

**Simulation of Quality of Service (QoS) Graph-Based Load-Shedding Algorithm in
Aurora System Using CSIM**

Xiaolan Hu

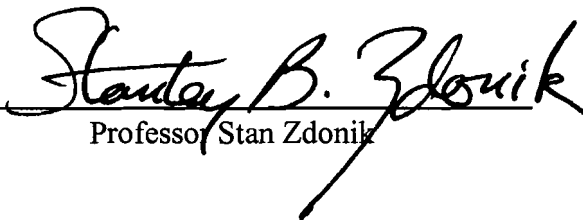
Department of Computer Science

Brown University

Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in the Department of Computer Science at Brown University

PROVIDENCE, RHODE ISLAND

May 2002



Professor Stan Zdonik

TABLE OF CONTENTS

1. Introduction	1
2. Simulation Model	1
2.1. Random Dropping	3
2.2. Drop-based Load-Shedding Algorithm	3
2.3. Value-based Load-Shedding Algorithm	3
3. Implementation	3
4. Experiments	5
4.1. Parameters	5
4.1.1. Queue Threshold	6
4.1.2. Time Slice	7
4.1.3. Maximum Drop Unit	8
4.1.4. Monitor Hold Time and Scale Factor	8
4.2. Comparing the Performances of Load-Shedding and Random-Dropping Algorithms	9
4.2.1. Single Application	9
4.2.2. Multiple Applications	13
4.2.2.1. Multiple Applications with Identical QoS Graphs	14
4.2.2.2. Multiple Applications with Different QoS Graphs	17
4.2.3. Input Rates	21
5. Improving the Load-Shedding Algorithm	24
6. Summary and Future Directions	28
References	29

LIST OF TABLES

Table 1. Drop-based Simulation for Query Network Example 1	11
Table 2. Drop-based Simulation for Query Network Example 2	12
Table 3. Drop-based Simulation for Query Network Example 3	16
Table 4. Drop-based Simulation for Query Network Example 4	20

LIST OF FIGURES

Figure 1. Architecture of Query Entry Generation Using CSIM18.	2
Figure 2. Architecture of the Query Network.	4
Figure 3. Pseudocode for Drop-based Load-Shedding Algorithm and A Simplified Description of Moving Box Method in the Load Shedder.	4
Figure 4. Query Network Example 1 with one filter box.	6
Figure 5. Drop-based Simulation in Query Network Example 1 with Varying Queue Thresholds	7
Figure 6. Drop-based Simulation in Query Network Example 1 with Varying Time Slices..	8
Figure 7. Drop-based Simulation in Query Network Example with Varying Scale Factors.	9
Figure 8. Drop-based Simulation for Query Network Example 1 with One Filter Box.	11
Figure 9. Query Network Example 2 Network.	12
Figure 10. Drop-based Simulation for Query Network Example 2 with Two Filter Boxes.	13

Figure 11. Query Network Example 3 for Multiple Applications with Identical QoS Graphs.	14
Figure 12. Drop-based Simulation for Query Network Example 3.	15
Figure 13. Query Network Example 4 for Multiple Applications with Different QoS Graphs.	17
Figure 14. Drop-based Simulation in Query Network Example 3 for Multiple Applications with Different QoS Graphs.	19
Figure 15. Different Input Rates for Drop-based Load-Shedding Simulation in Query Network Example 1 with One Filter Box	21
Figure 16. Different Input Rates for Random Dropping Simulation in Query Network Example 1 with One Filter Box.	22
Figure 17. Different Input Rates for Drop-based Load-Shedding Simulation in Query Network Example 2 with Two Filter Boxes.	22
Figure 18. Different Input Rates for Random Dropping Simulation in Query Network Example 2.	23
Figure 19. Pseudo Code for Drop-based Drop Box Removal Algorithm in Load-Shedder	24
Figure 20. Removal of Drop Boxes Improves the Throughput for Load-Shedding Algorithm in Query Network Example 1 with One Filter Box.	25
Figure 21. Query Network Example 5 Network.	26
Figure 22. Removal of Drop Boxes Improves the Throughput for Load-Shedding Algorithm in Query Network Example 5.	27

1 Introduction

Aurora is a trigger-oriented data-flow system designed to handle continuous streams of data in real time [1]. In comparison with traditional database system, it deals with large-scale trigger processing. Moreover, Aurora system is designed to meet the real time requirements for monitoring applications. We expect that a large number of applications will benefit from this original system. For example, soldiers and ammunitions wear sensors in the battlefield. Stream data from the sensors will be generated, monitored and processed continuously for commanders to make decisions. Aurora system may also speed up weather applications that track and monitor streams of data from weather sensors to alert extreme conditions.

In Aurora system, thousands of streams are passed in and processed in a query network. Overloading of the network becomes a prominent problem, which requires efficient load shedding to relieve the system. A simple strategy to deal with this problem is to specify a queue threshold for input messages arriving at each query box and to randomly drop all the messages above the threshold. Alternatively, we can employ a greedy algorithm to specify the dropping percentages (Drop-based algorithm) or filter out specific messages (Value-based algorithm) according to the two-dimensional QoS (Quality of Service) graphs provided by the application administrator [1].

The first goal of this project was to build a simple model for Aurora network, a directed graph including streams, boxes, arcs and applications that can be tracked dynamically through run-time data structures. Experiments were then conducted on various example networks, in which the behaviors of the load-shedding algorithm were observed and recorded. The immediate goal of these experiments was to identify the parameters in the networks that could influence the performance of the algorithms. We then compared the performances of the load-shedding algorithm with random dropping algorithm under fair parameters. The results may provide valuable suggestions for improvements and modifications of the algorithm in the real system.

This report documents simulation models for a simplified Aurora network using CSIM. Simulation models for random dropping and drop-based load-shedding algorithms are proposed (section 2) and implemented¹ (section 3). The implementation of value-based load-shedding algorithm will be completed in the future. Experiments were conducted to investigate the effects of various parameters on the load-shedding algorithm (section 4). We further improved the algorithm by dynamically adding and removing boxes according to the output delay time units (section 5). Conclusions and future directions will be discussed in Section 6.

2 Simulation Model

We chose CSIM18 for simulating query network and load shedder in Aurora system. CSIM18 is a process-oriented discrete-event simulation library, including classes, functions and procedures that are necessary to implement complex systems [7,8]. CSIM provides a simulation clock that allows us to track the processes dynamically. CSIM has been used to model a large number of systems, including communication, computer, transportation, microprocessors and fault tolerant systems [2,5,8]. Recently, CSIM has been used to simulate computer networks [3].

¹ Designs and implementations were collaborated with Nesime Tatbul.

To simulate the performances of QoS graph-based load-shedder algorithms in real-time, we used CSIM18 to build a simplified version of query network model (Fig. 1). In this model, input **stream** sources are generated with specified rates and processed through a directed graph of operations (e.g. **query boxes**) and passed on to the **applications**. A typical CSIM model includes both processes that mimic the active entities of the model system and structures that represent the resources in the model system [6]. We simulated query boxes as processes and **scheduler** as an active facility. A round-robin function was selected for the scheduler, in which case the “active” query boxes continuously process input streams and generate output streams to next boxes or applications. A query box is inactive if there is no message waiting to be processed. Once a new message comes in, a new query process for the box is generated and remains active until all the messages in the queue are processed. A number of data structures were implemented to track the statuses of data streams in the dynamic network. In addition, we implemented an independent process named “**monitor**”. It wakes up and inspects the query network. If the network is overloaded, the monitor activates the load-shedder.

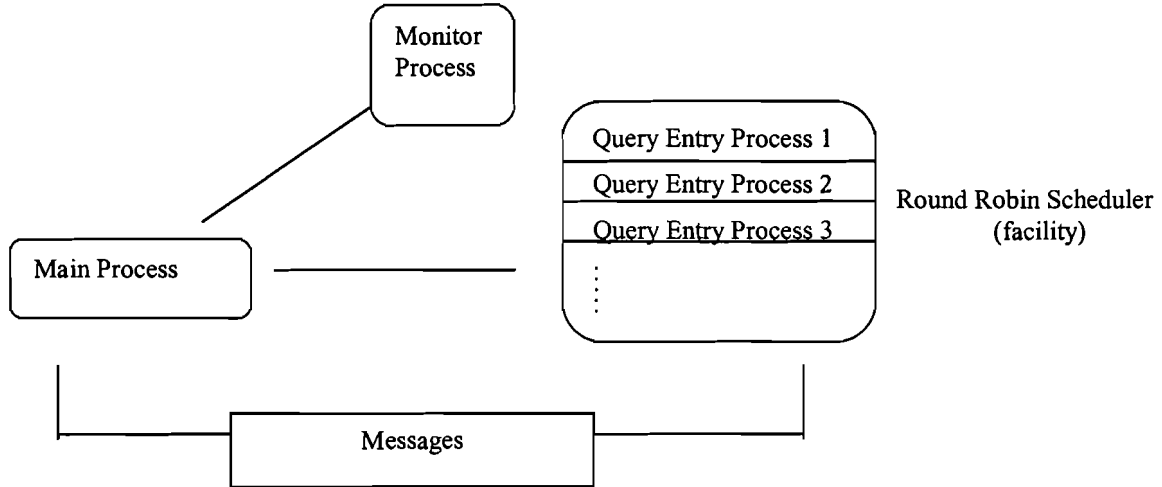


Figure 1. Architecture of query entry generation using CSIM18.

Incoming streams are generated at specific rates by the main process, which holds between stream generations. Stream data are passed into the Network as messages. Each messages has a time stamp, a stream schema, total wait time and total process time. Each query entry represents a Box operation for incoming messages. All the query entries share a facility (round robin scheduler). Monitor is simulated as an independent process, which holds for a certain period, examines the outputs and triggers the load shedder if necessary.

In the first version, we only considered four types of operator boxes: **filter box** which selects for data streams satisfying specified conditions, **map box** which applies functions to each element of a data stream and **merge box** which takes two input streams and combines them after some union operations and **drop box** that removes messages from the stream. In drop-based load-shedding algorithm, we only considered cost and selectivity. The operation semantics for these boxes were also coded for implementing value-based load shedding algorithm in the future.

2.1 Random dropping

We simulated the random dropping algorithm by specifying a **Queue_Threshold** for each box in the network. If the size of the message queue exceeds the threshold, additional messages will be dropped. We expect such threshold to be necessary to ensure that queue lengths will not grow infinitely and that sufficient computational resources are available in the Aurora system.

2.2 Drop-based load shedding

To simulate drop-based load-shedder in the query network, stream values are ignored for they have no effects on the algorithm performance. Each box is assigned a **cost** value, representing the processing time for each message and a **selectivity** value ranging from 0 to 1. Selectivity value specifies percentage of messages that passes through. For a filter box, selectivity is usually less than 1 since messages that fail the filter condition will be dropped. For a map box or merge box without duplicate elimination, selectivity equals 1.

In a running query network, monitor wakes up every **Mon_Hold_Time** and detects applications that are beyond their **delay thresholds**, specified in their **delay-based QoS graph**. Monitor then activates the load shedder algorithm for those applications. One or more applications are chosen based on their **message-based QoS graphs**. The minimum slope is chosen so that reducing the workloads will sacrifice the least utility. Drop boxes are inserted and moved forward to the upstream network where a split point is detected. Multiple drop boxes can be combined and moved further if possible. Meanwhile, the monitor dynamically traces the new delay values after load shedding. If the system is still overloaded and the delay time value is not sufficient to reach the required utility, this process will be repeated until goals are met.

2.3 Value-based load shedding

The load shedding strategy described above reduces the workloads by randomly dropping messages at various points in the network. The approach will improve the system in terms of delay-based utility. However, it fails to consider the effects on utilities based on the output value accuracy since all the dropped messages are randomly picked.

City Simulator from IBM was chosen as our data source generator [4]. It is a scalable model system, which creates dynamic spatial data simulating the positions of people in a three-dimensional model city. The output of the simulation is produced as a text file, which contains a person ID, a time stamp and x, y, z coordinates. We imported the text file as the input streams through our **CitiSimPorter** into the simulation network. The stream data were passed through query boxes with specified data operations. Value-based load shedding algorithm will be implemented in the future.

3 Implementations

To implement the simulation models described in the previous section, a query network was built. This network includes streams, boxes, applications and implicit arcs that connect individual components into a directed graph (Fig. 2). Drop-based load shedding algorithm was

implemented according to the pseudo-code shown in Fig. 3. The City Simulator generates stream data, which will be imported by CitySimPorter in value-based load-shedding algorithm.

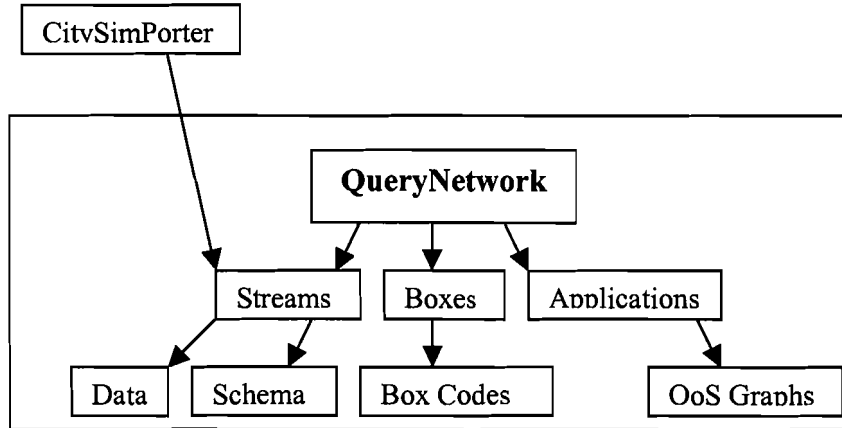


Figure 2. Architecture of the query network.
Major components (classes) are shown in squares.

$A = [A_1, A_2, A_3, \dots, A_n]$; A_i : application i , $i \in [1, n]$.
 $Q = [Q_1, Q_2, Q_3, \dots, Q_n]$; Q_i : message-based quality of service graph for application i , $i \in [1, n]$.

ShedLoad (QueryNetwork* net).
 Min_Slope $\leftarrow 0$;
 Min_Count $\leftarrow 0$;
First we identify minimum slope on message-based QoS graphs and count how many applications have it.
 for $i \leftarrow 1$ to n
 if $A_i \rightarrow \text{delay_flag} == 1$
 slope \leftarrow (slope of Q_i for current cursor)
 if slope < Min_Slope
 Min_Slope \leftarrow slope
 Min_Count $\leftarrow 1$
 else if slope == Min_Slope
 Min_Count \leftarrow Min_Count + 1
If Min_Slope == 0, all the messages will be dropped.
 if Min_Slope == 0 and Min_Count > 1
 for $i \leftarrow 1$ to n
 if $A_i \rightarrow \text{delay_flag} == 1$
 slope \leftarrow (slope of Q_i for current cursor)
 if slope == Min_Slope
 $j \leftarrow 0$
 do
 slope \leftarrow (slope of Q_i for current cursor - j)
 $j \leftarrow j + 1$
 while slope == Min_Slope and (current cursor - j) > 0
 $B \leftarrow$ create new drop box
 selectivity \leftarrow (current cursor - j) / current cursor
 id \leftarrow id of A_i
 move box (id, B, id).
Otherwise, we drop at most MAX_DROP_UNIT percent of messages when Min_Slope is less than 0.
 if Min_Slope < 0 and Min_Count > 1
 for $i \leftarrow 1$ to n
 if $A_i \rightarrow \text{delay_flag} == 1$
 slope \leftarrow (slope of Q_i for current cursor)
 if slope == Min_Slope
 $j \leftarrow 0$
 do
 slope \leftarrow (slope of Q_i for current cursor - j)
 $j \leftarrow j + 1$
 while slope == Min_Slope and (current cursor - j) > 0
 and $j < 100 * (\text{MAX_DROP_UNIT} / \text{Min_count})$
 $B \leftarrow$ create new drop box
 selectivity \leftarrow (current cursor - j) / current cursor
 id \leftarrow id of A_i
 move box (id, B, id).


```

Move-Box ( Cur_id, B, Prev_id)
  if Cur_id → type == DATA_STREAM
    if next object is a query box
      if it is a drop box, attached the new drop box B with the existing drop box
      else insert the drop box B
    else if the next object is a APPLICATION
      insert the drop box B
  else if Cur_id → type == APPLICATION
    Move-Box (Cur_id → Previous ID, B, Cur_id)
  else if Cur_id → type == QUERY_BOX
    if current_box is at a splitting point
      if drop boxes are present in all branches after the current box
        combine all the drop boxes and move forward
      else
        if current_box → type != MERGE and message queue are above the threshold
          if current_box → type == DROP
            the selectivity of the existing drop box ← ((current cursor-j) / current cursor)
                                                    * old selectivity
          else
            insert box B
        else if current_box → type != MERGE and message queue are below the threshold
          Move-Box (Cur_id → Previous ID, B, Cur_id)
        else if current_box → type == MERGE
          if can not move drop due to large queue 1
            insert B
          else
            Move-Box (Cur_id → Previous ID1, B, Cur_id)
            if can not move drop due to large queue 2
              insert B
          else
            Move-Box (Cur_id → Previous ID2, B, Cur_id)
    else
      if current_box → type != MERGE and message queue are above the threshold
        if current_box → type == DROP
          the selectivity of the existing drop box ← ((current cursor-j) / current cursor)
                                                    * old selectivity
        else
          insert box B
      else if current_box → type != MERGE and message queue are below the threshold
        Move-Box (Cur_id → Previous ID, B, Cur_id)
      else if current_box → type == MERGE
        if can not move drop due to large queue 1
          insert B
        else
          Move-Box (Cur_id → Previous ID1, B, Cur_id)
          if can not move drop due to large queue 2
            insert B
        else
          Move-Box (Cur_id → Previous ID2, B, Cur_id)

```

Figure 3. Pseudocode for Drop-based Load-Shedding Algorithm and a simplified description of moving box method in the load shedder.

4 Experiments

4.1 Parameters

A number of parameters in our simulation model could potentially influence the performance of the load shedder, including **Queue_Threshold**, **Maximum Drop Unit**, **Time Slice**, **Monitor Hold Time** and **Scale Factor**. To observe the behavior of the algorithms under various parameters, we first tested drop-based load-shedding algorithm and random dropping algorithm in a simple network with one filter box (Fig. 4).

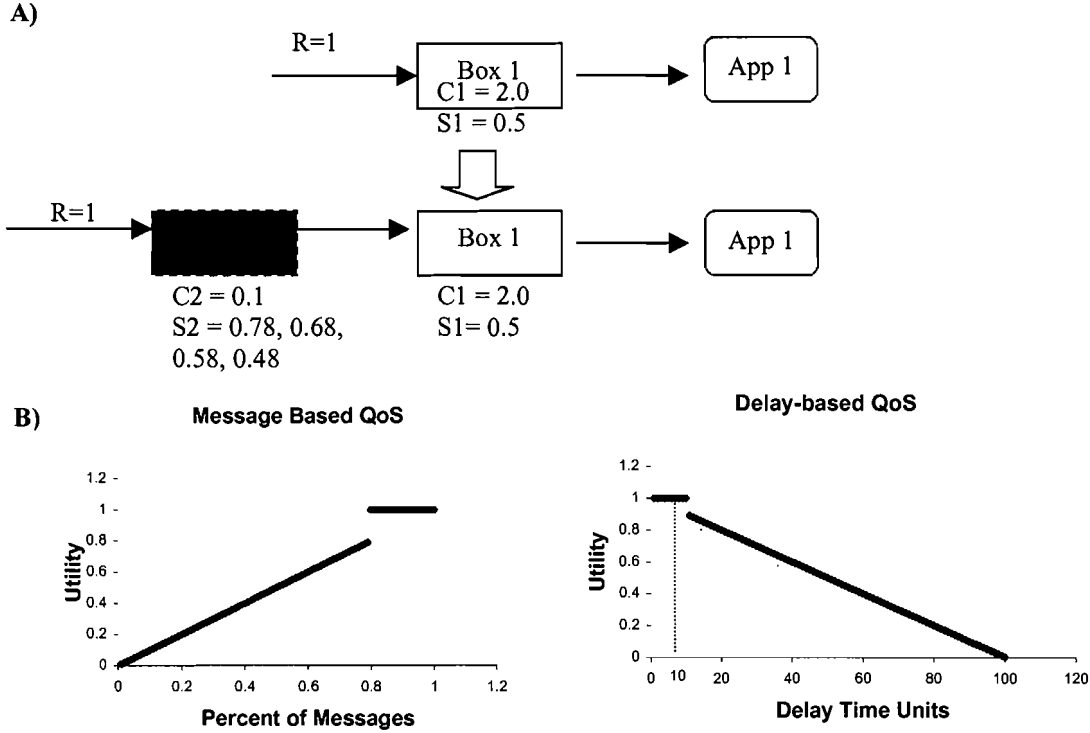


Figure 4. Query Network Example 1 with one filter box.

A). Original network for Query network example 1 and the network after load shedding.

B) QoS (quality of services) graphs for the application (App).

R: input stream rate; C: box cost; S: box selectivity.

4.1.1 Queue Threshold

Queue Threshold (QT) specifies the maximum number of messages in the queue for each query box. A big queue threshold will lead to possible exhausting of the computer resources and overloading of the system. On the other hand, unnecessary loss of messages will occur if the queue threshold is too low. To be fair, we tested and compared the performance of the load-shedding algorithm and random dropping algorithm under the same reasonable queue threshold value.

For random dropping algorithm, best performance is obtained when QT equals 5. Since all the outputs can pass through the network simply by random dropping when QT is 5, the load shedder is not activated and the two algorithms share the same graph (red and green). As QT increases from 5 to 15, it takes longer for the network to reach its stable state when delay time periods and corresponding utilities are constant. Similar tendency was observed for load shedder and random dropping algorithms.

It is evident from the graph that more messages are passing through but the final constant value for the utility decreases when QT increases from 5 to 15 for random dropping algorithm (Figure 5). This is because the messages are delayed more in the queue when QT increases. In contrast, the load-shedding algorithm always manages to reach the targeted utility value of 1 for

the outputs although it may take longer period to do so. In addition, the total number of outputs slightly fluctuates with rising QT values.

In query network example 1 which includes one filter box, a slightly more messages are dropped in the load-shedding algorithm compared with random dropping at the same QT. However, load-shedding algorithm succeeds in reducing the workload in the network so that final outputs can be processed within the delay threshold. Theoretically, modifying the queue threshold for each box could reduce the workloads so that every output will have delay utility equal to one. This strategy seems to be impossible in a complicate network where different combinations of queue thresholds for boxes grow exponentially.

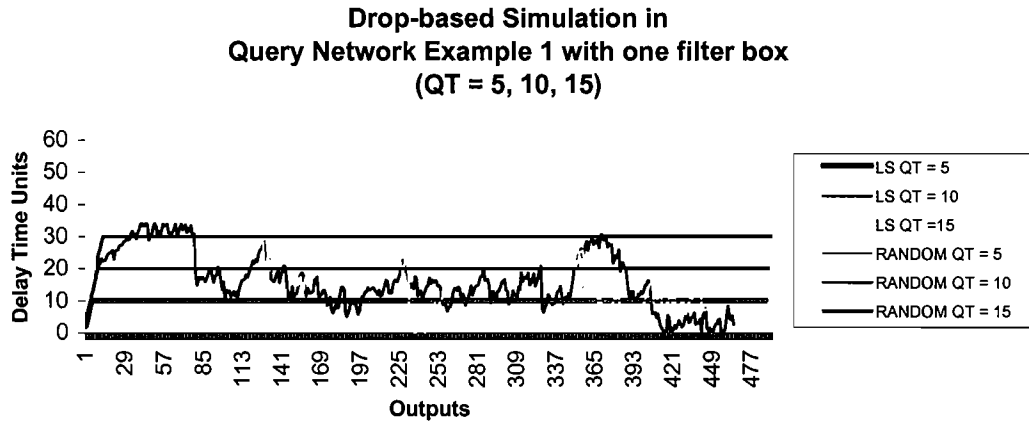


Figure 5. Drop-based Simulation in Query Network Example 1 with Varying Queue Thresholds.

Simulation is conducted in the following parameters. Input Streams: 2000;
Max_Drop_Unit = 0.1; Time_Slice = 1.0; Mon_Hold_Time = 10;
Scale_Factor = 5. LS: Load-shedding algorithm; QT: Queue_threshold;
RANDOM: random dropping algorithm.

4.1.2 Time Slice

Time slice affects the round-robin scheduler in CSIM, which defines the basic processing time unit for each process in the scheduler. This parameter is important for load-shedding algorithm since drop boxes with small costs are added and time slice will influence the sequence of the processes in the queue for the scheduler (Figure 6). Intuitively, time slice should be close to the cost of the drop box to ensure that drop boxes are processed without waiting long in the scheduler. However, a time slice value favoring drop boxes will also result in extra loss of messages. In addition, time slice should be less than or equal to the maximum cost of all the boxes in the network so that time will not be wasted in every slice. Since scheduling is a dynamic process, different time slice values have to be tuned before evaluating the algorithm performance. In the following experiments, time slice was tuned by testing values between the cost of a drop box and the maximum cost of a box in the network.

Random dropping algorithm is not sensitive to time slice for query network example 1 (shown in Fig. 4) since only one box is present. When multiple boxes are present with different

cost values, we expect that some time slice values will change the performance for the random dropping algorithm.

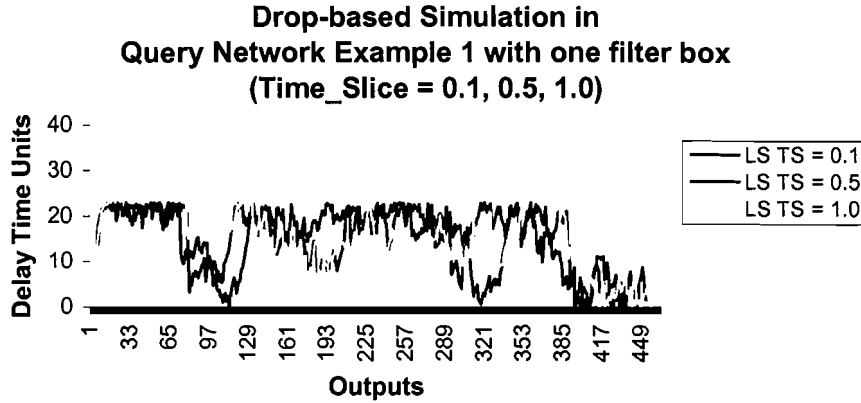


Figure 6. Drop-based Simulation in Query Network Example 1 with Varying Time Slices. Simulation is conducted in the following parameters. Input Streams: 2000; Max_Drop_Unit = 0.1; Queue_Threshold = 10; Mon_Hole_Time = 10; Scale_Factor = 5. LS: Load-shedding algorithm; TS: Time_Slice.

4.1.3. Maximum drop unit

Maximum drop unit between 0 and 1 is present in the load-shedder and determines the dropping rate. We chose the target application based on the slopes in value-based QoS graphs. A maximum drop unit is set so that we do not drop too much messages unnecessarily. An appropriate maximum drop unit has to be chosen so that workload can be reduced below the threshold while the number of lost messages is minimum. An optimal value for this parameter was chosen through testing multiple values.

4.1.4. Monitor Hold Time and Scale factor

After additions of drop boxes, the network has to be recovered for a time period before the load-shedder starts another round of load shedding. This time interval is determined by two parameters: the Mon_Hold_Time and the scale factor.

Mon_Hold_Time controls the basic unit of interval the load-shedder sleeps in between. Since addition of the drop box will take time to empty the existing queues in the network, the sleeping time for the load shedder will increase over time for the network to recover. This is achieved by multiplying a scaling factor with Mon_Hold_Time.

In the following figure (Fig. 7), when the Scale_Factor equals one, the network performs poorly with only 26 messages being processed, compared with 401 messages when the scaling factor is 2 and 464 messages when the factor is 5. A big scaling factor like 10 fails to reach utility 1 in our simulation time period, since it will take much longer for the load-shedder to

reduce the delay time for the outputs. The ideal scaling factor has to be dynamically adjusted in the real Aurora network so that the delay time units can be controlled below the threshold while minimum unnecessary messages are dropped.

It is interesting to note that monitor hold time, scale factor and maximum drop unit can all influence the dropping rate in the load-shedding algorithm. When maximum drop unit is big, we can adjust a longer monitor hold time and scale factor to achieve similar results.

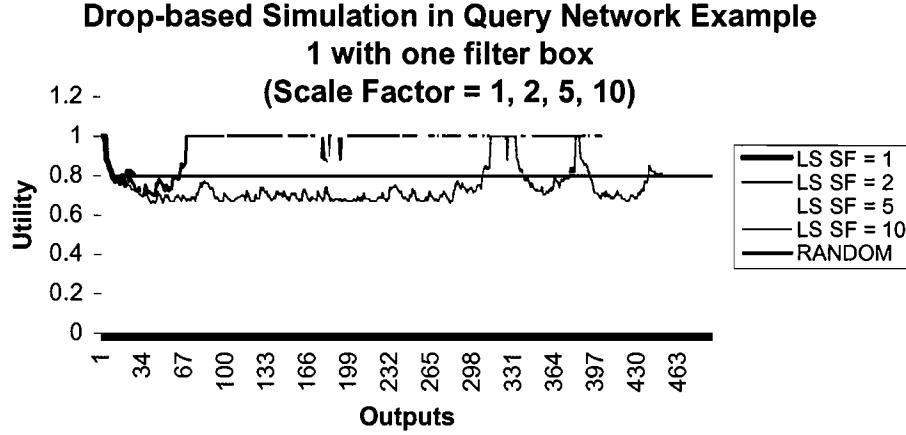


Figure 7. Drop-based Simulation in Query Network Example with Varying Scale Factors.

Simulation is conducted in the following parameters. Input Streams: 2000;
Max_Drop_Unit = 0.1; Queue_Threshold = 10; Mon_Hold_Time = 10.
LS: Load-shedding algorithm; SF: Scaling_Factor;
RANDOM: random dropping algorithm with QT = 10.

4.2 Comparing the performances of load-shedding and random-dropping algorithms

4.2.1 Single Application

Parameters mentioned in the previous section were tuned and optimized for the given networks (Fig. 4 and Fig. 9) with a single application. The running parameters are specified in the legends of Figure 8 and Figure 10. Simulation is ended when all of the input streams are either dropped or reached the application.

In Query network example 1 with one filter box (shown in Fig. 4), a drop box is added before Box 1 with cost 0.1 and selectivity 0.48 after load-shedding (S decreased from 0.78, 0.68, 0.58 to 0.48) (Fig. 4A). In Query network example 2 with two filter boxes, a drop box with selectivity 0.38 is inserted in front of Box 1 after load-shedding (S decreased from 0.78, 0.68, 0.58 to 0.48) and Fig. 9A). Theoretically, since the input rate is 1 and the cost of each box equals 2, a drop box with cost 0 and selectivity 0.5 will be able to reduce the workload so that every message can pass through the network without delay. A selectivity value of 0.48 was chosen in the algorithm. This is close to the ideal value 0.5 in Query network example 1. The selectivity value for the drop box in query network example 2 is much smaller than the ideal value since more processes stay in the queue of the scheduler and the actual waiting time for each box is longer.

Three measures were taken into account to compare the performances of the load-shedder algorithm and the random dropping method. First, we compared the delay time units for each output stream. This value is a direct measure of how well the algorithms reduces the total waiting period for each output along the network pathway. Utility is also calculated according to the delay-based QoS graph. Finally we recorded the average output rate (throughput) and calculated utility based on the message-based QoS graph.

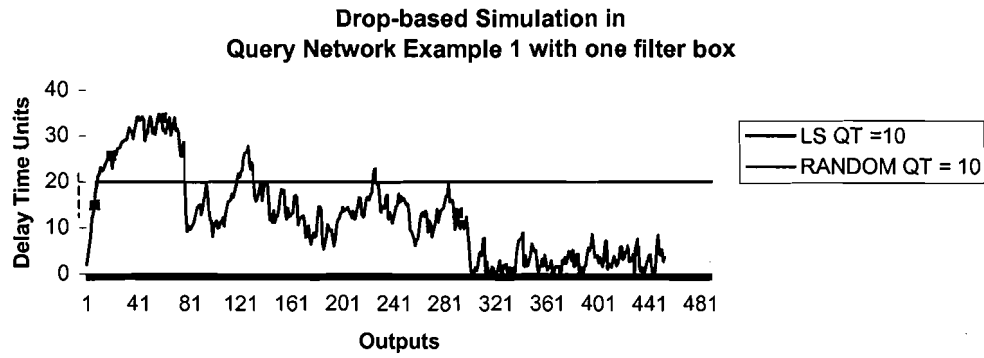
Our results indicated that the drop-based load-shedding algorithm successfully reduces the delay time units below the delay threshold whereas the random dropping algorithm reached a constant value dependent on the specified queue threshold (Fig. 8A, Fig. 10A). As a trade off, more stream messages may be dropped before they reached the destination under the load-shedding algorithm according to message-based QoS graph in Query network example 1 and Query network example 2 (Compare the throughput for random dropping and load-shedding algorithm in Table 1 and Table 2). In both cases, fewer messages will exceed the queue threshold after the appropriate drop box is applied. Meanwhile, more messages will be lost by applying the drop box. In load shedding algorithm, the total number of messages reaching the application is therefore dependent on the additional loss of messages dropped by the new drop box and the gain of messages evading the random dropping by queue threshold. Nevertheless, parameters in the load shedder can be tuned so that the loss of message-based utility, if any, is minimal.

Accordingly, the utilities of the output messages calculated based on the QoS graph eventually increases to 1 with the load-shedding algorithm (Fig. 8B and Fig. 10B). Although the values fluctuate before the final drop selectivity is applied. This is due to the fact that the previous drop boxes temporarily release the network before it becomes overloaded again. The random dropping algorithm decreases the workloads initially and remains constant after the initial stage. With the same queue threshold with the load-shedding algorithm, it fails to reach utility 1 in both examples (Fig. 8B and Fig. 10B).

For each specified network, there is a theoretical maximum throughput that the streams could reach. Although load-shedding algorithm reduces the overall delay time for each output, the throughput values fluctuate over the whole time course. Since the number of streams dropped by a filter box is defined by a random function, it will reach the specified value only in an extended time course but fluctuate in a small time window. Therefore we only calculate the overall throughput and percent of outputs during the simulations course (Table 1 and Table 2). For query network example 1, the actual throughput 0.23 by load-shedding algorithm is close to the theoretical value 0.25 (Table 1). In query network example 2, the theoretical throughput equals 0.045 (Table 2). The actual throughput for query network example 2 is 0.034. This effect may be a result of more processes in the round robin scheduler and additional waiting time (depending on the time slice) required in the queue for each process.

In summary, load-shedding algorithm attempts to minimize the loss of message – based utility while it efficiently controls the delay time for messages below the threshold, which is defined by the delay-based QoS graph.

A)



B)

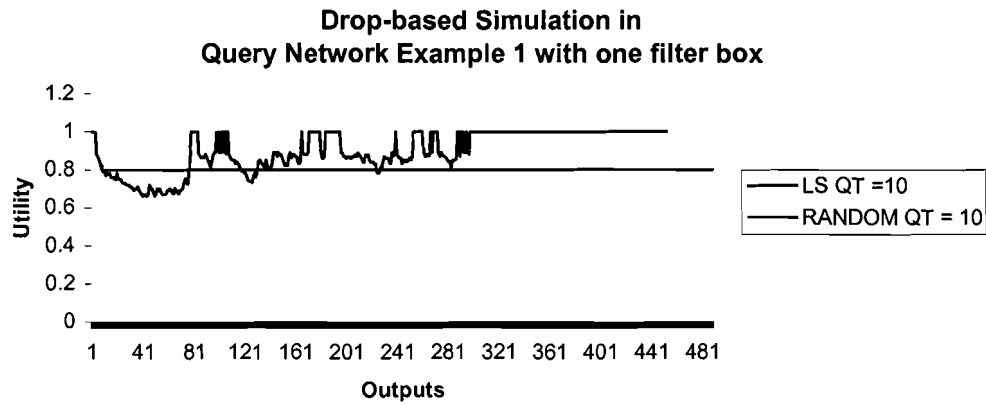


Figure 8. Drop-based Simulation for Query network example 1 with one filter box.

Simulation is conducted in the following parameters. Input Streams: 2000; Max_Drop_Unit = 0.1; Time_Slice = 1.0; Mon_Hold_Time = 10; Queue_threshold (QT) = 10; Scale_Factor = 4.5.

A) Delay-outputs and B) Utility-outputs graphs for Query network example 1.

LS: Load-shedding algorithm; QT: Queue_threshold;

RANDOM: random dropping algorithm.

Table 1. Drop-based Simulation for Query network example 1

	Load-Shedding QT = 10	RANDOM
Total Outputs	453	489
End Simulation Time	2003.7	2015
Throughput	0.226	0.243
Theoretical Outputs	1000	1000
%Messages	45.3%	48.9%
Utility (Message-based QoS)	0.45	0.49

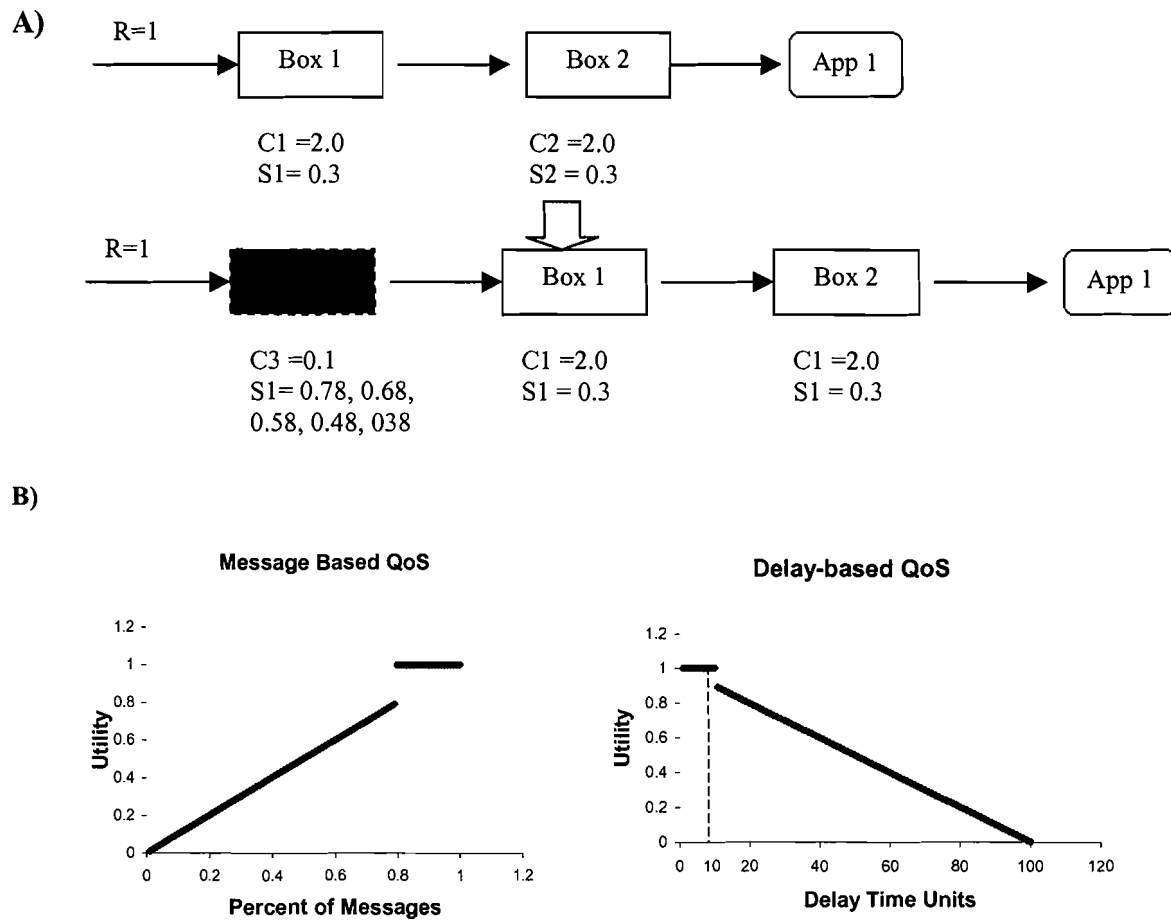
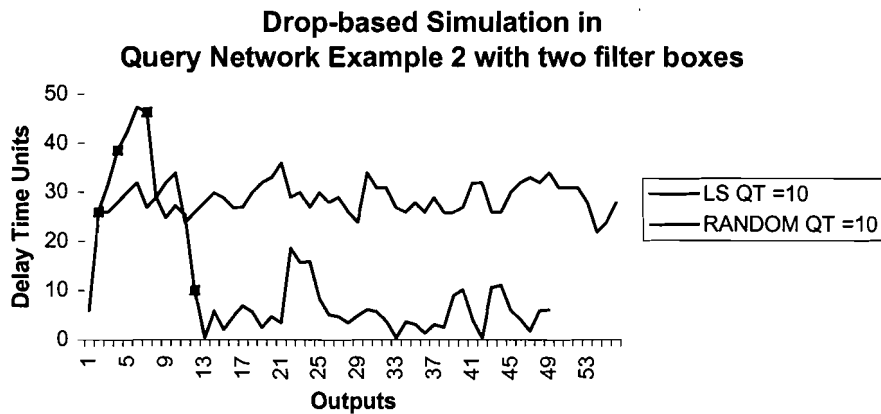


Figure 9. Query network example 2 Network. A). Original network for Query network example 2 and the network after load shedding. B) QoS (quality of services) graphs for the application (App1). R: input stream rate; C: box cost; S: box selectivity.

Table 2. Drop-based Simulation for Query network example 2

	Load-Shedding QT = 10	RANDOM
Total Outputs	49	56
End Simulation Time	2000.1	2011
Throughput	0.024	0.028
Theoretical Outputs	180	180
%Messages	27.2%	31.1%
Utility (Message-based QoS)	0.27	0.31

A)



B)

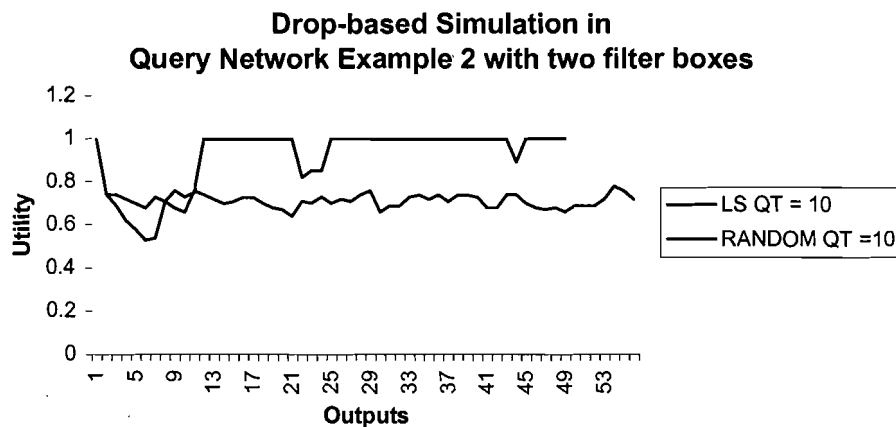


Figure 10. Drop-based Simulation for Query network example 2 with two filter boxes.

Simulation is conducted in the following parameters. Input Streams: 2000;

Max_Drop_Unit = 0.1; Time_Slice = 1.0; Mon_Hold_Time = 10;

Queue_threshold (QT) = 10; Scale_Factor = 2.

A) Delay-outputs and B) Utility-outputs graphs for Query network example 2.

LS: Load-shedding algorithm; QT: Queue_threshold;

RANDOM: random dropping algorithm.

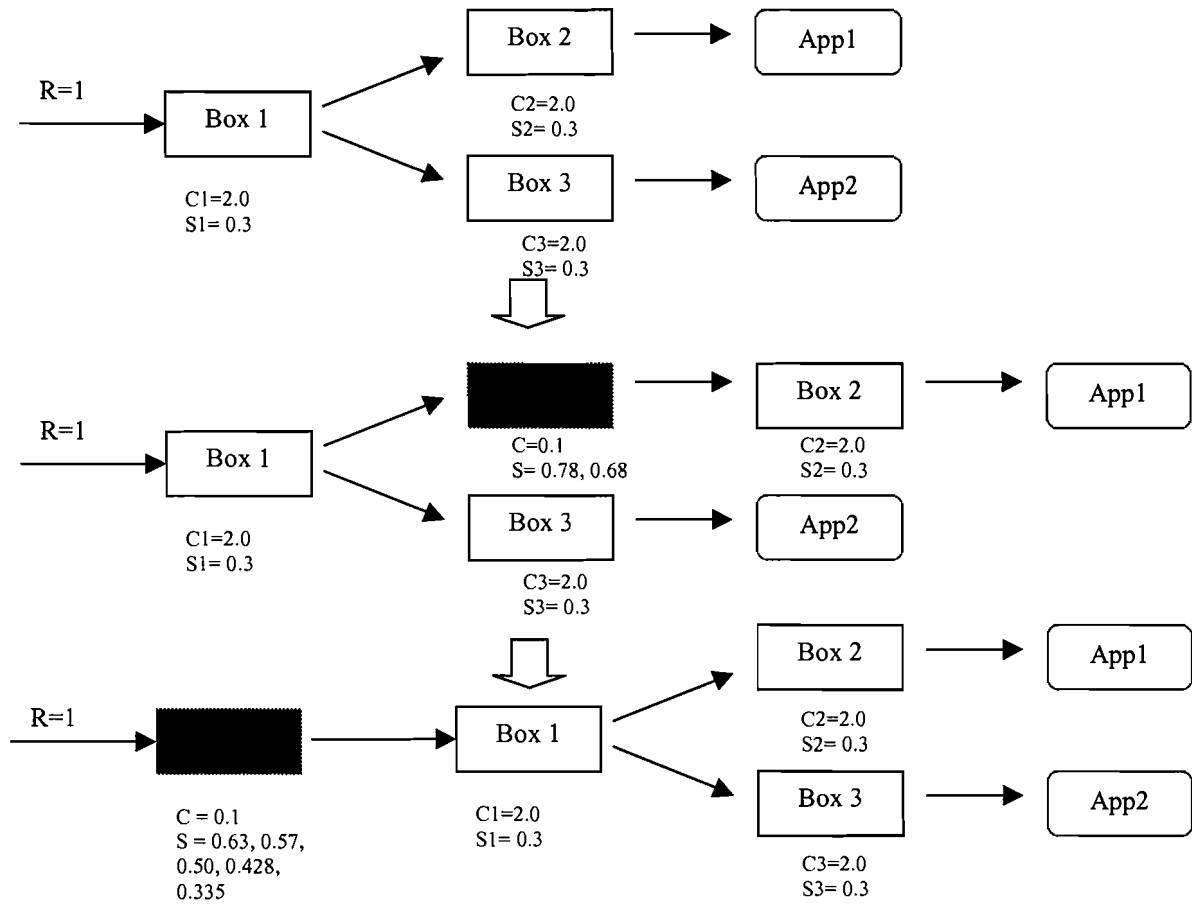
4.2.2 Multiple Applications

In the real Aurora system, it will be a common scenario when multiple applications are connected with the network. Since different applications may specify the same or different QoS graphs, it is pivotal for the load-shedding algorithm to be fair to multiple applications when no priority is specified.

4.2.2.1 Multiple Applications with Identical QoS Graphs

Parameters were tuned and optimized for the following network with two applications (Fig. 11) having identical QoS graphs. The running parameters are specified in the legends of Figure 12. Simulation ends when all of the input streams are either dropped or reached the application.

A)



B)

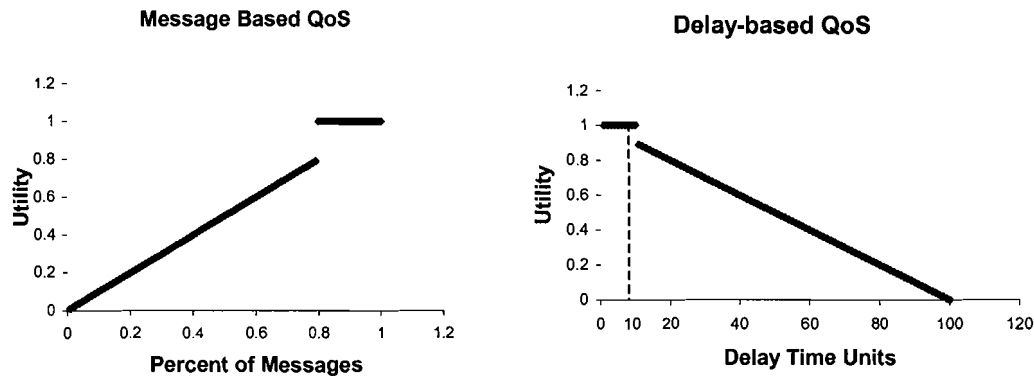


Figure 11. Query network example 3 for multiple applications with identical QoS graphs.

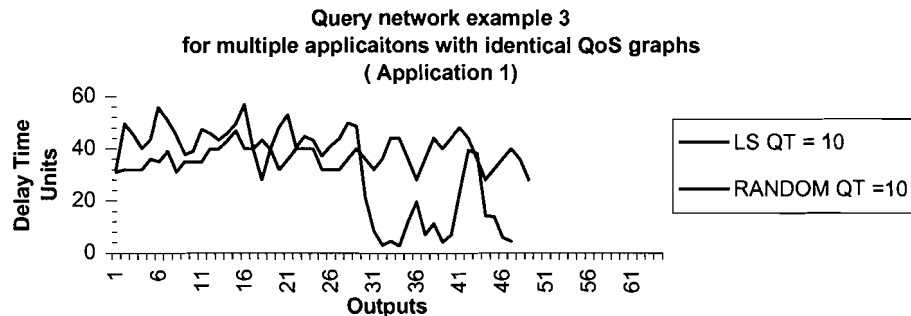
A). Original network for Query network example 1 and the network after load shedding.

B). QoS (quality of services) graphs for the applications (App 1 and App 2).

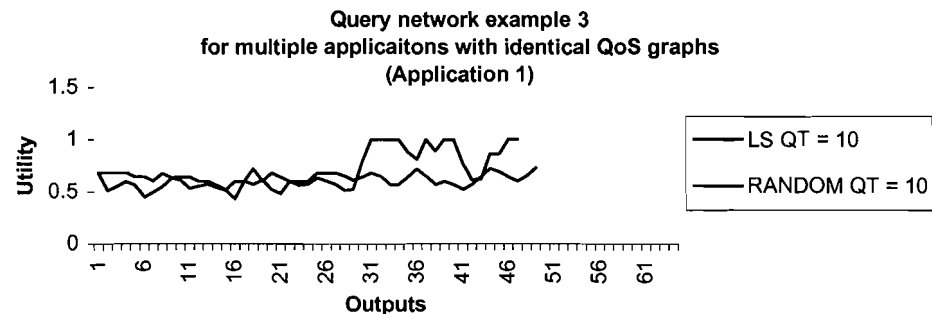
R: input stream rate; C: box cost; S: box selectivity.

In query network example 3, a drop box is first generated for application 1 before Box 2 with cost 0.1 and selectivity 0.78 after load-shedding (S decreases from 0.78 to 0.68) (Fig. 11A). Later on, when the outputs for application 2 exceed the delay threshold, a new drop box is generated for application 2 which is combined with the first drop box and inserted in front of Box 1.

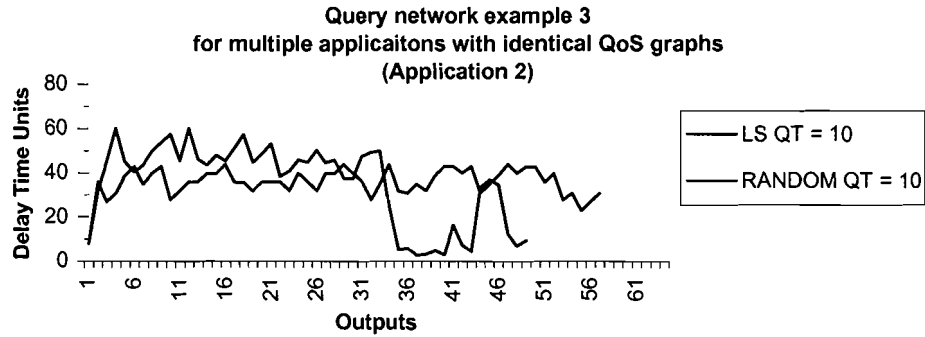
A)



B)



C)



D)

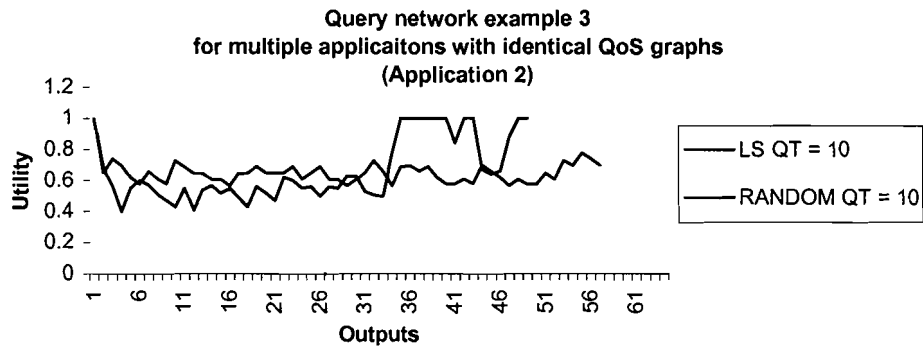


Figure 12. Drop-based Simulation for Query network example 3.

Simulation is conducted in the following parameters. Input Streams: 2000; Max_Drop_Unit = 0.1; Time_Slice = 0.9; Mon_Hold_Time = 10; Queue_threshold (QT) = 10; Scale_Factor = 2.
A) Delay-outputs and B) Utility-outputs graphs for application 1.
C) Delay-outputs and D) Utility-outputs graphs for application 2.
LS: Load-shedding algorithm; QT: Queue_threshold;
RANDOM: random dropping algorithm.

Table 3. Drop-based Simulation for Query network example 3

Application 1	Load-Shedding QT = 10	RANDOM
Total Outputs	47	49
End Simulation Time	1952.5	1993
Throughput	0.0241	0.0246
Theoretical Outputs	180	180
%Messages	26.1%	27.2%
Utility (Message-based QoS)	0.26	0.27

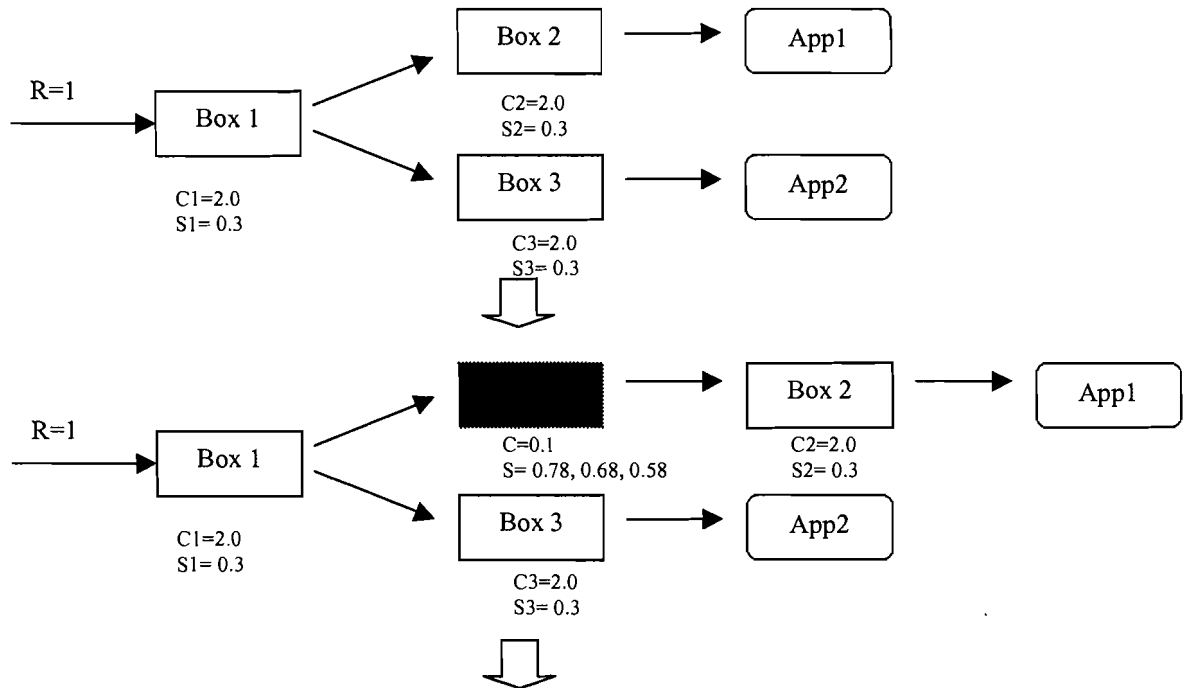
Application 2	Load-Shedding QT = 10	RANDOM
Total Outputs	49	57
End Simulation Time	1995.5	2021.8
Throughput	0.0246	0.0283
Theoretical Outputs	180	180
%Messages	27.2%	31.7%
Utility (Message-based QoS)	0.27	0.32

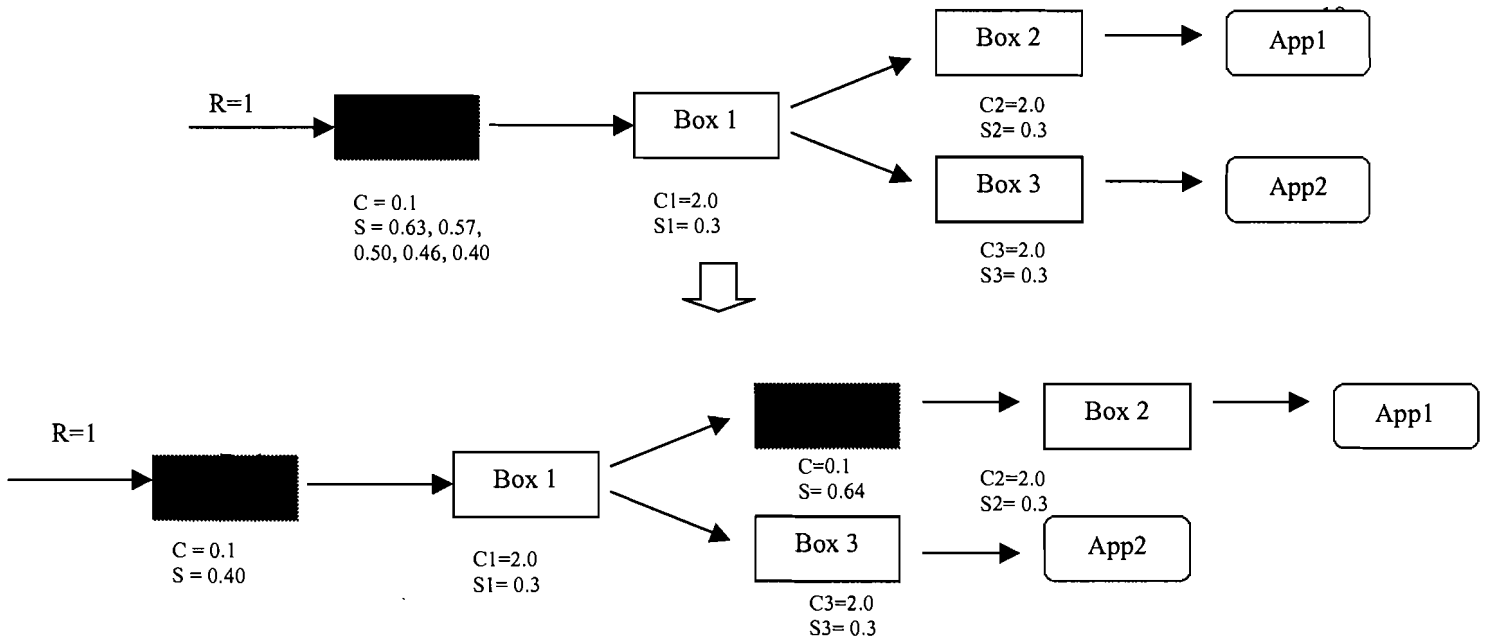
Similar to query network example 1 and query network example 2, the load-shedder successfully reduced the delay time units for the messages below the threshold (Fig. 12A, 12C) and the delay-based utility for the outputs reached 1 eventually (Fig. 12B, 12D). In addition, similar throughputs and message-based utilities were achieved for both applications (Table 3). Overall, it was fair for both applications in terms of the delay-based utility, overall throughput and the message-based utility (Fig. 12 and Table 3). Of note, the parameters were more difficult to be adjusted for multiple applications. A small change for time slice would alter the results significantly. Additionally, delays for the output messages also depended on the positions of the boxes in the scheduler (observation not shown). The performance of the load shedding algorithm and the scheduler are therefore tightly related.

4.2.2.2 Multiple Applications with Different QoS Graphs

To access the performance of the algorithms in network with multiple applications and different QoS graphs, we kept the framework of query network example 3 but specified different delay thresholds for each application and evaluated whether the load-shedder algorithm yielded fair results (Fig. 13).

A)





B)

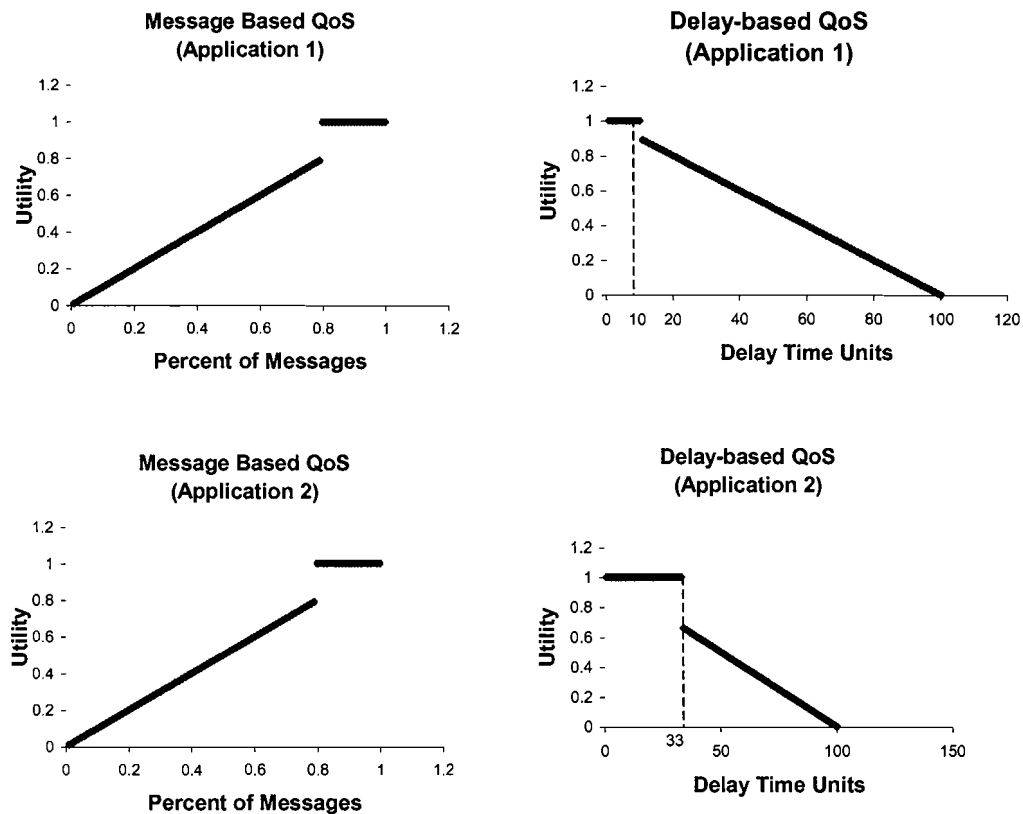


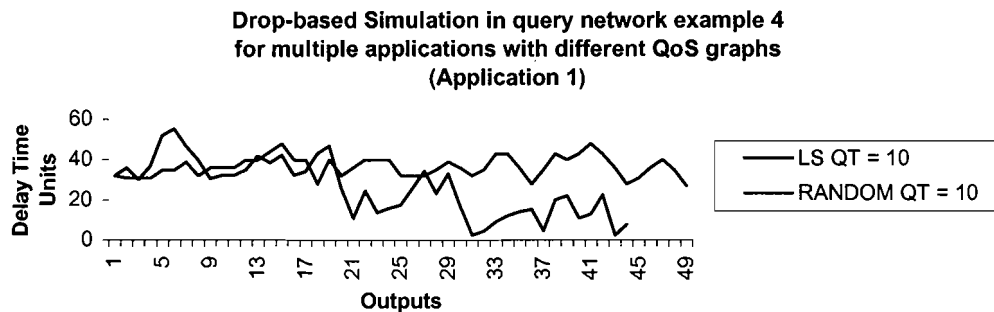
Figure 13. Query network example 4 for multiple applications

with different QoS graphs. A). Original network for Query network example 4 and the network after load shedding. R: input stream rate; C: box cost; S: box selectivity. B). QoS (quality of services) graphs for application 1 (App 1) and application 2 (App 2). App 1 and App 2 share the same message-based QoS graph whereas their delay thresholds are different (10 for App 1 and 33 for App 2).

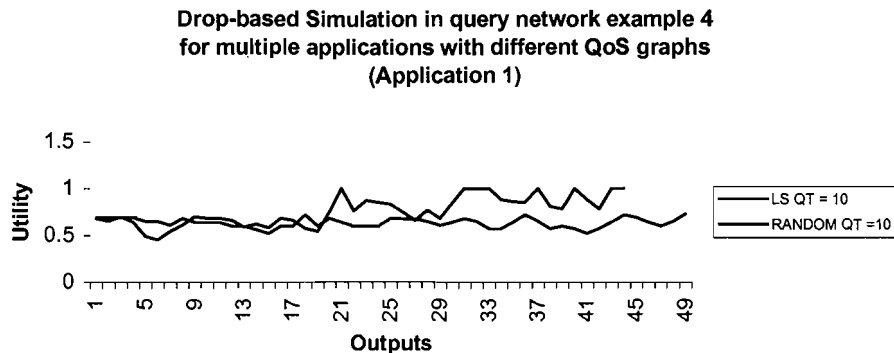
In query network example 4, a drop box is first generated for application 1 before Box 2 with cost 0.1 and selectivity 0.78 after load-shedding (S decreased from 0.78, 068 to 0.58) (Fig. 13A). Later on, when the outputs for application 2 exceed the delay threshold, a new drop box is generated for application 2 which is combined with the first drop box and inserted in front of Box 1 (S decreases from 0.63 to 0.40). Since the threshold for application 2 is larger than application 1, an additional drop box with selectivity 0.64 is finally generated and inserted before Box 2 instead of continuing to decrease the selectivity in the drop box before Box 1.

Similar to query network example 1, query network example 2 and query network example 3, the load-shedder successfully reduces the delay time units for the messages below the threshold (Figure 14A, 14C) and the delay-based utility for the outputs reaches 1 eventually (Figure 14B, 14D). More importantly, it is fair for both applications in terms of the delay-based utility, overall throughput and the message-based utility (Figure 14 and Table 4), although these two applications have different delay thresholds. Although one additional drop box was added in the path of application 2, the overall throughputs for both applications are equal. This coincidence may be caused by other factors such as time slice in the scheduler. In contrast, random dropping ignores the specific requirements for each application and generates the same results.

A)

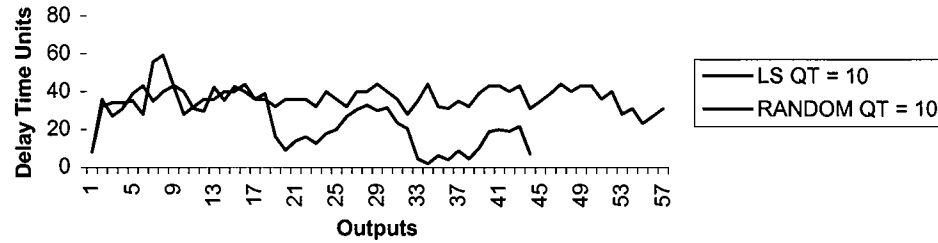


B)



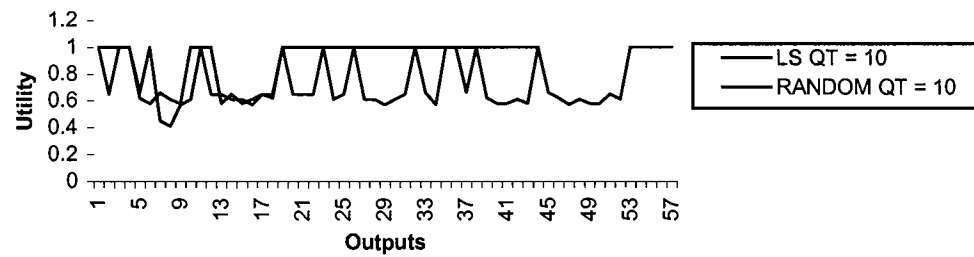
C)

**Drop-based Simulation in query network example 4
for multiple applications with different QoS graphs
(Application 2)**



D)

**Drop-based Simulation in query network example 4
for multiple applications with different QoS graphs
(Application 2)**



**Figure14. Drop-based Simulation in Query network example 3
for multiple applications with different QoS graphs.**

Simulation is conducted in the following

Parameters: Input Streams: 2000; Max_Drop_Unit = 0.1; Time_Slice = 0.9;
Mon_Hold_Time = 10; Queue_threshold (QT) = 10; Scale_Factor = 2.

A) Delay-outputs and B) Utility-outputs graphs for application 1.

C) Delay-outputs and D) Utility-outputs graphs for application 2.

LS: Load-shedding algorithm; QT: Queue_threshold;

RANDOM: random dropping algorithm.

Table 4. Drop-based Simulation for Query network example 4

Application 1	Load-Shedding QT = 10	RANDOM
Total Outputs	44	49
End Simulation Time	1955	1993
Throughput	0.0225	0.0246
Theoretical Outputs	180	180
%Messages	24.4%	27.2%
Utility (Message-based QoS)	0.24	0.27

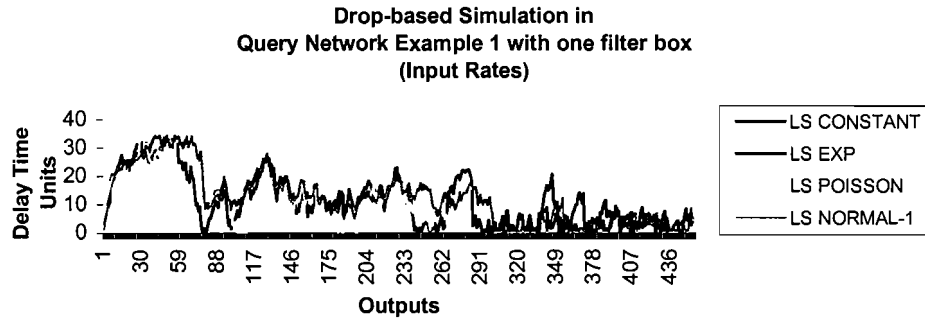
Application 2	Load-Shedding QT = 10	RANDOM
Total Outputs	44	57
End Simulation Time	1993.8	2021.8
Throughput	0.0225	0.0283
Theoretical Outputs	180	180
%Messages	24.4%	31.7%
Utility (Message-based QoS)	0.24	0.32

4.2.3 Input Rates

In the previous experiments, input rates for all the data streams were kept constant throughout the simulation. However, some of the streams may enter the Aurora system with changing rates. In this section, we first address the influences of input rates on both drop-based load-shedding algorithm and the random-dropping algorithm.

The random dropping algorithm appears to be insensitive to the changes of the input rates (Fig. 16 and Fig. 18). In contrast, input rates generated by different distribution functions (parameters specified in the figure legends) change the performance of the load-shedding algorithm in both query network example 1 (one filter box) and query network example 2 (two filter boxes) (Fig. 17 and Fig. 19). Small variance in the normal distribution leads to better results (Fig. 16 B and Fig. 18B). However, when the maximum rates are fast and the rate variations are frequent, the algorithm performs poorly. Statistical analysis on the previous outputs for a given application may provide some useful information to predict the input rates for periodic streams. The load-shedding algorithm can then adjust its parameters accordingly. No simple solution can be applied for completely random input rates.

A



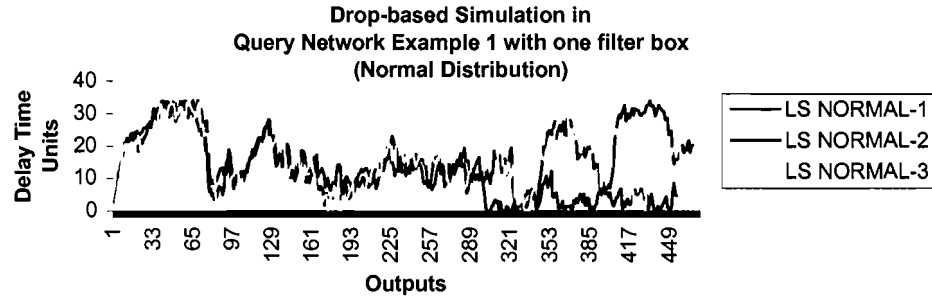


Figure 15. Different input rates for Drop-based load-shedding Simulation in Query network example 1 with one filter box. Simulation is conducted in the following parameters: Input Streams: 2000; Max_Drop_Unit = 0.1; Time_Slice = 1.0; Mon_Hold_Time = 10; Queue_threshold (QT) = 10; Scale_Factor = 4.5. A, B delay-outputs graphs for application 1. LS: Load-shedding algorithm; CONSTANT: input rate = 1.0; EXP: input rate = exponential (1.0); POISSON: input rate = poisson (1.0); NORMAL-1: input rate = normal (mean = 1.0, variance = 1.0/10); NORMAL-2: input rate = normal (mean = 1.0, variance = 1.0/5). NORMAL-3: input rate = normal (mean = 1.0, variance = 1.0/2).

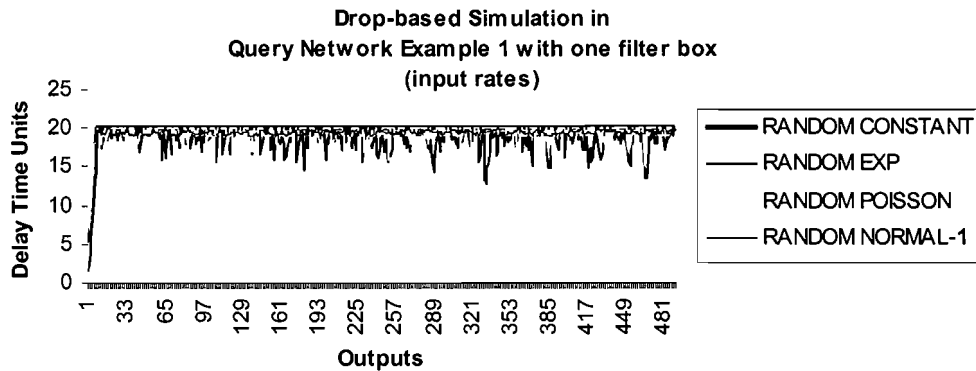
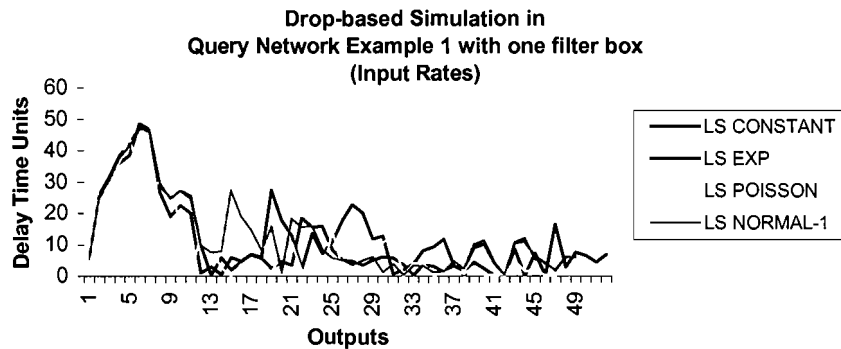


Figure 16. Different input rates for random dropping Simulation in Query network example 1 with one filter box. Simulation is conducted in the following parameters: Input Streams: 2000; Time_Slice = 1.0; Queue_threshold (QT) = 10. Delay-outputs graph for application 1. RANDOM: random dropping algorithm. CONSTANT: input rate = 1.0; EXP: input rate = exponential (1.0); POISSON: input rate = poisson (1.0); NORMAL-1: input rate = normal (mean = 1.0, variance = 1.0/10).



**Drop-based Simulation in
Query Network Example 1 with one filter box
(Normal Distribution)**

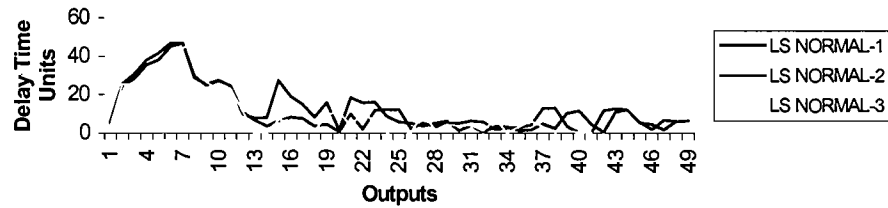
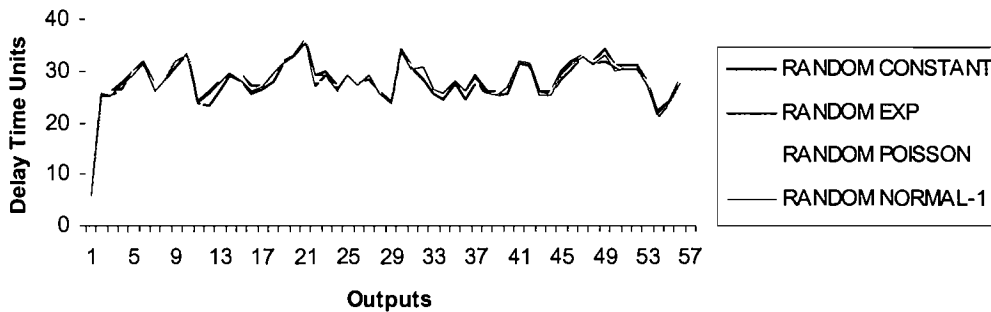


Figure 17. Different input rates for Drop-based load-shedding Simulation in Query network example 2 with two filter boxes. Simulation is conducted in the following parameters: Input Streams: 2000; Max_Drop_Unit = 0.1; Time_Slice = 1.0; Mon_Hold_Time = 10; Queue_threshold (QT) = 10; Scale_Factor = 4.5. A, B delay-outputs graphs for application 1. LS: Load-shedding algorithm; CONSTANT: input rate = 1.0; EXP: input rate = exponential (1.0); POISSON: input rate = poisson (1.0); NORMAL-1: input rate = normal ($\delta = 1.0$, $\lambda = 1.0/10$); NORMAL-2: input rate = normal (mean = 1.0, variance = $1.0/5$). NORMAL-3: input rate = normal (mean = 1.0, variance = $1.0/2$).

Case 2 Drop-based Simulation



Case 2 Drop-based Simulation

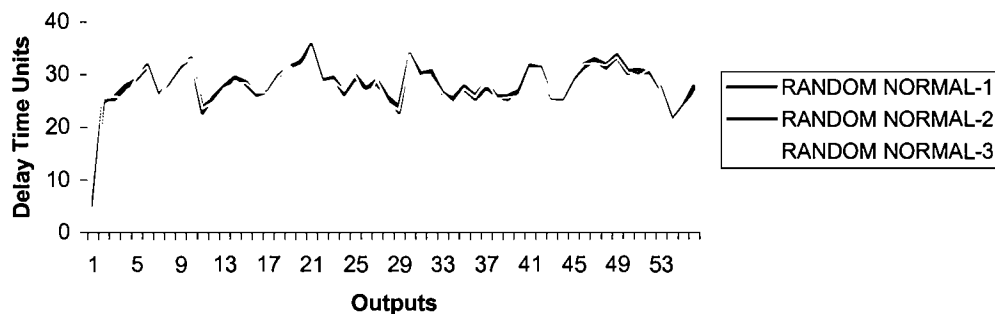


Figure 18. Different input rates for random dropping Simulation in Query network example 2. Simulation is conducted in the following parameters: Input Streams: 2000; Time_Slice = 1.0; Queue_threshold (QT) = 10. Delay-outputs graphs for application 1. RANDOM: random dropping algorithm. CONSTANT: input rate = 1.0; EXP: input rate = exponential (1.0); POISSON: input rate = poisson (1.0); NORMAL-1: input rate = normal (mean = 1.0, variance = $1.0/10$).

5 Improving the load-shedding algorithm

In the previous section, the new messages are dropped when the number of messages for each box (or arc) is beyond the specified **Queue_Threshold**. In this section, we first improve the load-shedding algorithm by dropping old messages in the queue. This approach will decrease the overall delay time for the outputs.

We also consider the situations when the input rates are so low that the output delay time is below the threshold and no drop boxes will be needed. However, the existing drop boxes would remain there and this may lead to unnecessary loss of messages. To resolve this problem, a simple algorithm was implemented according to the pseudo codes described in Fig. 19. Basically, the selected applications have drop box(es) on their paths and their output delay time units are below the threshold specified in the **delay-based QoS graph**. A maximum slope is picked and counted according to the **message-based QoS graphs** for these applications. The selectivity for each drop box is at most **MAX_DROP_UNIT**. This algorithm is combined with the load-shedding algorithm to dynamically adjust the network to its best performance.

```

Ai: application i,  $i \in [1, n]$ .
Qi: message-based quality of service graph for application i,  $i \in [1, n]$ .

A = {A1, A2, A3, ..., An};
Q = {Q1, Q2, Q3, ..., Qn};

Max_Slope  $\leftarrow$  0
Max_Count  $\leftarrow$  0
First, a maximum slope is picked and counted for the applications below the delay threshold. These applications should already at least have one drop box on the path.
for i  $\leftarrow$  1 to n
    if Ai  $\rightarrow$  delay_flag == 1
        slope  $\leftarrow$  (slope of Qi for current cursor)
        if slope > Max_Slope
            Max_Slope  $\leftarrow$  slope
            Max_Count  $\leftarrow$  1
        else if slope == Min_Slope
            Max_Count  $\leftarrow$  Max_Count + 1
    2
for i  $\leftarrow$  1 to n
    if Ai  $\rightarrow$  delay_flag == 1 and Ai has drop box(es) on its path
        slope  $\leftarrow$  (slope of Qi for current cursor)
        if slope == Max_Slope
            j  $\leftarrow$  0
            do
                slope  $\leftarrow$  (slope of Qi for current cursor + j)
                j  $\leftarrow$  j + 1
                while slope == Max_Slope and current cursor + j < 100
                    j < 100 * MAX_DROP_UNIT / Max_Count

            scale_factor  $\leftarrow$  (current cursor + j) / current cursor
            Start from the last drop box DB being added for the application
            m  $\leftarrow$  number of applications in connection with DB
            if m > 1 remove the drop box from the net
            else if m == 1
                new_selectivity  $\leftarrow$  scale_factor * old selectivity
                if (new_selectivity > 1 and no messages attached to the drop box)
                    remove drop box
            else
                set the drop box's cost to be 0 and its selectivity to be 1

```

Figure 19. Pseudo code for Drop-based Drop Box Removal Algorithm in Load-Shedder

The monitor was adjusted accordingly to consider both shedding loads and removing drop boxes. Mon_Hold_Time value, which decides the sleep interval for the monitor, increases when the load shedder is adding new drop boxes. This allows the network to recover. The value of Mon_Hold_Time decreases when the load shedder is successively removing existing drop boxes. One apparent pitfall of this method is that the load shedder may constantly adding and removing boxes. To avoid this problem, a range for delay-based threshold was specified according to the tolerance of delay-based utility from the customers. Once the delay time period is within that range, the monitor will not call the load-shedder. When all drop boxes are removed from the network and the delay time period for all the applications are within the delay threshold range, the network reaches a stable stage and Mon_Hold_Time will increase.

Testing of this algorithm in simple query network example 1 indicated that the removal of the drop boxes significantly improved the total throughput when the rates were periodically below and above the total process time (Fig. 20).

This algorithm can also be applied to query network examples with multiple applications with different input rates (Query network example 5 in Fig. 21). Removing the drop boxes greatly improved the overall throughput compared with the algorithm with no removal. **The input rates were described in the figure legends.**

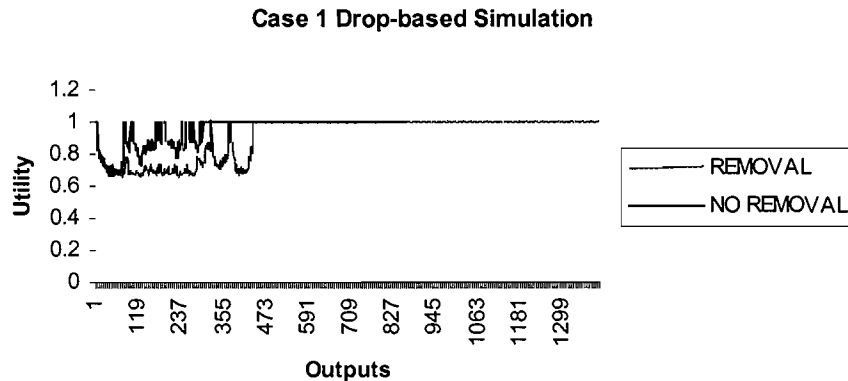
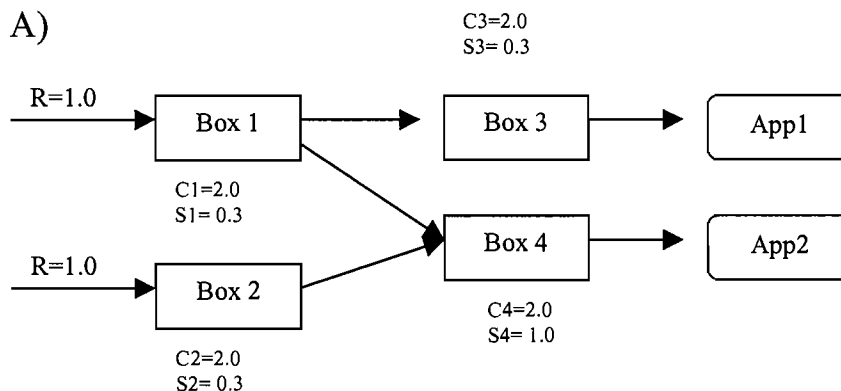
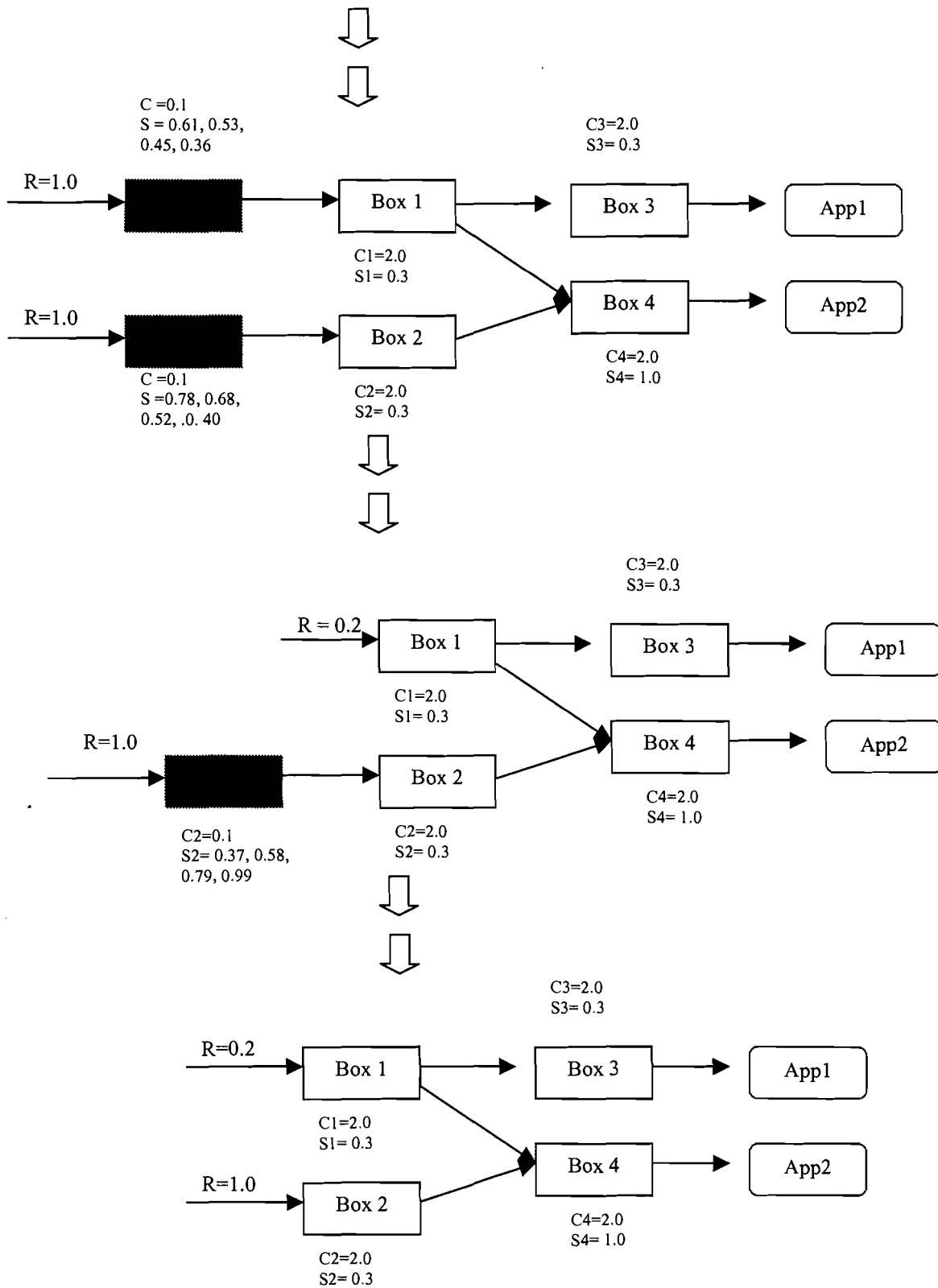


Figure 20. Removal of drop boxes improves the throughput for load-shedding algorithm in query network example 1 with one filter box.

Simulation is conducted in the following parameters: Input Streams: 2000; Max_Drop_Unit = 0.1; Time_Slice = 1.0; Mon_Hold_Time = 10; Queue_threshold (QT) = 10; Scale_Factor = 4.5. **Input rates:** stream 1 to 2000, $r_{in} = 1.0$; stream 2000 to 4000, $r_{in} = 0.2$.





6 Summary and Future Directions

Our simulation model successfully simulates the query network in CSIM18. The random dropping algorithm is only sensitive to Queue_Threshold and in some query network examples, Time_Slice. In contrast, a number of parameters including Queue_Threshold, Time_Slice, Maximum_Drop_Unit, Mon_Hold_Time, Scale_Factor and input rates all influence the performance of the load-shedding algorithm. Except Time_Slice, the other parameters are likely to be present in the Aurora system. Since the large-scale network possess more complicate features. Dynamically monitoring and tuning these parameters will be required to ensure that efficient load shedding is achieved without sacrificing too many messages. From our experiments, a Scale_Factor larger than 1 appears to be especially important for the network to recover before additional drop box is inserted into the network.

The goal of the load-shedding algorithm is to reduce the workload so that eventually most of the messages can pass with delays below the specified thresholds. In our experiments, the load-shedding algorithm achieves this goal in all query network examples. As a tradeoff, however, a slightly more messages are lost compared with the random dropping algorithm. When comparing the performance of the two algorithms, we use the same Queue_Threshold for load-shedding and random algorithm. Of note, load-shedding algorithm without queue threshold performs well in simple query network example 1 and 2. But it fails to effectively reduce workloads for the specified parameters in Query network example 3 and Query network example 4. Delay time units accumulate over time and utilities for the outputs become zero, indicating that applying the queue threshold in the load-shedding algorithm is realistic in the network.

One advantage of the load-shedding algorithm is that it can be adjusted according to the requirements of different applications without losing fairness to each one. We expect this feature will be valuable in the more complicate network of the Aurora system.

We improved the performance of the algorithm through removing drop boxes when the output delay time unit is less than the threshold in delay-based QoS graph. This feature was proved to increase the overall throughput while achieving the goal of the load-shedding algorithm. One pitfall of this algorithm is that the load-shedder might constantly adding and removing drop boxes. To reduce the fluctuations, a threshold range is specified, within which the load shedder is not triggered.

Our future directions include the following goals:

- 1) Identify and characterize general properties of the network graph which can change the performance of the algorithm, such as number of splitting point, number of merge boxes, positions of splitting point and positions of merge boxes
- 2) Implement and test value-based load-shedding algorithm.
- 3) In our current version, we assume that the delay threshold has priority over the percentage of messages being processed. In reality, these two factors may be equally important or have different weights for the customers. An interesting direction will be to define a comparable message percentage threshold on the message-based QoS graphs. On one hand, new drop boxes are added when delay time units exceed the delay threshold. On the other hand, drop boxes will be removed when the percentage of output messages are below the specified message percentage threshold. The load shedder will

try to achieve the goal of reducing the delay time while maintaining a reasonable number of output messages.

References:

- [1] D. Carney, U. Cetinemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. Zdonik. Monitoring Streams – A New Class of Data Management Applications. Brown Computer Science Technical Report, TR-CS-02-04.
- [2] G. Edwards and R. Sankar. Modeling and simulation of networks using CSIM. Simulation. 58(2): 131-136, 1992.
- [3] Y. Huang, P. K. McKinley. Switch-Aided Flooding Operations in ATM Networks. IEEE INFOCOM.1997.
- [4] J. H. Kaufman, J. Myllymaki, J. Jackson. 2001. City Simulator V 2.0. IBM Almaden Research Center. <http://alphaworks.ibm.com/tech/citysimulator>.
- [5] P. K. McKinley and C. Trefftz. Multisim. A simulation tool for the study of large-scale multiprocessors. In Proceedings of the 1993 Int. Workshop on Modeling, Analysis, and Simulation of Comp. and Telecom. Networks, 57 – 62, 1993
- [6] Mesquite Software, Inc. CSIM18 Users Guide. Austin, TX. 1996.
- [7] H. Schwetman. "Csim: A C-based, process-oriented simulation language," Tech. Rep. Microelectronics and Computer Technology Corporation. PP080 –85, 1985.
- [8] H. Schwetman, J. Charnes, D. Morrice, D. Brunner, and J. Swain. 1996. CSIM18 - The Simulation Engine. In Proceedings of the 1996 Winter Simulation Conference. 517 - 521.