2005

Hippocratic Data Streams-Concepts, Architectures and Issues

M. H. Ali

M. Y. ElTabakh

Elisa Bertino
Purdue University, bertino@cs.purdue.edu

Report Number:
05-025

HIPPOCRATIC DATA STREAMS - CONCEPTS, ARCHITECTURES AND ISSUES

Mohamed Ali
Mohamed ElTabakh
Elisa Bertino

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #05-025
November 2005

Hippocratic Data Streams - Concepts, Architectures and Issues

M. H. Ali M. Y. ElTabakh E. Bertino

Department of Computer Science, Purdue University
{mhali, meltabak, bertino}@cs.purdue.edu

Abstract

The goal of an Hippocratic DBMS *is to preserve the privacy of data without much sacrificing performance. In this paper, we address problem of developing privacy-preserving systems into the more challenging context of data streams. We contrast data streams to traditional databases and identify the new challenges posed by data streams. We discuss examples of streaming applications in which privacy and security issues are crucial. We propose a visionary architectural design of how a Hippocratic Data Stream Management System (HDSMS) may look like. We identify several open research directions where privacy preserving issues meet the data streaming paradigm.*

1. Introduction

Recent advances in pervasive computing and sensor networks combined with techniques for stream data processing [1, 13, 16, 23, 25] have made possible not only acquiring and processing large amount of data from our environment, but also extracting relevant information and knowledge from such data. We can expect unprecedent innovative applications arising from the massive deployment of such technology in large variety of domains, such as urban planning, environmental protection, homeland security, health care, entertainment, education [19, 28, 29]. Enabling such a vision requires however addressing increasing privacy and confidentiality requirements in that very often stream data convey information related to individuals or information that are critical to organizations. Though stream data management systems have been extensively investigated by the research community, the focus has been on topics such as query processing[16, 25] and watermarking [27]. Issues related to privacy and confidentiality, and security more in general, have yet to be even understood.

In this paper we aim at developing a first definition of the privacy problem and to establish the main principles underlying the development of privacy-preserving techniques for stream data management systems. By refer to such systems as *Hippocratic data stream management systems* (HDSMS), by borrowing the term firstly introduced by Agrawal et al. [4], to emphasize the responsibility, in part imposed by existing laws and regulations, of protecting the privacy of the user's data. As part of their initial work, Agrawal and al. defined some principles that should guide the development of an Hippocratic Database Management System (DBMS). However, such principles were referred to conventional DBMS. We believe that stream data management systems are more crucial to privacy and have challenges that go beyong the challenges identified in the context of conventional DBMS.

1.1. Key Differences between databases and data streams

The most fundamental difference between traditional databases and data streams is that the latter are unbounded sequences of data continuously arriving according to a specific data rate. Such a difference impacts many features and components of DBMS architectures, such as the types of query that are supported and execution strategies for such queries. Here we elaborate in more details on such differences:

1. *Query types.* In conventional DBMS queries are snap-shot queries that retrieve data from a consistent state of the database at a specific time. By contrast, queries applied to data streams are usually continuous queries [10]. Continuous queries keep running as long as they are registered in the system to consider the newly-incoming data. Continuous queries make use of the notion of sliding windows [22]. A sliding window limits the query to the most recent portion of the stream that is long w time units.

2. *Access methods.* In conventional DBMS, the database records are usually accessed randomly with no notion of data order. In data streams, we usually cannot afford more than one *sequential* pass over the data stream. Moreover, once a data item passes, the system either explicitly processes this data or it loses it for ever.

3. *Incremental processing.* Conventional DBMS accept the user's query, process the query over the current snapshot of the data, and terminate the process once the result is returned to the user. In the case of data streams, the system streams the output as well it streams the input. A newly-incoming data item triggers the execution of the query plan operators to *incrementally* add the effect of this data item to the query answer. Blocking operators, e.g., join, group-by, and aggregates, are approximated with non-blocking operators that are incrementally evaluated to obtain approximate, but early, results.

4. *Approximate answers.* Conventional DBMS usually return exact answers to queries or answers with high accuracy. However, when dealing with data streams, some level of approximation can be acceptable in order to deal with the limited CPU time in front of the high arrival rates of data streams and to make up for the bounded memory resources in front of the infiniteness of the data stream.

1.2. Challenges

The differences we have outlined in the previous subsection introduce important challenges for the development of Hippocratic DBMS for stream data. The data streaming environment is a highly-demanding environment where continuous queries are executed over stream data that are usually generated at high rates. Continuous query processing requires highly efficient query processors to avoid loosing much of the input data. The limited CPU time is shared among the outstanding continuous queries to process data as they evolve. The enforcement of security and privacy measures may trigger heavy-weight processes that may heavily affect performance. As a result, security and privacy mechanisms should be optimized for speed and should be tuned to work in a single sequential pass over the streamed data.

To be suitable for data streams, security and privacy mechanisms should be able to process the stream data incrementally. A stream system continuously receives new data and expires old data that are not longer of interest to any query. Security and privacy mechanisms should take into consideration the effect of new data and, meanwhile, should fade the effect of expiring data. Moreover, security and privacy-preserving mechanisms should be able to run on top of approximate or incomplete data instead of the exact data to make up for any data loss. In case of approximate or incomplete data, approximations should not be skewed in a way that reveals the sensitive components of individual records.

The voluminous data that is streamed in and streamed out increases the system's vulnerability to security attacks. For example, the higher the number is of data encrypted with the same key, the easier it is to compromise that key. Security mechanisms should aim at releasing no sensitive information even after long periods of operation.

1.3. Contributions

In this paper we address the problem of privacy-preserving data management techniques for stream data. Such problem, already challenging for conventional database systems, is much more difficult in a context characterized by huge amounts of fast arriving data and by strong performance requirements. The major contributions of this paper can be summarized as follows:

1. We shift the problem of privacy assurance from conventional databases to the more challenging paradigm of data streams.

2. We revisit and substantially extend the privacy principles, initially devised for conventional databases, in the context of stream data. In particular, we introduce two more principles that are stream-specific.

3. We identify the major challenges that are to be addressed in the design of an Hippocratic data stream management systems.

4. We develop an architectural design of a Hippocratic data stream management system (HDSMS) that addresses those challenges and can be used as a reference framework for research in the area of security and privacy techniques for stream data.

The remainder of this paper is organized as follows: Section 2 presents twelve principles representing the main requirements that should guide the development of privacy-preserving management systems for stream data. Section 3 emphasizes the principles of Hippocratic data streams through some example applications. Section 4 discusses the challenges that must be addressed in the development of an HDSMS. Section 5 presents our design of a data stream management system that complies with the principles of Hippocratic data streams. Section 6 overviews related work. Finally, Section 7 concludes the paper and gives some guidelines future research.

2. Principles of Hippocratic Stream Data Management Systems

In this section, we present twelve principles that guide the behavior of Hippocratic stream data management. The first ten principles were initially proposed for conventional database systems, and we revisit them here in the context of stream data. The last two are novel and have been devised specifically for stream data.

1. *Purpose Specification.* This principle states that the purpose for which the data is collected needs to be collected and associated with the data itself. In data streams, the purpose of data collection may change over time. At the extreme, the purpose of the data is streamed continuously into the system with the streaming of the data.

2. *Consent.* This principle guarantees that the purpose for which the data is collected has the consent of the user. As in conventional database systems, a data stream management system must maintain the consent of the user over the lifetime of the stream.

3. *Limited Collection.* Data collection should be limited to the minimum amount of data that satisfies the user's specified purposes. There are several challenges one has to address in order to limit the data collection in a data stream management system. For example, streams with no associated queries should not streamed into the system at all; stream data items that are of no interest to any query are filtered at the system's input buffers. Additionally, since the streaming environment is dynamic, i.e., existing queries expire and new queries arrive, rules and filters that the system uses to limit the data collection change dynamically.

4. *Limited Use.* This principle ensures that the data is used by queries that do not violate the purposes of the collected data. In a data stream management system, at the time of query registration, admission control policies are applied to accept or to reject the query. More flexibly, if the admission control detects that a query will violate the purpose of the data, then the admission control can negotiate the query demands with the query's owner instead of rejecting it.

5. *Limited Disclosure.* The system is not allowed to release any information to a third party that is outside the system without the owner's approval. In data streams, limited disclosure has the same meaning as in conventional databases with one additional challenge. A minimum quality of the released data should be included in the owner's consent, and when the system releases data to a third party, the minimum data quality has to be ensured. Otherwise, the third party may not get a true image of the stream.

6. *Limited Retention.* Limited retention necessitates the immediate deletion of the user's data once the associated purpose is fulfilled. To map the same concept to data streams, the user's data should not be maintained beyond the largest time-window of any query that accesses the stream.

7. *Accuracy.* The accuracy of a database system means how accurate and up-to-date the information stored in the database is. For data streams, accuracy has two interpretations: (a) the accuracy of the approximate version of the stream inside the system and (b) the query output delay, which is measured by the time required for a data item to appear in the output (i.e., to bring the output up-to-date).

8. *Safety.* Saftey entails the adoption the security measures protecting sensitive data from various types of attack. The challenge in data streams is represented by the huge amount of data that needs to be secured. Moreover, security mechanisms should consider the scarcity of resources in the streaming environment by avoiding CPU-intensive operations.

9. *Openness.* Openness allows the individual to whom the data are related to access all information about himself stored in the database. In data stream systems, openness is very difficult to achieve because of the continuous change in the stream data due to the arrival and the expiration of data. A stream owner should be able to retrieve the data that are actually streamed inside the system and to compare these data with its original data. Then, the stream owner can decide how satisfactory the system's sampling rate is.

10. *Compliance.* As for conventional databases, compliance means making sure that interested subjects, for example data owners, be able to verify that the system complies with the privacy principles. At any time instant, the stream owner should

be able to track the query answers that are based on his own stream to check that his privacy requirements are met. Verifying the compliance of a data stream management system is a hard task given the huge amount of output data that are streamed out of the system over time.

11. *Faithful Representation.* Because of the scarcity of system resources, the query processor may not be able to catch up with the stream arrival rates. Consequently, some data may drop from the input buffers. A Hippocratic data stream management system guarantees that the processed data are a faithful representative subset of the original stream data. Data dropping should (a) treat all streams fairly and (b) result in a uniform random sample of the original streams. No specific streams or stream values should be repeatedly ignored either accidentally or intentionally.
12. *Minimum QoS.* A Hippocratic data stream management system should guarantee a *minimum Quality of Service* (QoS) per registered query. This principle assures that the security and the privacy mechanisms do not overload the system resources. Adoption of this principle also prevents *denial of service* attacks where the system accepts a huge number of queries that are beyond the system's processing capabilities. According to this principle, a query is not accepted unless the system can secure enough resources for the processing of this query.

3. Applications of Hippocratic Data Streams

Almost every data streaming application has some relevant privacy and security requirements. However, some applications are particularly challenghing because privacy requirements are dynamic and are streamed to the system with the streaming of the data itself. Here we briefly discuss three such applications and identify the main challenges they pose.

3.1. Streams of Video Frames

A video clip is a stream of video frames that are displayed in a timely fashion. Users are allowed to view video clips based on some access permissions. The simplest case is either to grant or to reject the user's request to view the entire video clip. Alternatively, the user is authorized to view a subset of the video frames and is prohibited from viewing other frames that have higher levels of confidentiality. These two cases can be handled efficiently through traditional access control policies that operate on the frame level as its basic granularity.

Our application scenario assumes that every user is allowed to view the entire video clip after blurring some portions of the video frames that contain confidential information, e.g., faces of the people. In this case, the trajectories of the moving faces that have special access control policies are streamed to the system with the streaming of the video frames. Then, access control mechanisms are applied online to blur specific portions of each video frame based on the user's access permission.

3.2. Streams of Spatiotemporal Data

Assume that a data stream management system (*DSMS*) receives a stream of spatiotemporal readings that represent the locations of moving vehicles at specific time instants. Based on a continuous query, the DSMS feeds each vehicle with information about its nearest service stations, e.g., gas stations, restaurants, and hospitals. Also, the *DSMS* may release a vehicle's information to its nearest service stations for the sake of advertisements provided that the system gets the vehicle's consent. The *DSMS* can bargain on behalf of the vehicle's owner with the service stations for a better service with less cost.

In the above scenario, the system maintains information about the users' vehicles and the service stations. The system authorizes only close-by stations to advertise themselves to a vehicle and hides the vehicle's information from other stations to avoid that the vehicle be spammed. Each user specifies what type of information about himself can be released, e.g., location, name, and preferences. The user also tells his notion of distance, i.e., how near the nearest service stations should be. Service stations have the notion of confidentiality as well. The prices of each station may be kept confidential to allow the system exclusively to bargain for a better rate. Also, a service station may require its information to be released only to specific types of vehicles or to vehicles with specific preferences. Finally, it is the system's responsibility to match the requirements of both the vehicle and the station.

3.3. Streams of Retail Transactions

The online processing of retail transactions is a rich area with respect to which to explore the functionalities of an HDSMS. A retail transaction contains information about the sold items, their quantities, their prices, the customer information, and the method of payment. Various entities are entitled to view differ-

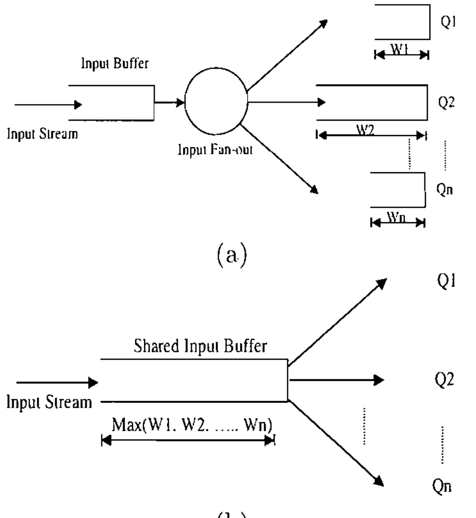

Figure 1. Shared execution of queries over the same data stream.

ent pieces of information. For example, a cashier is not allowed to view any information once a transaction is complete. A store's manager should access only the information of his local store. The management of the retail store series has the authority to view the information of all stores and, moreover, to grant access to individuals. A third party may be responsible for processing the customer's payment. Research centers are authorized to access a view that is defined over the data for analysis purposes only. A customer should be able to specify his confidentiality requirements, for how long to keep his own data, and whether or not to use his data in analysis and advertisement purposes.

In the above scenario, many entities interact to complete a single transaction. An HDSMS has the responsibility of enforcing all the security and the privacy rules of each entity. An HDSMS is globally trusted and is believed to protect the rights of retail stores, customers and other parties that are involved in the transactions.

4. Challenges

Hippocratic data stream systems pose several challenging problems. Here we elaborate on these challenges, outline possible approaches and identify open research directions.

4.1. Shared Execution

In a data stream system, a user may execute a query that accesses multiple streams; also multiple users may execute queries that access the same stream. Each user has certain authorizations and restrictions over each stream. One of the main challenges in this respect is how to enforce authorizations without degrading the system's performance. For example, the authorizations that are common among the current users should be applied once instead of applying them to each user separately. One possible solution is that the stream owner creates a *view* for each user that determines the access permissions of the user's queries. In the *create view* statement, the stream owner specifies the following parameters:

1. A set of attributes. This set of attributes restricts the scope of the user's queries.
2. A minimum sampling rate. The stream owner grants the user a minimum sampling rate to ensure that the user's queries capture the minimum required behavior of the stream that is necessary for these queries.
3. A maximum sampling rate. The stream owner restricts the amount of information that a query can see through a maximum sampling rate.
4. A maximum size of a sliding window. This parameter limits how far a query can go in the past. The query must operate over a window that is of equal or less size.
5. A set of filters. The stream owner has the authority to apply a specific filter over his data before it is fed to a query. For example, a retail store may prevent certain queries from accessing transactions with special items or with prices over a certain amount.

Figure 1a illustrates a naive approach under which the input stream is pushed into the system's input buffer and then the output from the buffer fans out to each query based on its access permissions. The buffer manager feeds each query Q_i with stream readings over the most recent time-window w according to its specified minimum and maximum sampling rates and window W_i. Stream tuples are projected to eliminate any unauthorized attributes and are filtered out to limit the access to authorized values only. This approach suffers from duplication in the input buffers of outstanding queries where a single tuple is fed to many queries. If no security and privacy techniques are in place, the system would use a single input buffer per stream that feeds all queries with input tuples. The major challenge here is to avoid the duplication overhead in the query input buffers that arises because of security and privacy.

Figure 1(b) illustrates the case where one shared buffer is used for all queries. The shared input buffer should include all the attributes that are accessed by all queries and should be long enough to satisfy the maximum window size, i.e., max(W_1, W_2, $\cdots$, W_n), with the highest required sample rate. The key issue is to avoid the materialization of the view of each query and to use one global view that is shared among all queries.

4.2. Dynamic Change of Security and Privacy Preferences

The security and privacy preferences can be streamed with the streaming of the data to indicate the privacy preferences of the current portion of the stream. Recall the example of video streams in Section 3.1 where the trajectory of the objects that needs protection is streamed with the streaming of the video frames. The dynamic and continuous change in security and privacy preferences places two major challenges:

1. *Synchronization between the stream tuples and their associated preferences.* A lag between the stream tuples and their preferences can undermine data privacy. Imagine the problem of blurring a portion of the video frame after the object that is required to be protected is already displayed. The lag between the stream tuples and associated preference may occur due to network delays between the stream source and the system.
2. *Dynamic query optimization.* Certain optimizations take place to tune the query performance based on the security and privacy preferences. A change in these preferences triggers a change in the query plans that are executed over the stream. The online optimization of queries should make sure to avoid wasting the systems resources at run-time. Some work has been conducted to reorganize the operators dynamically in the context of the Telegraph project based on the concept of Eddies [7].

4.3. Stream Ciphers

Stream ciphers, e.g., [20] are used to encrypt and decrypt stream data. The huge amount of data that is conveyed in a data stream requires careful *key management*. The longer the data sequence that is encrypted with the same encryption key, the easier it is to compromise that key. The key should be changed frequently or generated using a pseudo random generator that approximates the one-time-pad technique.

Key synchronization is another challenge in stream ciphers. The streaming environment is lossy by nature and, hence, stream tuples drop from input buffers if no sufficient resources are allocated for these tuples. Encryption techniques should take into consideration possible loss of data that may *desynchronize* the decryption key from the stream cipher. Some work has been initiated in [5] to develop slef-synchronizing stream ciphers that tolerate possible data loss.

Stream ciphers should be tuned to provide fast encryption and decryption operations while avoiding CPU-intensive operations. The objective of stream ciphers is to secure streams without competing with other system's functionalities for system resources. The system's total throughput should be minimally affected by the encryption and decryption processes.

4.4. Compliance

An important mechanism supporting proof of privacy compliance by the system is represented by logs of audit trails. Such logs are generated by the DBMS upon each data access and give users the ability to track all the operations that have touched their data. Users can investigate the audit trails at any time to verify the compliance of the system to their privacy preferences. One possible solution to achieve compliance and to monitor users' queries is to generate, for every user, log entries on the form: *(Query, Item, ts, $\cdots$)*. Such a log entry records the query issued by the user, the item(s) accessed by the query, and the timestamp at which the query took place. Notice that the item can refer to various granularities of data, e.g., attribute, tuple, or table. Other information can be added to record the purpose of the query and a reference to the user's consent.

In data stream systems, audit trails are extremely huge. Therefore, storing the infinite data composing the original stream is not feasible. As a result, logging the answer to multiple queries over data streams would be impossible. An alternative solution is to store audit trails at coarser granularities. In contrast to recording a data item and the timestamp at which the item is accessed, a lazy approach can be adopted according to which the log records a stream range and the period of time (ts_{start}, ts_{end}) over which the stream range is accessed. This approach reduces the accuracy of the audit trails but addresses the problem of the limited memory space. The audit trails can be enhanced with extra summaries to sketch the result that is returned to the query over the associated range of the stream. The log entries that are generated have the form: *(Query, strem-range, ts_{start}, ts_{end}, Summaries, $\cdots$)*.

Another challenge in the data streaming environment is to produce the audit trails online without significantly degrading the performance of continuous

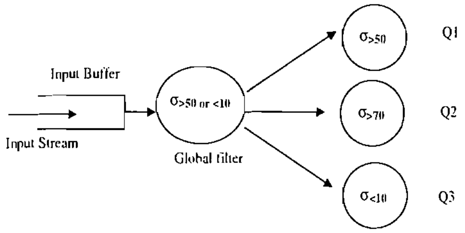

Figure 2. Global filters.

queries. The coarser the granularity over which the query answer is logged, the less load is placed over the system.

4.5. Limited Collection

Limiting the data that is streamed into the system to the minimum possible has two main advantages: (1) it decreases the processing load of the system and (2) it supports compliance to the *minimum collection* principle. Simply, if a stream has no associated query, it remains inactive and is not streamed into the system. Once a query is registered at the system, all its inactive target streams are initialized and are streamed into the system. A warm-up period that is equal to the length of the query's sliding window is required to fill up the query's input buffers. Similarly, any attributes in the stream that are of no interest to any query are not streamed into the system.

In contrast to the above *stream-level* or *attribute-level* data collection, the *tuple-level* data collection is more challenging. The system should be able to decide whether a single stream tuple is of interest to any outstanding query. A *global* filter is pushed at the system's input buffer to take out, as early as possible, any stream tuples that do not satisfy the selection predicates of *all* outstanding queries. Figure 2 illustrates the case where a global filter for three queries is pushed to the input buffer level to take out tuples that are of no interest to the three queries. Pushing filters to the input buffers requires the buffer manager to be equipped with additional processing capabilities.

4.6. Limited Retention

The principle of *limited retention* in traditional databases is relatively straightforward compared to data stream systems. In traditional databases, once a data item fulfills its associated purpose, it is deleted from the database and from the logs as well. Consequently, the data item is no longer visible to any query.

In data stream systems, data are not persistent. They can be in the system's input buffers, the queries' input buffers, or queued in one or more query plan operators. Moreover, the effect of stream data is reflected in the maintained summaries over the stream. To comply with *limited retention* principle the HDSMS has to ensure the following:

- *Queries window size:* Continuous queries usually restrict their interest to a certain window over the stream. The retention time for a stream has to be larger than the query window size. If the query window size is larger than the stream retention time, then the admission control component has to either reject the query or negotiate with the query's owner to reduce the window size.
- *Fading summaries:* Streams may have summaries built on top of them. With the retention time principle, the time dimension has to be considered while building the summaries such that the effect of the tuples should fade over time until the effect disappears after the specified retention time.
- *Dynamic limited retention:* Streams are continuous and the retention period may change over time. The system should provide a mechanism to readjust the existing data based on the new retention period.

4.7. Limited Disclosure

A stream owner has the right to instruct the data stream system to release its data to a third party to obtain a specific service. For example, a moving vehicle requests the system to release its information to the nearest service station or a retail store requests a release of its information to a research center. In the context of data streams, we identify the following two basic challenges:

1. *Faithful disclosure.* If the system is to disclose the user's information, it should disclose them correctly. Imagine an overloaded data stream system that releases the information of a retail store to a research center with low sampling rate. Missing tuples in the released stream can negatively affect the provided service. The system should be able to commit to release a *good-quality* image of the user's data to the intended third party.
2. *Timed-out disclosure.* The system should be able to accept timed-out requests from the user to release his information. For example, a moving vehicle allows the system to release its information to the nearest neighbor. As the vehicle moves, its nearest neighbor changes. The system should be eager to time out the disclosure request once the

service station is not recognized as the vehicle's nearest neighbor.

4.8. Quality of Service

Traditional databases provide exact and accurate answers. On the other hand, data streams are tuned for approximate answers to alleviate the bursty nature of data streams that usually arrive in high rates. The *quality of service (QoS)* principle assess "how approximate are approximate answers?" and places minimum guarantees over the quality of service measures. Moreover, the system is required to notify the query owner whenever the *QoS* measures are violated. Various (QoS) measures can be in effect. Some of these measures are summarized as follows:

1. *Maximization of the number of output tuples.* A query processor returns a subset of the query's exact answer. The larger the returned subset, the less output tuples are lost and the more accurate the result is.

2. *Uniformity of the output.* A query processor aims at providing a subset of the answer that approximates a uniform sample of the exact answer. The returned subset should not be skewed towards any tuples. Notice that taking a uniform sample of the input does not imply that we will obtain a uniform sample of the output, e.g., the join operation.

3. *Output delay.* The difference between the current timestamp and the tuple's timestamp represents the delay imposed by the system in the tuple processing. A query processor minimizes the output delay to achieve a better response time.

5. Design of an Hippocratic Data Stream Management System

Figure 3 illustrates the basic components that, in our view, should be part of an HDSMS. We introduce several components that are specific to privacy, such as *Security and Privacy Preferences*, *Data Collection Manager*, *Data Retention Manager*, and *View Manager*. Each of these components plays a role in assuring the compliance to privacy principles. These components interact with other components, that are typically part of stream data management systems, such as *Stream Registration Unit*, *Summary Manager*, and *Query Admission Control* to ensure full system security. The basic components of an HDSMS are sandwiched between two layers of the *Security Manager* (input stream and output stream security managers) that provide security services. Such services include encryption, decryption, authentication, and non-repudiation. Notice that integrity is usually not assured for data streams due to the frequent and natural data loss over the communication medium in the streaming environment. The *Security Manager* is responsible to achieve the required security services, i.e, achieve the *safety principle*.

The stream owner communicates with the system through two major interfaces: the *Stream Registration Unit* and the *Stream Manager*. The *Stream Registration Unit* accepts *Stream Definition Language (SDL)* statements defining the schema of a new stream along with the privacy preferences of the stream owner. The stream owner is allowed to define a view according to which each user accesses the stream, as discussed in Section 4.1. Notice that the SDL statements may be changing continuously to form a stream of preferences that controls the access to the associated stream data (as described in Section 4.2). The registration of a new stream places no load on the system because a stream is not physically read into the system unless it has an associated query. The load is imposed at the time of query registration not at the time of stream registration. Hence, the system grants all the stream registration requests unless the stream owner requires security services that are not provided by the system. The *Stream Registration Unit* acquires the *purpose* of the stream data and the *consent* of the stream owner before a stream is successfully registered in the system.

The *Stream Manager* physically receives the input stream tuples and prepares it for further processing inside the system. It consists of two basic components: the *Buffer Manager* and the *Summary Manager*. The *Buffer Manager* accepts the newly incoming stream tuples and maintains them in a raw representation while the Summary Manager builds stream intermediate representations of the streams that are suitable for query processing. The *Stream Manager* should be eager to push the incoming tuples into the systems as soon as they arrive to fulfill the *accuracy principle*. Also, when the system load is high, the *Stream Manager* applies load shedding techniques to relieve the system from the burden of stream bursts arriving at high rates. However, load shedding techniques provide the system with a uniform sample of the input stream and abide to the *faithful representation principle*.

The system accepts a user's query expressed according to the *Stream Manipulation Language (SML)*, e.g., an SQL query. The *Query Admission Control Unit* accepts, rejects, or negotiates a query with the query owner based on (1) the access privileges of the query owner to limit access to confidential data and based on

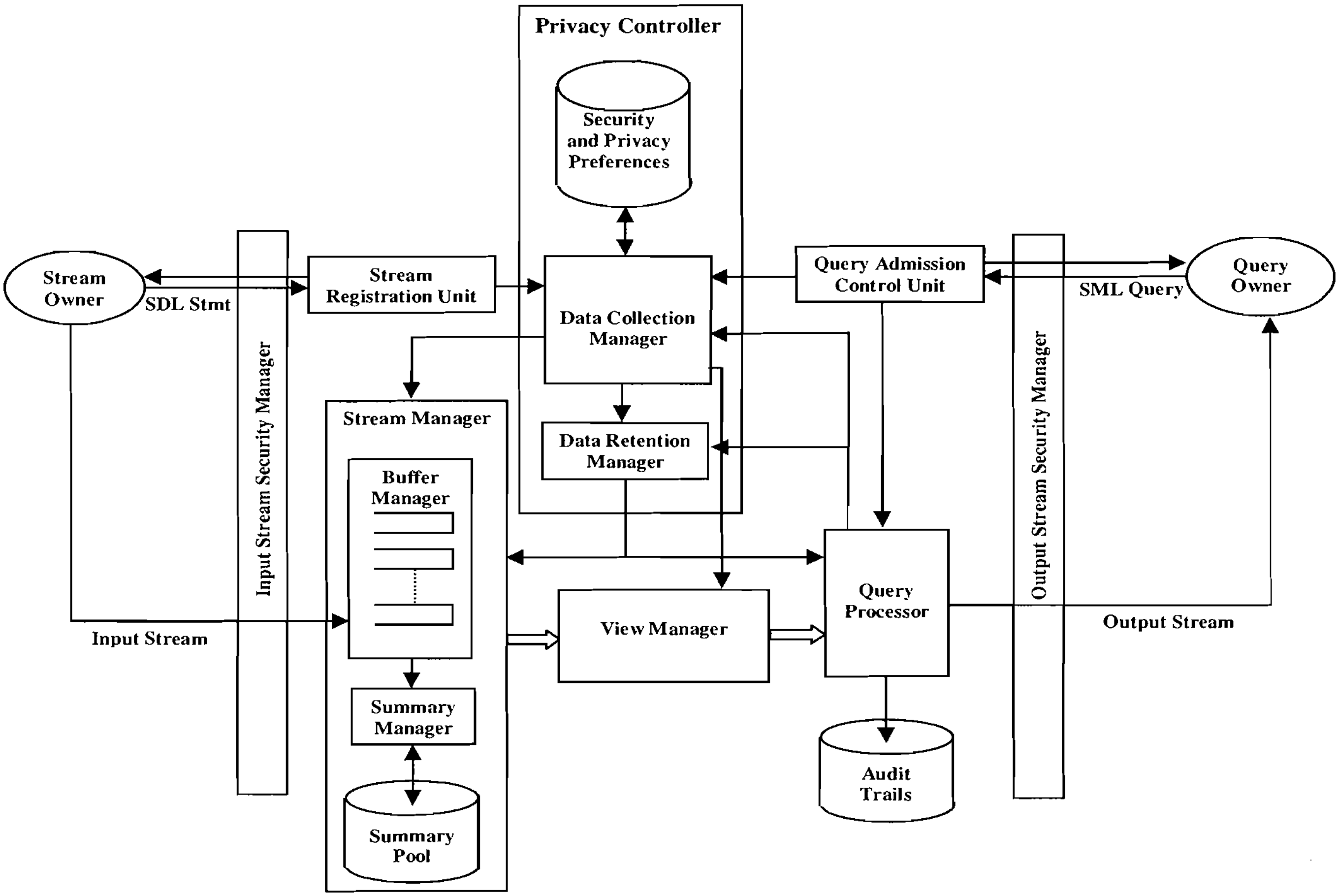

Figure 3. A preliminary architectural design for a Hippocratic data stream management system

(2) the current system load to guarantee a minimum QoS for the new query as well as the existing queries. The *Query Admission Control Unit* fulfills both the *limited use* and the *minimum QoS* principles. Once a query is accepted, it is dispatched to the query processor to compete for system resources.

The *Data Collection Manager* maintains both the privacy preferences of the stream owner and the access requirements of admitted queries. The *Data Collection Manager* has the following three basic functionalities:

1. It controls the *Stream Manger* to limit the data collection to the attributes that are of interest to outstanding queries (*limited collection principle*).
2. It contacts the *View Manger* to establish a view for each user or query to hide confidential data from unauthorized entities (*limited use principle*).
3. It informs the *Retention Manager* of how long to maintain each piece of collected data (*limited retention principle*).

Notice that the *Data Collection Manager* accepts feedback from the *Query Processor* to further prune the collection of data. The *Data Collection Manager* uses the feedback of the *Query Processor* to push global filters as early as possible at the system's input buffers (as described in Section 4.5).

The *Data Retention Manager* is responsible for deleting the user's data from the system once they expire (i.e., get outside the largest sliding window of interest) or once they fulfill their associated purpose. Based on the maintained security and privacy preferences, the *Data Collection Manager* informs the *Data Retention Manager* of the expiration time of each data item. Whereas the *Query Processor* informs the *Data Retention Manager* of any data items that have already fulfilled its associated purpose. Accordingly, the *Data Retention Manager* removes these data items from the system input buffers, the summaries and the operator queues of query plans.

The *View Manager* is interposed between the input streams and the *Query Processor* to enforce the privacy preferences of the stream owner. Instead of materializing each view, the *View Manager* shares the execution of multiple queries on the same buffers (as described in Section 4.1). The *View Manager* is responsible for both the stream's raw input data and its maintained summaries. Notice that the stream owner is granted a view with no limited access over his own stream to follow the *openness principle*.

The *Query Processor* executes the registered continuous queries over their associated views and streams the result out to the query owner. The stream input data are pushed into the *Query Processor* by the *View Manager*. Each data item propagtes up the query plan passing through various query plan operators until the result is obtained at the operator on top of the query plan. The *Query Processor* records its activities in audit trails that are available for the stream owner to check the *compliance* of the system to the Hippocratic principles.

6. Related Work

In this section, we survey related work in the context of data streams through the following two major guidelines:

(1) We survey several prototypes for data stream management systems that are developed by various research groups. (2) We highlight two research directions that provide security and preserve the privacy of data streams.

Prototype data stream systems have been developed to address the challenges of streaming environments. Although these prototypes do not address data privacy as their primary concern, they provide the basic foundation for continuous query processing over data streams [10]. Stanford STREAM [9, 25] focuses on resource management in the context of data stream processing. The AURORA project [1] optimizes for certain QoS measures. Telegraph [13] achieves adaptivity in query execution by dynamically changing the order of query operators during execution. The Niagara project [16] performs group optimizations over sets of continuous queries. Gigascope [17] is a data stream system that is developed at AT&T to process streams of network traffic. The Fjord project [24] proposes a framework for query execution plans over data streams. The COUGAR system [11] introduces a new data type for sensors that facilitates query processing over sensor data. Nile [23] is a research prototype that is currently being developed at Purdue University. Nile extends relational database management systems with the data streaming functionalities.

Like for traditional databases, the work carried out so far to assure privacy of data streams can be classified into two directions: (1) maintaining statistics over data streams and (2) developing security mechanisms for data stream processing. We discuss each direction briefly in the following sections.

6.1. Maintaining Statistics over Data Streams

To preserve the privacy of individual stream data items, queries are allowed to access only statistical information over a window of the stream [18, 31]. In addition to privacy requirements, maintaining statistics over data streams was initiated by the need to represent the stream data items using low memory requirements and to provide fast approximate answers using a smaller subset of the data. Stream data are summarized incrementally as data arrive into the system. Then, queries are performed on top of the summaries to extract various statistics. Although statistics provide an insight of the stream's behavior, they blur the sensitive data of individual data items.

Sampling is one of the solutions to provide a smaller subset of data that faithfully represent the original stream. Sampling with a reservoir techniques are tuned for data stream processing in [8]. Various stream operations, e.g., the join operation [3, 15] and the group-by operation [2], are processed on top of a random sample of the original stream. Sketches [6] are another form of data summarization. Some sketching techniques are developed to capture the stream's most frequent items [14]. The most frequent item list can represent the stream data for some special distributions, e.g., the Zipfian distribution. Several techniques are proposed to build approximate histograms [21, 26] and wavelets [12, 30] in a single pass or in few passes to fit in the online processing paradigm of data streams.

6.2. Security Mechanisms for Data Stream Processing

To the best of our knowledge, there is very little work that addresses the security for data stream systems and to date only a preliminary analysis of security issues has been undertaken [5]. Although the main research focus is directed to the performance issues of stream query processing, data stream systems can borrow many security techniques from traditional database systems. For example, encryption and decryption techniques are applied in the streaming environment to ensure confidentiality whenever sensitive data is required to be protected. In contrast to block ciphers, stream ciphers fit better in the streaming environment. A preliminary approach based on such type of cipher has been recently proposed to provide a fault tolerant efficient encryption mechanism for stream data [5]. However, such a mechanism alone is not enough to provide a comprehensive solution to the problem of securing stream data.

7. Conclusions and Future Work

Hippocratic databases extend the functionalities of traditional databases with privacy-preserving capabilities. Similarly, privacy preserving data stream systems are expected to be the next step for the data streaming paradigm. In this paper, we presented a visionary architectural design for HDSMS. We aimed at allowing privacy preserving capabilities to go hand-in-hand with the efficient execution of continuous queries over data streams. We highlighted some applications that benefit from Hippocratic data stream management systems. In our design, we adhered to the ten principles of Hippocratic databases and extended these principles with two more principles that are crucial to data streams. In this paper, we identified the major challenges one has to address in the development of an HDSMS and we believe that these challenges will trigger many directions for future research. The realization of our design into an initial prototype for a Hippocratic data stream system will further provide additional research issues.

References

[1] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 2:120–139, August 2003.

[2] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 487–498, May 2000.

[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 275–286, June 1999.

[4] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *Proceedings of the Intl. Conf. on Very Large Data Bases (VLDB)*, August 2002.

[5] M. Ali, M. ElTabakh, and C. Nita-Rotaru. Robust security mechanisms for data streams systems. In *Technical Report CSD-04-019, Purdue University, Computer Sciences Department*, May 2004.

[6] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the annual ACM symp. on Theory of computing*, pages 20–29, May 1996.

[7] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 261–272, May 2000.

[8] B. Babcoc, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proceedings of the Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 633–634, Jan. 2002.

[9] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the ACM Symp. on Principles of Database Systems*, pages 1–16, June 2002.

[10] S. Babu and J. Widom. Continuous queries over data streams. In *SIGMOD Record*, volume 30, pages 109–120, Sept. 2001.

[11] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proceedings of the Intl. Conf. on Mobile Data Management*, pages 3–14, Jan. 2001.

[12] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *Proceedings of the Intl.Conf. on Very Large Data Bases (VLDB)*, pages 111–122, Sept. 2000.

[13] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *Proceedings of the First Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2003.

[14] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of the Intl. Colloquium on Automata, Languages and Programming*, pages 693–703, July 2002.

[15] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. In *Proceedings ACM SIGMOD Intl. Conf. on Management of Data*, pages 263–274, June 1999.

[16] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagracq: A scalable continuous query system for internet databases. In *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 379–390, May 2000.

[17] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 647–651, June 2003.

[18] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *of the Thirteenth Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 635–644, Jan. 2002.

[19] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *Proceedings of International Conference on Acoustics,*

Speech, and Signal Processing (ICASSP). pages 2675-2678, May 2001.

[20] S. Fluhrer and D. McGrew. Statistical analysis of the alleged rc4 keystream generator. *Fast Software Encryption, Springer-Verlag*. 2000.

[21] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proceedings of the annual ACM symp. on Theory of computing*, pages 471-475, July 2001.

[22] M. A. Hammad, T. M. Ghanem, W. G. Aref, A. K. Elmagarmid, and M. F. Mokbel. Efficient execution of sliding-window queries over data streams. Technical Report CSD-03-035, Department of Computer Science, Purdue University, June 2004.

[23] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Eltabakh, M. G. Elfeky, T. Ghanem, R. G. andIhab F. Ilyas, M. Marzouk, and X. Xiong. Nile: A query processing engine for data streams. In *Proceedings of the Intl. Conf. on Data Engineering*, page 851, April 2004.

[24] S. Madden and M. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings of the Intl. Conf. on Data Engineering*, pages 555-566, Feb. 2002.

[25] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *Proceedings of the First Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2003.

[26] V. Poosala and V. Ganti. Fast approximate answers to aggregate queries on a data cube. In *Proceedings of the Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 24-33, July 1999.

[27] R. Sion, M. J. Atallah, and S. Prabhakar. Resilient rights protection for sensor streams. In *Proceedings of the Intl. Conf. on Very Large Data Bases (VLDB)*, pages 732-743, Sept. 2004.

[28] S. Srinivasan, H. Latchman, J. Shea, T. Wong, and J. McNair. Airborne traffic surveillance systems: video surveillance of highway traffic. In *The 2nd ACM international workshop on Video surveillance & sensor networks*, pages 131-135, 2004.

[29] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Communications of ACM*, 47(6):34-40, 2004.

[30] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 193-204, June 1999.

[31] J. Yang and J. Widom. Incremental computation and maintenance of temporal aggregates. In *Proceedings of the Intl. Conf. on Data Engineering*, pages 51-60, April 2001.