Quality of Service in Stateful Information Filters

Peter M. Fischer Donald Kossmann

Department of Computer Science
ETH Zurich
8092 Zurich, Switzerland
www.dbis.ethz.ch

ABSTRACT

Information Filters play an important role in processing streams of events, both for filtering as well as routing events based on their content. Stateful information filters like AGILE [15], Cayuga [13] and SASE [24] have gained a significant amount of attention recently. Such filters not only consider the data of a single event, but also additional state such as a sequence of previous events or a context state. Applications for Wireless Sensors and RFID data are particularly prominent examples for the need for stateful information filtering, with use cases like event correlation or sensor data affecting the routing of other events. While quality of service has been researched fairly thoroughly for networking systems and general data stream management systems, no comprehensive work exists for information filters. The goal of this work is to present QoS criteria for stateful information filters and to examine how QoS control methods established in other areas can be applied to information filters.

1. INTRODUCTION

Information filtering has seen a significant amount of interest from both academia and industry. The database community has developed matching algorithms that support expressive profile languages like XPath [14, 20, 9] and are scalable to millions of profiles [16] while maintaining high message throughput rates. Under the name of *message broker* [1] or *message queuing system* [2, 5], the industry has picked up information filters. A fairly recent and promising trend are stateful information filters such as AGILE [15], Cayuga [13] and SASE [24], which incorporate context data (for example from sensors) or event sequences (for example from RFID readers) into the matching. Information filters are often used for in-network query processing. Prominent examples are SIENA [10], or Gryphon [22].

Considering quality of service is well-established in related areas like networking, data stream management systems, and –in a very rudimentary manner– even in commercial message brokers. What is lacking, however, is a wider view on how the information filter algorithms developed in the database community can provide specific service levels, including timely delivery and error-free processing, if the available resources are limited.
This work contributes the following aspects to the area of quality of service (QoS) in information filters:

Proceedings of the 3rd International Workshop on Data Management for Sensor Networks (DMSN'06), Seoul, South Korea, 2006

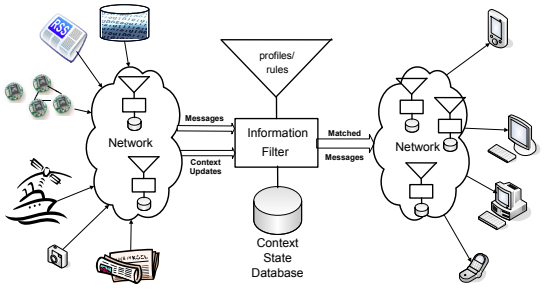


Figure 1: Information Filtering

1. it defines QoS parameters for information filters;
2. it reviews (qualitatively) the suitability of existing processing architectures and algorithms devised in the database community to support the QoS parameters;
3. it presents some results of an extensive performance study of these architectures and algorithms with regard to QoS parameters.

2. QOS OF INFORMATION FILTERS

2.1 What is an Information Filter?

An information filter (as depicted in Figure 1) connects sources and sinks of information using profiles. In many cases, information filters are distributed over the network in order to increase scalability and remove single points of failure. Parties interested in receiving information (sinks) submit a profile of their interest to the information filter, while parties interested in disseminating information (sources) send messages to the information filter. Sources are often sensors or data derived from sensors. The purpose of an information filter is therefore the matching of messages to the profiles, so that the matching messages can be sent to the relevant subscribers or routed to another information filter closer to the subscriber. A message matches a profile if it contains values for all the attributes involved in predicates of the profile and these values meet the restrictions specified in these predicates. For instance, the message [temperature=20,pressure=850,speed=8] meets the profile (temperature<22) $\wedge$ (speed>5), whereas it does not match

the profile (humidity ≤ 75). The information filter keeps track of the profiles, and in stateful information filters [15, 13, 24] additional state (named context in [15]) is used for the matching decision. The updates in the profiles (un-/re-subscribe or state change) compete with the messages for timely processing.

2.2 QoS Scenarios

Existing evaluations of information filters mostly target the cost aspect: how much is the cost (in time or CPU) to process a message or a subscription change. There are, however, many situations where cost is just one aspect among others. The usefulness (or quality) of an information filtering "service" needs to be judged by many, often competing, aspects. Most services do not have a constant load, but rather massive changes in their utilization. Provisioning those services for peak load is -under most circumstances- not possible at acceptable cost. So while fulfilling all possible requirements at low load is not a problem, a tradeoff has to be made for higher loads. Some of these requirements are "hard", therefore they must be fulfilled, while others can be treated with lower priority. The following examples show some very different, yet typical scenarios of information filter quality of service:

- **E-science**: A big scientific instrument (like the particle accelerators at CERN) sends results of a running experiment at very high rates. An information filter is used to only keep the data that is relevant for specific interests. No errors are allowed to occur, as important information about not yet discovered particles might get lost. The arrival of events does not have to be timely, however.
- **Sports results**: During a sports events like a soccer or a basketball game, the current result is transmitted when it changes (e.g. a goal/point is scored). Messages with the result can be dropped, if new results are coming in quickly (e.g. many points in the same minute).
- **Location based service**: Messages with offers are sent to a subscriber based on his/her current location. If there is a delay in processing the messages, it is actually better to filter based on the new location than filter on the old location (which might be hard to reach again).
- **Load balancing**: Incoming jobs (=messages) are placed on the server that is the best fit (lowest CPU load, enough free memory). Placing a short-running job on a system that has somewhat outdated load statistics might result in sub-optimal utilization, but the impact is limited. Updating the load statistics every microsecond will be more costly overall.

Each of these scenarios needs different QoS requirements, so in practice most of them are solved by building a specialized, ad-hoc system or massive overprovisioning. The goal of this work is to study how already existing processing algorithms for information filters can be used for those different requirements, and find out if there is one single method that would cover all the requirements.

2.3 QoS Requirements

Several related areas have established their respective notion of quality of service: In packet networking, requirements are minimum throughput, latency/delay, jitter, content errors, lost/duplicated packets and packet order. The commercial message broker systems, on the other hand, put their focus on reliability, fairness of arrival (same time for all subscribers).

Since we focus on the algorithms inside each information filters, the following requirements are the most important:

- **Latency**: Latency here is defined as the time between message creation and message arrival at the designated target: In the scenarios mentioned above, soccer, load balancing and location based services require low latency.
- **Jitter**: Jitter is the change in latency between events, more specifically between two consecutive messages. In the e-science scenario it is important to avoid overloading later stages with bursts of messages.
- **Errors**: Two types of errors may occur:
  - **False negatives**: False negatives are matching messages that are not delivered to a subscriber. In the scenarios above, the e-science example is sensitive to false negatives, since lost messages cannot be recovered.
  - **False positives**: False positives are non-matching messages that are delivered to a subscriber. False positives are again important in an e-science scenario in order to avoid flooding the subscribers with spurious data.

Besides these main requirements, there are other relevant parameters: availability, message order and content errors. For all these parameters, there exist solutions orthogonal to the processing algorithms: Availability can be ensured by techniques like replication and failover. Message order (which is given up by some processing algorithms to improve the throughput) can be restored by an extra sorting step before delivery. Content errors (i.e. the message content is destroyed) are normally not a problem of the filtering system, but more of the network. Solutions include error-correcting encoding and retransmitting.

Processing in the filter is not the only factor influencing the QoS parameters, but the transport network and the sources may also have an impact. These effects influence all the processing algorithms in the same way, thus they need not to be taken into consideration in the rest of the study.

An important aspect of QoS requirements is the granularity. With coarse-grained QoS, all messages, updates and profiles have the same requirements. With fine-grained QoS [8], certain sets of messages, updates and profiles have different QoS requirements: some profiles may require strict latency while others do not allow errors. Since this work is a study of the current state of the art, and no information filter fine-grained QoS currently exists, the focus will be on coarse-grained QoS.

3. INFORMATION FILTER TECHNIQUES

3.1 Components of an Information Filter

The right-hand side of Figure 2 gives an overview of the architecture of an information filter (following [15] and [18]). Such an information filter has four main components: (a) indexes, (b) merge, (c) postfiltering, (d) state management. Typically, there are several indexes for different kinds of predicates of the profiles. An index implements a function that gets a single message as input and returns a set of profiles that potentially match that message. Indexing to speed up the process of finding matching predicates have extensively studied in the literature [16, 14, 9, 11].

Since a profile can involve several predicates, the sets of profiles returned by each index need to be merged. Logically, the merging step carries out conjunctions and disjunctions. The result of the merging step is a set of profiles that match

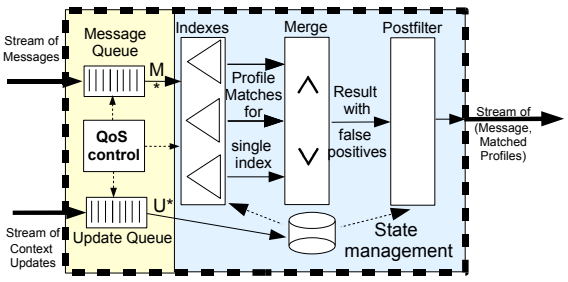


Figure 2: Information filter architecture

the message according to all predicates that are indexed. Since a profile can involve additional predicates that are not indexed, a postfilter step is necessary in order to evaluate those predicates. The state management is used to keep the state; it is used by the indexes and the postfilter stage. As an example of the use of the state management, consider a profile which determines that messages specifying a location within 500 meters of the current position of the subscriber should be received. The context management will contain the current position, and a spatial index or the postfilter will use this location information for the matching.

3.2 Extensions for QoS Control

To support the QoS parameters of latency, jitter and errors, a system needs to take control over the flow of messages and updates (similar to queuing disciplines in networking like random early detection [19]) and also control the activities of the filter. The left-hand side of Figure 2 shows queues which keep the incoming messages and updates until they are processed. The QoS policy can access these queues and also steer the filter for specific operations.

4. QOS CONTROL APPROACHES

While there are many policies to enforce QoS parameters on queues of events, we focus on three approaches that work on information filters and represent the state of the art: **Traditional processing**, **Shedding** and **Batching**.

4.1 Traditional Processing

The general idea of traditional processing is to process all messages individually in the order they arrive and *not* exert any control over the messages and updates in the queues. Existing publish/subscribe systems like SIENA [10], Gryphon [22], Tibco [1], Sun JMS [2], Oracle Streams [4], Websphere MQ [5] and Microsoft BizTalk [3] use the *traditional* approach in their processing. The published version of AGILE [15] also uses this approach, but adapts the filter component to changes in the message/update ratio [15] in order to optimize the throughput (and thus the latency). In the rest of this paper, the method without this kind of load adaptation will be called **Traditional Eager**, the method with load adaption **Traditional Agile**.

A qualitative evaluation of the traditional approach gives the following results: Since no events are discarded and the order of the events stays the same, there will be neither false positives nor false negatives. In turn, there are no guarantees on latency: If the arrival rate of events is lower than the processing capabilities of the filter, the latency will be very low. If the arrival rate exceeds the capcity, the filter will be overloaded and there will be a backlog of events. Jitter will be relatively low, since the changes in processing time between two consecutive messages are relatively small, even at the begin or at the end of a burst.

There are no control parameters for traditional processing; it will always process all events in the given order.

4.2 Load Shedding

The main idea of load shedding is to discard events when processing within the QoS bounds is not possible. This strategy is used in networking (e.g. [19]) and has also received a significant amount of interest in stream processing systems like Aurora [23], LoadStar [12] and STREAM [6].

When looking into the details of possible shedding policies, the following issues need to be considered:

- **What to shed**: Messages alone (**Shed messages**), updates alone (**Shed updates**) or both messages and updates (**Shed both**)?
- **When to shed**: How does the system detect that it needs to shed events?
- **Which specific events to shed**: Which events are taken out of the queue: the last arrived, some random events or specific events (e.g. semantic shedding [23])?

In the context of this work, we decided to use a combination of cost-based heuristics and a random shedding policy to cater for the above issues: When the filter is ready to process new events, it determines the available time from the latency bound, the current time and the arrival timestamp of the message as well as the number of outstanding updates. The filter "knows" (as set by the administrator or by self-monitoring) a (close) upper bound of the cost of processing a single message or a single update (in Section 5, 1850 μsec for a message and 80 μsec for updates were used, derived from observing the filter). From this information, the filter determines if all events can be processed. If the time is not sufficient, it randomly drops as many events needed until the bound can be reached (again based on the cost). On the issue what to shed, we evaluate all variants mentioned above.

Load shedding is the only method (within the scope of this work) that can provide hard bounds on latency, unless the wrong type of events is shed, e.g. *Shed updates* in a message burst or the overall rate is so high that even the cost of load shedding exceeds the available resources. In turn, it does not provide any error bounds. *Shed messages* leads to false negatives due to the dropped messages during an overload situation. The two other shedding methods lead to false negatives and false positives during and also after the burst, as updates are shed. Since the shed updates are not applied, the state of the filter after the burst is different compared to the traditional execution, an effect we call "error propagation". In terms of jitter, shedding is very similar to traditional processing: the processing in the filter is at the same speed, and the removal of events does not cause significant variations in queuing times.

For all shedding methods, the targeted latency bound is the main control parameter. For *Shed both*, the preference to shed messages or to shed updates can be adapted.

4.3 Batch Processing

The concept of batching is to combine several events or

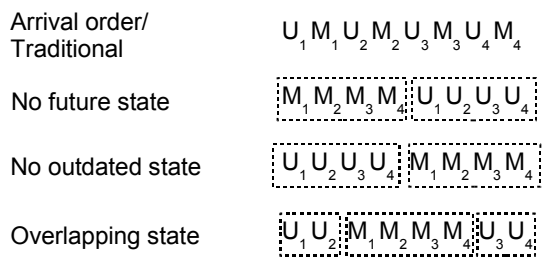


Figure 3: Batching consistency models

operations (of the same type) and do a combined processing, which is more efficient than processing them individually. In the context of databases, bulk index operations are a very typical example [7]. This technique is also widespread in networking, for examples Nagle's algorithm in TCP [21]. In the context of information filtering, a number of algorithms on batched message processing have been proposed in [18]. The results show that throughput can be improved by an order of magnitude even on moderately sized batches.

While the improved throughput can reduce the overall latency, batching can also have adverse effects on the quality of service: Collecting a sufficiently large batch to actually increase throughput can introduce additional latency. Another aspect is that the stream of incoming events is usually fairly mixed in terms of events, but batches can only be formed for messages and updates separately. So batching needs to reorder the flow of events, introducing errors from the different execution order of messages and updates.

Batching can neither give a hard bound on latency nor can it guarantee that there are no errors. However, latency under heavy load will be low since batching can speed up the processing of events. Compared to load shedding, batching has much stronger guarantees on errors: At the end of a batch, the state is the same as in traditional execution, therefore excluding error propagation. Additionally, the borders of batches form a limit on how much the state can diverge from traditional execution. Batching introduces jitter: the first message on the next batch has to wait all the time until the previous batch has been processed. The different execution orders (see Fig. 3) allow specifying an error model, e.g. location-based services would use *no outdated state* to match against the current location of a subscriber instead of his/her location when the message arrived.

In general, the batchsize is automatically adapted to the current load by taking all events that have arrived out of the queue when the filter is ready to process.

5. EXPERIMENTS

Based on the qualitative QoS analysis in the last sections, the approaches **Traditional** (both *Eager* and *Agile*), Shedding (**Shed messages**, **Shed updates** and **Shed both**) and **Batching** were evaluated. As stated in Section 2.3, **Latency**, **Jitter** and **Errors** (false negatives (FN) and false positives(FP)) were the relevant QoS requirements. These requirements become important if the information filter becomes overloaded. Many real-life workloads can cause overload: rush hours, major sports events like the soccer world cup and many more. Provisioning information systems that always can handle the peak of a burst is costly or –in most cases– impossible.

Parameter	Description	Values
MR	Message arrival rate (per second)	50 - 1500
UR	Update arrival rate (per second)	2,500 - 150,000
BD	Burst duration (seconds)	10
P	Number of profiles	500,000
Att	Number of attributes used in messages, contexts and profiles	8
AttI	Indexed attributes	2
Val	Values for messages, contexts and constants	0–10,000
UpdAtt	Percentage of updates on indexed attributes	25
UpdProf	Distribution of updates over profiles	uniform
MD	Dist. of msg values	Zipf
Lat	Latency bound	100–10,000 millisec

Table 1: Workload and tuning parameters

Due to the lack of space, only a subset of the experiments performed is shown here. For the remaining experiments, the reader is referred to [17].

5.1 Methodology and Setup

To perform the analysis, we used an existing information filter implementation ([15],[18]). This system provides a high-speed main-memory information filter for messages consisting of sets of attribute-value pairs and context-enabled profiles consisting of conjunctions or disjunctions of point or range predicates. To support the different QoS policies outlined in Section 4.1, 4.2 and 4.3, we extended this system with queues for the incoming events and the respective control policies. The system is implemented in C++ and was run on a Linux 2.6 system with a single 2.2 GHz AMD Opteron processor and 4 GB of main memory.
Since the focus of this work is on the impact of the matching engine on the different QoS parameters, we excluded the cost of I/O and message parsing from the measurements:

Each event gets a timestamp that corresponds to its designated arrival according to the workload requirements. The times to process a message, the updates between messages, a batch of messages or a batch of updates are measured using the `gettimeofday()` OS method which provides an accuracy of about 1 μsec. If processing previous events finished before the arrival time, the arrival time is counted as start for the next processing, otherwise the finishing time of the previous processing. From this starting time and the actual processing cost the new finishing time is computed.

5.2 Workload

The parameters to create the profiles, messages and updates are shown in Table 1. To simulate changes in the workload, the arrival rates for messages (MR) and updates (UR) are changed. An experiment with a combined burst of BD 10 seconds, MR 800 msgs/sec, UR 115,000 updates/sec is shown here. The other parameters follow the settings in [15] and [18]: Both messages and contexts are sets of attribute/value pairs. The number of profiles (P) is 500,000. The overall number of attributes (Att) is 8. Profiles con-

tain only conjunctions of simple predicates. Predicates, in turn, specify an epsilon environment around a constant or a context value. The selectivity of profiles on individual attributes was chosen to create a global selectivity order among the attributes. We put indexes on predicates involving the two most selective attributes ($AttI$). The values used in message attributes, context attributes and constants (Val) are of type `float` and are taken uniformly from the range $[0; 10,000]$. We quantified all values to three relevant digits in order to create a reasonably large number of different values. The distribution of updates over the attributes ($UpdAtt$) is uniform, issuing about 25 percent of the updates on attributes of indexed predicates. The distribution of updates over profiles ($UpdProf$) is also uniform. The distribution of message (MD) value was skewed following a zipf distribution. For *shedding*, the experiments using a latency bound (Lat) of 5 seconds are shown, while we also did experiments with latency bounds ranging from 100 milliseconds to 10 seconds. The results of these (and other) experiments are summarized in Section 5.4

5.3 Selected Experiment: Combined Burst

In the selected experiment the effect of a combined burst of messages and updates is shown, as many real-life scenarios might exhibit this. The workload is split in three phases:

1. **Steady State**: about 10 seconds of 150 msgs/sec, 7500 updates/sec (significantly below capacity)
2. **Combined Burst**: A burst of 10 seconds (BD) with 800 msgs/sec(MR), and 115,000 updates/sec (UR), an increase in the message rate by more than a factor of 5 and an increase in the update rate by more than a factor of 15 (each profile is updated every 5 seconds),
3. **Decay**: 50 seconds of 50 msgs/sec, 2500 updates/sec, to show the return to steady state.

The latency bound for shedding (Lat) was 5 seconds.

The latency evaluation (Figure 4) shows the following result: *Traditional* (a) reaches a maximum latency of about 19.5 seconds, returning slowly to steady state afterwards. *Shed messages* and *Shed both* (b) limit the latency to the designated latency and return to normal latency levels faster as there is no big backlog of events in the queues. *Shed Updates* is not able to achieve this latency bound, reaching about 19 seconds latency, since the messages alone cause an overload. *Batching* exhibits a sawtooth pattern with a maximum latency of 5.6 seconds. In evaluation of errors (Figure 5) *shed messages* (a,d) has a larger number of shedded messages during the burst (7500) than *shed both* (6900), but the latter has a tail of errors after the end of the burst (b,e). *Batching* has a fairly small number of errors during the batch (2609 FP, 2706 FN) and no errors after the batch (c,f).

5.4 Summary of Additional Experiments

We performed more experiments to show different types of bursts (message only, update only), the impact of setting the control parameters to the individual methods, and also longer bursts. For brevity, we do not show the results here, but they can be summarized as follows: Setting different latency bounds on shedding shows that it is effective to keep the latency under control by discarding events. The same overall results were observed, with increasing/decreasing number of errors for lower/higher latency bounds. On longer bursts, the number of errors for *shedding* and *batching* increases. More events need to be discarded to keep the bounds, and the batches get larger and thus the deviation in state. When a limit on batch time is set, overall latency becomes worse for the bounded batches compared to the unbounded batches, but overall errors are smaller.

6. CONCLUSION AND FUTURE WORK

The impact of filtering algorithms on the quality of service of information filters has –so far– been an uncharted territory. This work defines QoS requirements, reviews (qualitatively) the suitability of existing methods and does an extensive performance study.

The results of the qualitative analysis and the performance study show that there is no clear winner: *Traditional* and *Shedding* each satisfy one parameter completely (error and latency, resp.), but are weak on the missing parameter (latency and error, resp.). Batching is able to perform well on both latency and error in overload situations but cannot give a hard bound on latency and errors.

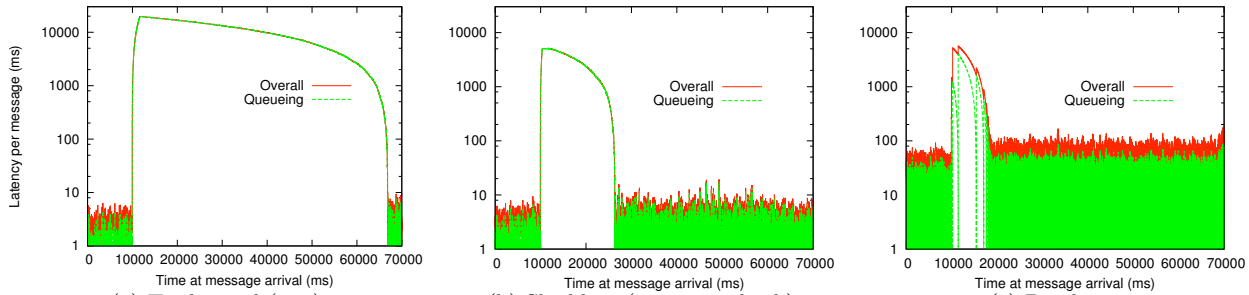
The overall processing strategies of existing information filters are very similar, so *traditional* and *load shedding* are easily applicable. Batching is currently not supported by any other information filter implementation, but we expect very similar performance characteristics if the other implementations are extended accordingly. QoS for stateless information filters is a subset of the issues discussed here, as the impact of state update need not to be considered.

A short-term goal is to extend information filters by a "frontend" that maps QoS specifications on the method that is the closest fit. The allow this, it is necassary to combine batching and shedding with a hybrid method (perhaps similar to semantic shedding [23]) that allows a gradual transition, since the other transitions are already possible.

Another direction of research that can be based on this work are models and techniques for fine-grained QoS guarantees, handling different QoS requirements on the same filter. The interesting issue with fine-grained QoS is that requirements specified on profiles mean that determining which QoS requirements apply to a messages is already as expensive as the actual filtering.

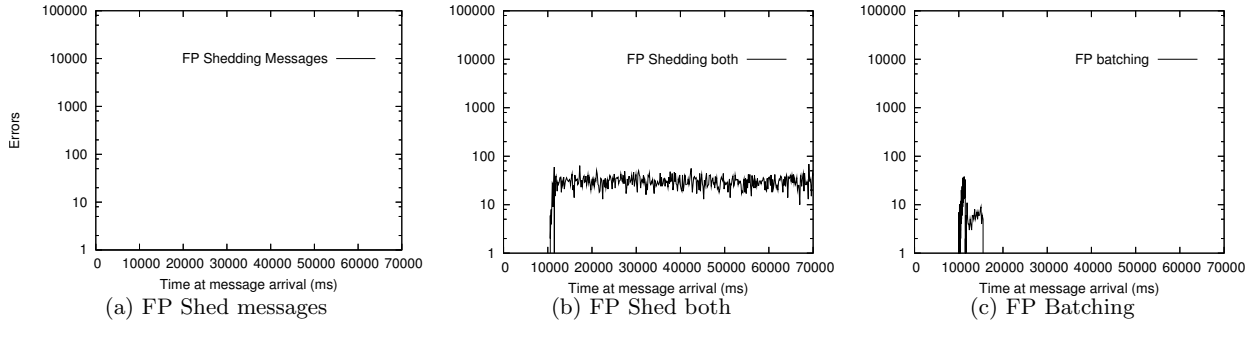
7. REFERENCES

[1] `http://www.tibco.com`.
[2] Java Message Service. `http://java.sun.com/products/jms`.
[3] Microsoft BizTalk. `http://www.microsoft.com/biztalk/`.
[4] Oracle Streams. `http://www.oracle.com/technology/products/dataint/index.html`.
[5] Websphere MQ. `http://www-306.ibm.com/software/integration/wmq/`.
[6] B. Babcock, M. Datar, and R. Motwani. Load Shedding for Aggregation Queries over Data Streams. In *ICDE*, 2004.
[7] J. Bercken and B. Seeger. An Evaluation of Generic Bulk Loading Techniques. In *VLDB*, 2001.
[8] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475: An Architecture for Differentiated Services, 1998.
[9] N. Bruno, N. Koudas, and D. Srivastava. Holistic Twig Joins: optimal XML Pattern Matching. In *SIGMOD*, 2002.
[10] A. Carzaniga, D. Rosenblum, and A. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM TOCS*, 19(3):332–383, 2001.
[11] C. Y. Chan, P. Felber, M. N. Garofalakis, and R. Rastogi. Efficient Filtering of XML Documents with XPath Expressions. In *ICDE*, 2002.



(a) Traditional (any) (b) Shedding (messages, both) (c) Batching

Figure 4: Latency for combined burst



(a) FP Shed messages (b) FP Shed both (c) FP Batching

(d) FN Shed messages (e) FN Shed both (f) FN Batching

Figure 5: Errors for combined burst

[12] Y. Chi, H. Wang, and P. S. Yu. Loadstar: Load Shedding in Data Stream Mining. In *VLDB*, 2003.

[13] A. J. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. M. White. Towards expressive publish/subscribe systems. In *EDBT*, 2006.

[14] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer. Path Sharing and Predicate Evaluation for High-Performance XML Filtering. *TODS*, 2003.

[15] J. Dittrich, P. M. Fischer, and D. Kossmann. AGILE: Adaptive Indexing for Context-Aware Information Filters. In *SIGMOD*, 2005.

[16] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. In *SIGMOD*, 2001.

[17] P. M. Fischer. *Adaptive Optimization Techniques for Context-Aware Information Filters*. PhD thesis, ETH Zurich, 2006.

[18] P. M. Fischer and D. Kossmann. Batched Processing for Information Filters. In *ICDE*, 2005.

[19] S. Floyd and V. Jacobson. Random early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[20] A. K. Gupta and D. Suciu. Stream Processing of XPath Queries with Predicates. In *SIGMOD*, 2003.

[21] J. Nagle. RFC 896: Congestion control in IP/TCP Internetworks, 1984.

[22] P. Pietzuch and S. Bhola. Congestion Control in a Reliable Scalable Message-Oriented Middleware. In *ACM/IFIP/USENIX International Middleware Conference*, 2003.

[23] N. Tatbul, U. Cetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load Shedding in a Data Stream Manager. In *VLDB*, 2003.

[24] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD*, 2006.