQmD%2FKUy rWm%2BtxExx%2 BivOALj6P5f2lz UcJNIHHa1i3Kg kV8%2FzHVFhIlRdY zEPwgtIwwOJ%2BLb0	Date, Description, Credit, Charge	0	1	0	Others

Continued on Next Page...

Table C.2 – Continued

8WYtd6XYEDyncrrxH h%2BDKW36%2FtSJw 6mMQ%2Bk6%2B k8DZgUncD1% 2BiprIFaUjQ%3D%3D					
Network Data Loss Information					
http://www.nlanr. net/Viz/End2end/	Source, Min, Avg, Max, Throughput, Destination, Timestamp	0	0	0	Others
http://www.cisco.com/en/US /products/hw/routers/ps3 41/products_data_ sheet09186a00801.html	Source, Packet drop, Througphut, Destination	0	0	0	Others
Chemical Properties Information					
http://www.chemical elements.com/ elements/fe.html	Name, Symbol, Atomic Number, Mass, Melting Point, Boiling Point, Density, Number Protons, Electrons	0	0	0	Others
http://www.lenntech. com/Periodic-chart-elements/H-en.htm	Atomic Number, Atomic Mass, Electrnegativity, Density, Melting Point, Boiling Point, Vanderwalls Radius, Ionic Radius	0	0	0	Others
http://www.ieer.org/ fctsheet/pu-props.html	Color, Melting Point, Boiling Point, Density	0	0	0	Others
http://www.chemistry explained.com/ elements/C-K/Indium.html	Symbol, Atomic Number, Atomic Mass, Family, Pronumciation	0	0	0	Others
http://www.chemsoc.org/ visElements/pages/data/ intro_groupii_data.html	Atomic Number, Relative Atomic Mass, Melting Point, Density	0	0	0	Others

Table C.2: Test data used for verifying the permissible and variation threshold values detected in Figure 4.2(a) and 4.2(b)