HipStream: A Privacy-preserving System for Managing Mobility Data Streams

Huayu Wu, Shili Xiang, Wee Siong Ng, Wei Wu and Mingqiang Xue
Data Analytics Department
Institute for Infocomm Research, A*STAR, Singapore
Email: {huwu, sxiang, wsng, wwu, xuem}@i2r.a-star.edu.sg

***Abstract*—Personal mobile data are being extensively collected by various service providers, in the form of data stream. Most service providers promise their customers for not misusing their data by paper-based agreement. However, the customers have no way to know whether the agreements are strictly followed or not, unless any scandals of private data misuse are revealed. To guarantee the correct use of customers' personal data and assure them of the service safety, system-level data privacy control between the data owners (i.e., customers) and the data users (i.e., service providers) is in compelling need. Inspired by the concept of Hippocratic data management, we design and implement a system, HipStream to systemically enforce different Hippocratic principles to preserve data providers' privacy when they send their data stream for services. In this paper, we describe the architecture of the HipStream system and demonstrate how it meets those privacy principles.**

I. Introduction

Nowadays personal mobile data are extensively collected by various service providers. For example, when a user makes a phone call, her location and calling time would be exposed to the telco company; when she uses a smart card for public transport fare payment, her daily route would be known by the public transport company; when she checks in her Facebook or uses any location-based services, her trajectories would be recorded by the service providers. Moreover, such mobile data of a person is normally continuously collected by the service providers, in the form of data stream, as many services are routinely used by their customers.

More and more people start realizing the privacy issues when they share their location data with service providers, especially after several scandals that governments and companies misused customers' private data have been revealed in recent years. Indeed, most service providers will promise privacy protection to their customers before the customers start using their services and sending data to them. However, this promise is generally in terms of paper-based agreement forms. There is a lack of systemic control over the data use agreement, and the customers can hardly know whether their data have been misused unless any scandals are revealed to the public.

Recent research proposes location-based anonymization techniques, which encourages location-based service users to send their approximate, rather than exact location information to the service providers [1]. Nevertheless, there will always be a trade-off between the accuracy and the utility of the data. This approach inevitably decreases the quality of services.

To guarantee the quality, most services require the accurate information provided from customers. On the other hand, most customers really concern about how the service providers make use of their data, and for what purposes. To this end, the concept of Hippocratic data management system ([2], [3]) needs to be re-mentioned.

The idea of a Hippocratic database or data stream management system is to put data provider's privacy as major concern during data collection and management. There are ten principles proposed to define a Hippocratic data system, among which *limited collection, limited use* and *limited disclosure* are at the core. Limited collection requires a system to collect the minimum amount of data from a user to provide service to her. Limited use ensures that only the operations with the authorized using purpose of the data, which is defined by the data provider, can be carried out. Finally, limited disclosure allows only the data provider approved operations to access the corresponding data (or part of data).

As mentioned, most service providers promise these privacy principles only by paper-based agreement, by which customers could not monitor and verify the real use of their data. There is a compelling need of building a third-party data collection and management system that controls the data flow from customers (i.e., data owners) to service providers (i.e., data users). More importantly, the system should be able to help the data owners to preserve their data privacy according to their ad hoc specifications. In other words, there should be a system-level enforcement of privacy protection before the data users can access the data, so that data providers can be assured that their data will not be misused.

In this paper, we will demonstrate our data stream management system, HipStream, which fully implements the above-mentioned Hippocratic privacy requirements. Our system will automatically enforce the privacy policies specified by data providers, and also limit the access to the data by the service queries as promised in the principles.

The rest of this paper is organized as follows. We review some existing works that are related to ours in Section **??**. We briefly describe the architecture and implementation of the HipStream system in Section III. We demonstrate the HipStream system in use case in Section IV. Finally we conclude this paper in Section V.

II. Related Work

Due to the fact that many data are collected in the form of data stream, how to effectively and efficiently manage streaming data and support monitoring queries has been studied for many years. There are a number of data stream management

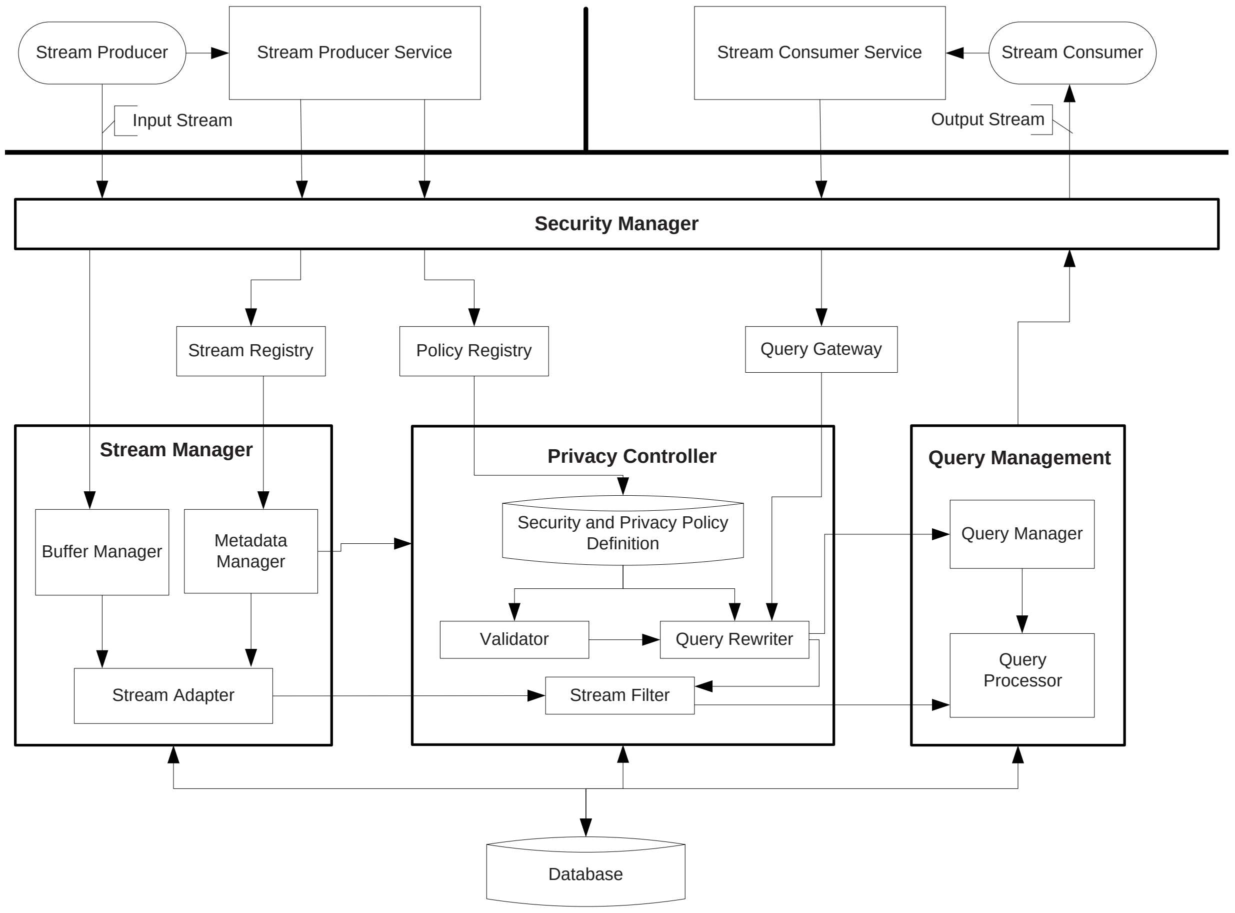


Fig. 1. Architecture of HipStream system

prototypes and systems developed in both academic (e.g., [4], [5]) and commercial (e.g., [6], [7]) domains. However, most of these early systems did not pay much attention to privacy protection on the collected data.

Later there are many research works focusing on how to support privacy protection in a stream system. Lindner and Meier [8] design a filtering operator and apply it to the stream query processing results to filter the output based on relevant access control policies. Nehme et al. [9] embed security policies into data stream, by security punctuations (SPs). Their query processor analyzes the SPs in each data stream, and enforces the policies during query processing. This framework is further improved by supporting dynamic access authorization of query issuers [10], [11]. Carminati et al. [12] propose another framework to enforce access control policies for stream query processing. They model continuous queries as graph of algebraic operators, and focus on query rewriting to incorporate policies into query graphs.

However, the existing systems only focus on access control, which is similar to the limited disclosure principle in the definition of a Hippocratic data stream system. In our system, we also support limited use and limited collection. Especially for the principle of limited collection, to the best of our knowledge, there is no existing system that considers this function, though this is a crucial requirement for a Hippocratic stream system to assure data safety to data providers.

III. System Architecture and Implementation

The architecture of the HipStream system is shown in Fig. 1. Let us walk through the architecture by assuming a use case. When a data provider would like to send her data stream to the system, she will register the data schema with the system first. Meanwhile, she can also specify the privacy policies, including who can access which parts of her data, for what purpose and under what kinds of conditions. The policy will be registered into the Policy Controller. Then she can start sending her data to the system. Each incoming data record will temporarily reside in a data buffer, and then be adapted according to the requirement of the query processor.

For service provider or other data consumers, they can issue queries to the system, together with the query purpose. The query will go through the query gateway and reach the query rewriter. The role of the query rewriter is to consult policies specified by the data owner, to validate whether the query is permitted, or needs to be rewritten. In the current system, we support SQL query. For example, if a driver only allows the service to view her data when her driving speed is less than 100 kph, then the query rewriter will enforce this policy by appending a condition of “speed $<$ 100” to any queries issued by the service. Also, the query rewriter will reject the query if the query purpose is not consistent with the authorized purpose on using the data. This component actually enforces the limited disclosure and the limited use principles.

After being validated, the query will be registered to the query manager for processing, as well as to the stream filter. The stream filter will enforce the principle of limited collection. In particular, the stream filter will maintain data structures to record what attributes of which streams are currently being queried. If there is no query trying to access a certain stream, all data tuples from that stream will be dropped directly from the buffer, instead of being sent to the query processor. Furthermore, if a stream is involved in some queries but a certain attribute of the stream is not asked for by any queries, then that field will be anonymized for all the incoming tuples. This is the attribute-level limited collection. The process of limiting data collection is dynamic w.r.t. the query registration and revocation.

Finally, the rewritten queries (if needed) and partially anonymized stream data (if needed) will be send to the query processor for processing. The result will continuously returned to the query issuer for further processing.

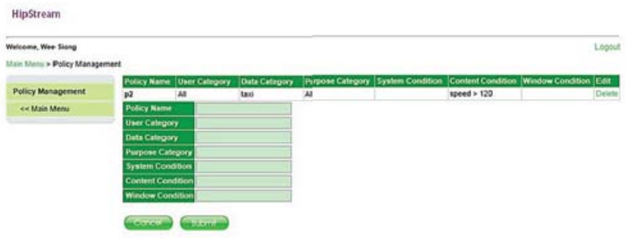
In our implementation, we use Esper [13] as the query processor, and the DataTurbine [14] for stream buffer management. The stream adapter wrap each incoming raw data tuple as an Esper object when the tuple is ready to go into the stream filter from the DataTurbine. We implement all other modules to guarantee the principles of limited collection, limited use and limited disclosure before the user data are sent to third-party service providers.

IV. SYSTEM DEMONSTRATION

In this section, we will demonstrate how our system achieves the three Hippocratic principles. We have a web portal that supports full authentication and authorization for data providers to register their stream metadata and privacy control policies, and for stream users to issue queries. In this demo, we do not show the process of stream registration, instead, we focus on the functions that are related to privacy preservation.

In the demonstration, we use the taxi trajectory and status data provided by a local taxi company. This data set is also provided to other research institutions and companies for innovative service design. Thus, as the data provider, the taxi company would like to control the access to the data for different parties. The schema of the data can be viewed from Fig. 2(c), and we would not further illustrate it as it is less important for the demonstration.

As shown in Fig. 2(a), the taxi company can specify any policy to limit the access to its data. Each policy consists of the Policy Name, User Category, Data Category, Purpose Category, and the optional fields of System Condition, Content Condition and Window Condition. The User Category, Data Category and Purpose Category state that this policy only allows particular user group to access particular data group for particular purposes. We use hierarchical model to organize users, data and query purposes, thus all the three fields are in the forms of categories. The System Condition includes the constraints on system settings, e.g., the valid period of a query in system time. The Content Condition specifies constraints to access data based on data content. The Window Condition specifies the window constraints for window join queries. More details about our policy model can be found in another report[15]. The example policy shown in Fig. 2(a) means


HipStream

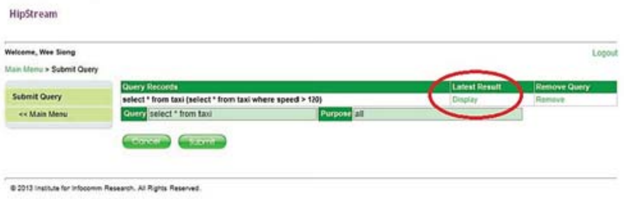
Welcome, Wee Siong

Main Menu > Policy Management

Policy Name	User Category	Data Category	Purpose Category	System Condition	Content Condition	Window Condition	Edit
p2	All	taxi	All		speed > 120		Delete

Policy Name	
User Category	
Data Category	
Purpose Category	
System Condition	
Content Condition	
Window Condition	

(a) Policy registration


HipStream

Welcome, Wee Siong

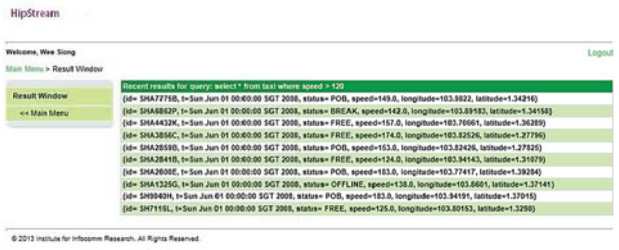
Main Menu > Submit Query

Query Records	Latest Result	Remove Query
select * from taxi (select * from taxi where speed > 120)	Display	Remove

Query: select * from taxi Purpose: all

© 2013 Institute for Infocomm Research. All Rights Reserved.

(b) Query registration


HipStream

Welcome, Wee Siong

Main Menu > Result Window

Recent results for query: select * from taxi where speed > 120

```
{id= SHA7775B, t=Sun Jun 01 00:00:00 SGT 2008, status= POB, speed=149.0, longitude=103.9822, latitude=1.34216}
{id= SHA6862P, t=Sun Jun 01 00:00:00 SGT 2008, status= BREAK, speed=142.0, longitude=103.89183, latitude=1.34158}
{id= SHA4432K, t=Sun Jun 01 00:00:00 SGT 2008, status= FREE, speed=157.0, longitude=103.70661, latitude=1.36289}
{id= SHA3856C, t=Sun Jun 01 00:00:00 SGT 2008, status= FREE, speed=174.0, longitude=103.82526, latitude=1.27796}
{id= SHA2859B, t=Sun Jun 01 00:00:00 SGT 2008, status= POB, speed=153.0, longitude=103.82426, latitude=1.27825}
{id= SHA2841B, t=Sun Jun 01 00:00:00 SGT 2008, status= FREE, speed=124.0, longitude=103.94143, latitude=1.31079}
{id= SHA2600E, t=Sun Jun 01 00:00:00 SGT 2008, status= POB, speed=183.0, longitude=103.77417, latitude=1.39284}
{id= SHA1325G, t=Sun Jun 01 00:00:00 SGT 2008, status= OFFLINE, speed=138.0, longitude=103.8601, latitude=1.37141}
{id= SH9040H, t=Sun Jun 01 00:00:00 SGT 2008, status= POB, speed=183.0, longitude=103.94191, latitude=1.37015}
{id= SH7116L, t=Sun Jun 01 00:00:00 SGT 2008, status= FREE, speed=125.0, longitude=103.80153, latitude=1.3298}
```

© 2013 Institute for Infocomm Research. All Rights Reserved.

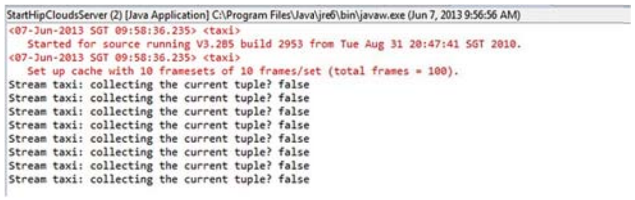
(c) Query result

Fig. 2. Policy and query registration in HipStream

that all users can access the taxi data (provided by the taxi company) for any purposes, only when the field of speed has value greater than 120. In other words, in this example the taxi company would like other parties to design services, e.g., alert to the overspeeding taxis it owns.

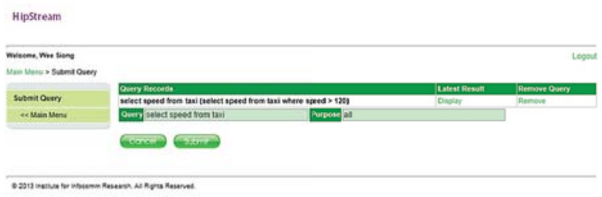
Fig. 2(b) shows the case that a party, e.g., a service designer tries to retrieve the taxi data by issuing a query of "select * from taxi". Because of the policy stating that only the record with high speed can be viewed, the query will be rewritten by being appended a condition "where speed $>$ 120", as shown in Fig. 2(b) in which the new query is within the brackets after the original query. The query result (dynamic latest 15 records) can be viewed on the web portal as shown in Fig. 2(c), and as we see all returned tuples have higher speed values than 120. All result will be streamed back to query issuer via another protocol, which is not covered in this demonstration.

Next, we will show how the system limits the data collection. We show the screenshot of the real-time backend console window after query addition and removal in Fig. 3. The initial backend data collection status when there is no query registered is shown in Fig. 3(a). We can see the system did not collect any incoming data tuple. After one query is registered, as shown in Fig. 3(b), the new backend screenshot is shown in Fig. 3(c). Note that the newly added query only queries about the attribute of speed, so in Fig. 3(c), though the system starts collecting tuples, all tuples collected only contains one value, i.e., speed value. All other attribute values



```
StartHipCloudsServer (2) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Jun 7, 2013 9:56:56 AM)
<07-Jun-2013 SGT 09:58:36.235> <taxi>
   Started for source running V3.285 build 2953 from Tue Aug 31 20:47:41 SGT 2010.
<07-Jun-2013 SGT 09:58:36.235> <taxi>
   Set up cache with 10 framesets of 10 frames/set (total frames = 100).
Stream taxi: collecting the current tuple? false
Stream taxi: collecting the current tuple? false
Stream taxi: collecting the current tuple? false
Stream taxi: collecting the current tuple? false
Stream taxi: collecting the current tuple? false
Stream taxi: collecting the current tuple? false
Stream taxi: collecting the current tuple? false
Stream taxi: collecting the current tuple? false
```

(a) Initial data collection



HipStream

Welcome, Wee Siong — Logout

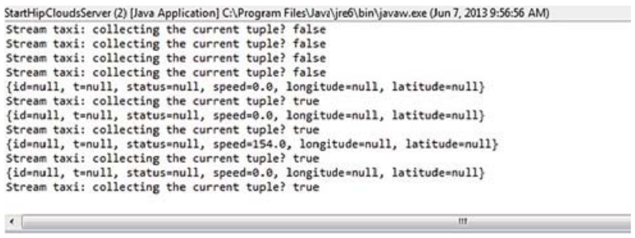
Main Menu > Submit Query

Query Records	Latest Result	Remove Query
select speed from taxi (select speed from taxi where speed > 120)	Display	Remove

Query: select speed from taxi — Purpose: all

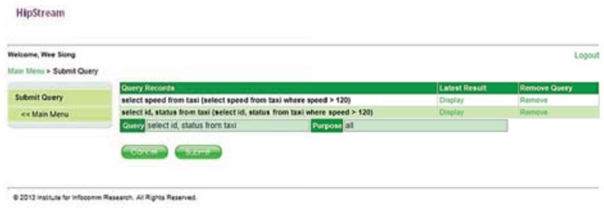
Submit Query
<< Main Menu
Cancel
Submit
© 2013 Institute for Infocomm Research. All Rights Reserved.

(b) A query added



```
StartHipCloudsServer (2) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Jun 7, 2013 9:56:56 AM)
Stream taxi: collecting the current tuple? false
Stream taxi: collecting the current tuple? false
Stream taxi: collecting the current tuple? false
Stream taxi: collecting the current tuple? false
{id=null, t=null, status=null, speed=0.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id=null, t=null, status=null, speed=0.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id=null, t=null, status=null, speed=154.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id=null, t=null, status=null, speed=0.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
```

(c) Data collection after adding a query



HipStream

Welcome, Wee Siong — Logout

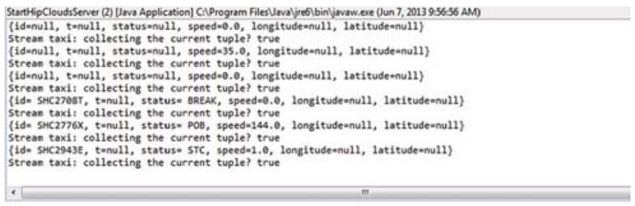
Main Menu > Submit Query

Query Records	Latest Result	Remove Query
select speed from taxi (select speed from taxi where speed > 120)	Display	Remove
select id, status from taxi (select id, status from taxi where speed > 120)	Display	Remove

Query: select id, status from taxi — Purpose: all

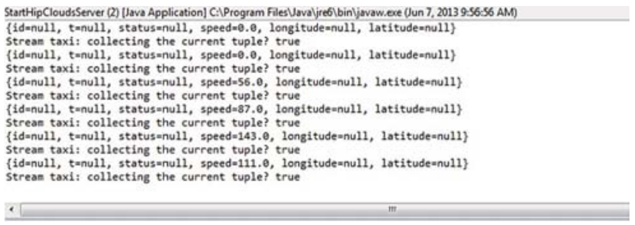
Submit Query
<< Main Menu
Cancel
Submit
© 2013 Institute for Infocomm Research. All Rights Reserved.

(d) Another query added



```
StartHipCloudsServer (2) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Jun 7, 2013 9:56:56 AM)
{id=null, t=null, status=null, speed=0.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id=null, t=null, status=null, speed=35.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id=null, t=null, status=null, speed=0.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id= SHC2708T, t=null, status= BREAK, speed=0.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id= SHC2776X, t=null, status= POB, speed=144.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id= SHC2943E, t=null, status= STC, speed=1.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
```

(e) Data collection after adding the second query



```
StartHipCloudsServer (2) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Jun 7, 2013 9:56:56 AM)
{id=null, t=null, status=null, speed=0.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id=null, t=null, status=null, speed=0.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id=null, t=null, status=null, speed=56.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id=null, t=null, status=null, speed=87.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id=null, t=null, status=null, speed=143.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
{id=null, t=null, status=null, speed=111.0, longitude=null, latitude=null}
Stream taxi: collecting the current tuple? true
```

(f) Data collection after removing the second query

Fig. 3. Backend data collection demo

are anonymized (i.e., replaced by null), because they are not asked for and should not be collected.

When we add another query, as shown in Fig. 3(d), which asks for another two attributes "id" and "status", the backend screen shows that the system starts collecting values of the two attributes, but still leaves other attributes anonymized, as shown in Fig. 3(e).

Finally, when the newly registered query is revoked, the system will no longer collected the attribute values for "id" and "status", as shown in Fig. 3(f).

V. Conclusion

In this demo paper, we present our system, HipStream to systemically enforce Hippocratic principles during data stream management. We depict the architecture of the system, and also demonstrate how the system achieve limited disclosure, limited use and limited collection. Using our system, service providers will not be able to view the full dataset from a data provider. They can only view the data that they are allowed to access, which is fully controlled by the data provider. The system itself will strictly enforce all the rules and policies.

For future work, we will customize the HipStream system for particular domains of services, to further improve the user experience.

Acknowledgment

This work was supported by the SERC of A*STAR under the Grant No. 1224200004.

References

[1] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *MobiSys*, 2003.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Hippocratic databases," in *VLDB*, 2002, pp. 143–154.

[3] M. H. Ali, M. Y. ElTabakh, and E. Bertino, "Hippocratic data streams - concepts, architectures and issues," Purdue University, Tech. Rep. 05-025, 2005.

[4] S. Babu and J. Widom, "Continuous queries over data streams," *SIGMOD Record*, vol. 30, no. 3, pp. 109–120, 2001.

[5] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, F. Reiss, and M. A. Shah, "TelegraphCQ: Continuous dataflow processing," in *SIGMOD Conference*, 2003, p. 668.

[6] StreamBase Systems Inc, "Streambase event processing platform. http://www.streambase.com/," 2009.

[7] IBM Watson Research Center, "Exploratory stream processing systems," 2009.

[8] W. Lindner and J. Meier, "Securing the borealis data stream engine," in *IDEAS*, 2006, pp. 137–147.

[9] R. V. Nehme, E. A. Rundensteiner, and E. Bertino, "A security punctuation framework for enforcing access control on streaming data," in *ICDE*, 2008, pp. 406–415.

[10] R. V. Nehme, H.-S. Lim, and E. Bertino, "FENCE: Continuous access control enforcement in dynamic data stream environments," in *ICDE*, 2010, pp. 940–943.

[11] R. V. Nehme, H.-S. Lim, E. Bertino, and E. A. Rundensteiner, "StreamShield: a stream-centric approach towards security and privacy in data stream environments," in *SIGMOD Conference*, 2009, pp. 1027–1030.

[12] B. Carminati, E. Ferrari, J. Cao, and K. L. Tan, "A framework to enforce access control over data streams," *ACM Trans. Inf. Syst. Secur.*, vol. 13, pp. 28:1–28:31, 2010.

[13] EsperTech, "http://esper.codehaus.org/," 2004.

[14] S. Tilak, P. Hubbard, M. Miller, and T. Fountain, "The ring buffer network bus (RBNB) dataturbine streaming data middleware for environmental observing systems," in *eScience*, 2007, pp. 125–133.

[15] W. S. Ng, H. Wu, W. Wu, S. Xiang, and K.-L. Tan, "Privacy preservation in streaming data collection," in *ICPADS*, 2012, pp. 810–815.