3-11-2003

Performance Analysis and Efficient Implementation of Latency Insensitive Systems

Ruibing Lu
Cheng-Kok Koh

Follow this and additional works at: http://docs.lib.purdue.edu/ecetr

http://docs.lib.purdue.edu/ecetr/148

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
Performance Analysis and Efficient Implementation of Latency Insensitive Systems *

Ruibing Lu, Cheng-Kok Koh
ECE, Purdue University
West Lafayette, IN, 47907, USA
{lur, chengkok}@ecn.purdue.edu
March 11, 2003

*This research is supported in part by SRC (99-TJ-689), NSF (CAREER Award CCR-9984553), and a grant from Intel Corporation.
# Contents

1 Introduction 1

2 Fundamentals of LISs 2

3 LIS with minimum queue (MQ-LIS) 3
   3.1 Performance analysis 4

4 General LIS 7

5 Circuit implementation 7

6 Experimental results 9

7 Conclusion 10
<table>
<thead>
<tr>
<th></th>
<th>Table Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Possible states of channel</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Experimental results</td>
<td>10</td>
</tr>
</tbody>
</table>
### List of Figures

1. State graph of communication channel. ........................................ 4
2. An example lis-graph. ................................................................. 4
3. An example extended lis-graph. ................................................... 6
4. Circuit implementation of LIS. ..................................................... 8
5. Enumeration of empty event transformation .................................. 10
Abstract

This paper studies the performance of latency insensitive systems with limited queue size, in contrast with previous studies that assumed unlimited queue size. We obtain a tight theoretical performance upper bound of such systems. This paper also proposes an efficient implementation of latency insensitive systems. Experimental results show that our implementation can always reach the theoretical performance upper bound. The results also validate the claim that the upper bound acquired through our performance analysis is tight.
1 Introduction

As the system complexity increases and the feature size of the manufacture technologies scales down to deep sub-micron dimensions, the interconnect delay becomes the dominating factor of the system performance. In fact, the interconnect delay can be as long as about ten clock cycles [5] in the near future. Unfortunately, an accurate estimation of interconnect delay is only possible at the physical design steps. As a result, IC designs become a slow iteration between logic design and physical design with no guarantee of convergence.

Recently, a latency insensitive design methodology that orthogonalizes the computation and communication of circuit blocks has been proposed [2, 3, 1]. A latency insensitive system (LIS) is composed of a collection of computational processes that exchange data on communication channels. The main characteristic of LISs is that arbitrary latency of any communication channels can be tolerated. In LISs, relay stations are used to partition and “pipeline” the communication channels that have long interconnect delay. This introduces different latencies to communication channels. To synchronize the data, a computational unit stalls when any of its input data is not available, and in such a case, non-informative data is put on the output ports of the stalled unit. Extra buffer queue is put on the channels to temporally store the informative data already generated but not ready to be consumed. After all non-informative data are screened, the function of an LIS is equivalent to the original one.

With the latency insensitive design methodology, the timing violation due to long interconnect delay can be resolved by inserting relay stations in the physical design step. Clearly, the design iteration and timing closure problem would be alleviated. However, the introduction of relay stations may reduce the system throughput because those relay stations introduce non-informative data into the system.

The performance analysis of LISs in [3] suggests that the system throughput is only limited by relay stations in feedback cycles. However, their results are based on the assumption that a unit stalls only if some input data is not available. A direct implication is that the buffer queue of communication channels would never overflow. However, it is unrealistic to require buffer queues to have unlimited capacity. To overcome that, an “equalization” method is proposed in [3]. Essentially, relay stations are inserted not only to eliminate timing violations due to long interconnect delay, but also to slow down fast components in order to equalize the throughput of all components. This however, may lead to an unacceptable large number of relay stations.

These limitations are not inherent in the latency insensitive design methodology. In fact, it is not necessary to have unlimited queue size if source units can stall when any of its output channels is full [1]. However, it is not clear what the performance of LISs with limited queue size is.

In this paper, we formally study the system performance of LISs with limited queue size. Our analysis shows that relay station insertion in both feedback cycles and re-convergence paths may affect the performance. Moreover, a tight provable performance upper bound can be computed efficiently. An efficient implementation of LIS is also proposed in this paper. Compared with the implementation in [1], our implementation can fully exploit the channel queues and inherently support queues with various sizes. Experimental results show that our implementation can always reach the theoretical performance upper bound. The results also validate the claim that the upper bound acquired through our
Table 1: Possible states of channel.

<table>
<thead>
<tr>
<th></th>
<th>SU</th>
<th>SA_{t+1}</th>
<th>DA_{t+1}</th>
<th>OF?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>R</td>
<td>R</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>R</td>
<td>S</td>
<td>YES</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>S</td>
<td>R</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>S</td>
<td>S</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>N</td>
<td>R</td>
<td>R</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>N</td>
<td>R</td>
<td>S</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>N</td>
<td>S</td>
<td>R</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>N</td>
<td>S</td>
<td>S</td>
<td>No</td>
</tr>
</tbody>
</table>

performance analysis is tight.

2  Fundamentals of LISs

A latency insensitive system can be divided into two parts: communication channels and circuit units. The latter includes both ordinary circuit blocks and relay stations. Communication channels in LISs perform the following functions:

1. Receive data from the source unit and automatically screen non-informative data.
2. Buffer the data into the dedicated buffer queue of the channel if the sink unit is not ready to accept it.
3. Provide data to the sink unit in order and inform the sink unit to stall when no data is available.
4. Inform the source unit to stall in time to avoid the lost of informative data due to queue overflow.

In order to fulfill all those functions, two additional interconnects with opposite directions are typically needed for each communication channel as in [1]; the one that has the same direction as data transmission is used to identify whether the data is informative and the other one is to inform the source unit to stall.

In this paper, we study the LISs with the following two characteristics:

1. A unit determines its action (to stall or to compute) of the next timestamp based on only the states of its input and output channels. Similarly, a channel determines its next state based on its source and sink units.
2. At any timestamp, a unit stalls if and only if at least one of its input channels cannot provide the required data or it may cause buffer queue overflow in its output channels.

The first characteristic simplifies the implementation complexity. More important, if channels or units use information from units or channels other than their immediate source or sink, additional long interconnect delay will be introduced for the transmission of such information.

The second characteristic avoids unnecessary stallings of circuit units. For a unit, both its input channels and output channels can force it to stall. If any of the input channels cannot supply the related data to the unit, it has to stall. An output channel may also request the source unit to stall in order to avoid possible buffer overflow.

When the queue of a channel is not full, clearly its source unit should not stall for this channel. To investigate the condition under which the source unit should be stalled in order to avoid overflow, we list all possible combinations
of a channel with full queue in Table 1. In this table, $SU_t$ shows whether the output of the source unit is informative at timestamp $t$. The entry “I” and “N” indicates an informative data and a non-informative data respectively. $SA_{t+1}$ and $DA_{t+1}$ refer to the action at timestamp $t+1$ of the source unit and the sink unit, respectively; the entry “R” indicates that the unit will proceed to generate new data while “S” indicates that the unit will stall. The column “OF?” shows whether overflow will happen at the $t+1$ timestamp.

Here, we explain only two of these combinations as the rest are similar. For the first combination, one set of data is already used by the sink unit at timestamp $t+1$, so the queue can have an empty space for the informative data generated at timestamp $t$ by the source unit.

In the second combination, the sink unit is stopped at the timestamp $t+1$, thus every data in the queue is informative and the queue is still full. However, the source unit generates a new informative data at the timestamp $t+1$; the informative data generated at timestamp $t$ then cause the overflow of the already full queue. Therefore, we should avoid such a combination. However, at timestamp $t$, it is impossible for the channel to know the future actions of its source or sink unit. Therefore, in order to avoid the possible overflow, it is necessary to make the source unit to stall in the following timestamp as long as the source unit generates an informative data while the queue is full.

The queue size will decide how frequent a channel will issue stall requests. For an LIS, the queue size of any channel should be at least 1 in order to guarantee the correct system behavior. We call an LIS in which the queue size of channels is uniformly 1 to be an LIS with minimum queue, or simply MQ-LIS. In this paper, we first focus on the performance analysis of MQ-LIS, and then generalize the results for LISs with arbitrary queue sizes.

### 3 LIS with minimum queue (MQ-LIS)

MQ-LIS can be modeled as a system with many finite state machines, each of which models one communication channel in the MQ-LIS.

The possible states of communication channels are: Informative event (IE), empty data (ED) or stall request (SR); the latter two are called empty events. A channel is in ED state if the queue is empty and the output of the source unit is non-informative; a channel is in SR state if the queue is full and the output of the source unit is informative; the state of a channel in all other situations is called IE. The state of a unit can be defined directly from the states of input channels and output channels: A unit is in stall state (SS) if any input channel is in ED state or any output channel is in SR state, otherwise it is in normal state (NS). Note that a unit is in SS state if and only if this unit will stall in the next timestamp.

The initial state of output channels of relay stations is ED, and all other channels are in IE state initially.

The state graph of communication channels is shown in Figure 1, in which “IN” and “OUT ” refer to the state of source unit and sink unit of the channel respectively. The state transition rules are as follows:

1. The next state of a channel, whose current state is IE, will still be IE if and only if both the source unit and sink unit are in NS state or both of them are in SS state. It will be ED if only the source unit is in SS, and will be SS if only the sink unit is in SS.
2. The next state of a channel, whose current state is ED, will be ED if and only if the source unit is in SS state. Otherwise, it will change to IE state.

3. The next state of a channel, whose current state is SR, will be SR if and only if the sink unit is in SS state. Otherwise, it will change to IE state.

3.1 Performance analysis

In this paper, we assume that all primary input channels can always supply the required data in time and all primary output would not generate stall requests. In other words, the system throughput is determined only by its structure.

From the state transition rules, the state of the system, which is the combination of states of all channels, is fully determined by the previous state at any timestamp other than the initial state. In addition, the possible combinations of channel states is finite; therefore, the system must be periodic. In this paper, the throughput is defined as the number of informative events produced by the system over the number of timestamps in each period.

In order to model the structure of LISs, we define the latency insensitive system graphs (lis-graphs) as following:

**Definition 1** A lis-graph $G(V, E, w)$ is a weighted connected directed graph, where $V$ is the set of all circuit units including original circuit blocks and relay stations, $(v_i, v_j) \in E$ refers to the communication channel from unit $v_i$ to unit $v_j$, $w(v_i, v_j) \in \{0, 1\}$ and $w(v_i, v_j) = 1$ if and only if the unit corresponding to $v_i$ is a relay station.

Figure 2 shows an example lis-graph. Here, $v_6$ is the only relay station.

**Definition 2** An induced cycle $C$ of a lis-graph $G(V, E, w)$ is a list $v_0, e_1, v_1, \ldots, e_{k-1}, v_k, v_0$ such that, for $1 \leq i \leq k$, $e_i$ is either $(v_{i-1}, v_i)$ or $(v_i, v_{i-1})$, and $v_0$ is the same vertex as $v_k$. The length of this induced cycle, denoted as $|C|$, is defined
as the number of vertices $k$. $e_i$ is a forward edge if it is $(v_{i-1}, v_i)$; otherwise, it is a backward edge. The relative sum of edge weight $\Delta(C)$ is defined as:

$$\Delta(C) = \sum_{e_i \in E_F(C)} w(e_i) - \sum_{e_j \in E_B(C)} w(e_j) \tag{1}$$

where $E_F(C)$ and $E_B(C)$ are the sets of forward edges and backward edges, respectively.

The relative sum of edge weight can be either negative or positive. Note that a negative relative sum of edge weight of any induced cycle can be converted to a positive one by simply reversing the cycle direction. Therefore, we assume that the relative sum of edge weight is non-negative.

**Definition 3** Given an induced cycle $C$, the relative number of empty events $\delta(C, t)$ at timestamp $t$ is $(N_{ED,F} + N_{SR,B}) - (N_{ED,B} + N_{SR,F})$, where $N_{ED,F}$, $N_{ED,B}$, $N_{SR,F}$ and $N_{SR,B}$ are the number of forward edges in ED state, the number of backward edges in ED state, the number of forward edges in SR state, and the number of backward edges in SR state, respectively.

**Corollary 1** Given an induced cycle $C$ in an MQ-LIS, its relative number of empty events at any timestamp $t$ is a constant value that is equal to $\Delta(C)$.

Proof: See Appendix.

As the relative number of empty events of any induced cycle $C_i$ in an MQ-LIS is a constant, we simply use $\delta(C_i)$ to denote this constant when the system is an MQ-LIS.

Now we want to count the number of units in SS state because a unit in SS state will stall in the next timestamp and this can help us to estimate the system throughput.

**Corollary 2** In an induced cycle $C_i$ in an MQ-LIS, the number of units in SS states is at least $\delta(C_i)$.

Proof: Clearly, any empty event will set exactly one unit into SS state: a channel in ED state will set the sink unit into SS state and a channel in SR state will set the source unit into SS state. However, in an induced cycle, two consecutive channels may set the unit between them to SS state simultaneously. For example, in instance B of Figure 5, either channel 1 with ED state or channel 2 with SR state make unit $y$ in SS state. We call these two consecutive channels “synchronized empty channels". The total contribution of any two synchronized empty channels to the relative sum of empty events is always zero. In addition, any channel with empty event can be synchronized with at most one other channel in one induced cycle. Therefore, the relative sum of empty events is unchanged if all the synchronized empty channel pairs are not counted. Excluding all these synchronized channel pairs, every counted empty event sets exactly one unit to SS state in the induced cycle by itself, and there are at least $\delta(C_i)$ such empty events. Therefore, the number of units with SS state in an induced cycle $C_i$ is always not fewer than $\delta(C_i)$.

A unit in SS state implies that the unit will stall in the next timestamp. Therefore, at least $\delta(C_i)$ units in $C_i$ do not generate informative data at any timestamp.
Figure 3: An example extended lis-graph.

**Theorem 1** Given an MQ-LIS, its lis-graph $G$, and the set of all induced cycle $SC$, the system throughput is at most

$$1 - \max_{C \in SC} \left( \frac{\delta(C)}{|C|} \right).$$

Proof: As the system is periodic, we denote the period of the system by $T$ and the number of informative data produced by the system in each period by $N$. Each unit in the system will produce the same number of informative data. Otherwise, the difference over the number of informative data produced by two units can be infinitely large after sufficient number of periods. This contradicts with the fact that the system is fully connected, and the total queue size of any path is limited. Therefore, the total number of informative data produced by all units in $C_i$ is $N \times |C_i|$, and the total number of non-informative data produced by all units in $C_i$ is at least $\delta(C_i) \times T$. We have:

$$\frac{N \times |C_i| + \delta(C_i) \times T}{T} \leq |C_i|$$

$$\frac{N}{T} \leq 1 - \frac{\delta(C_i)}{|C_i|}$$

Therefore, the minimum $1 - \frac{\delta}{|C_i|}$ among all induced cycles, or $1 - \max_{C \in SC} \left( \frac{\delta(C)}{|C|} \right)$, is an upper bound of the system throughput.  

**Definition 4** The extended lis-graph $G_E(V, E, w)$ of an LIS is a weighted connected directed graph acquired by adding into its lis-graph $G(V, E, w)$ mirror edges $(v_j, v_i)$ with weight $1 - Q(v_i, v_j) - w(v_i, v_j)$ for each edge $(v_i, v_j) \in E(G)$. $Q(v_i, v_j)$ is the queue size of the channel corresponding to $(v_i, v_j)$.

In an MQ-LIS, queue size is always 1. Therefore, an edge and its mirror edge will have opposite edge weights. Figure 3 shows the extended lis-graph for the lis-graph in Figure 2.

**Definition 5** Given a weighted directed graph, the mean weight of a cycle is defined as the sum of the weights of the edges of this cycle, divided by the length of this cycle. The mean weight of a cycle is also called the cycle mean.

Clearly, any induced cycle in a lis-graph corresponds to exactly one cycle in the extended lis-graph by mapping all its backward edges in the lis-graph to their mirror edges in the extended lis-graph. Moreover, for an MQ-LIS, the relative sum of edge weight of the induced cycle is equal to the sum of edge weight of the corresponding cycle.
Therefore, in order to obtain the critical induced cycles that corresponds to the throughput upper bound, we should seek the cycles with the maximum mean weight in the extended lis-graph. This can be acquired efficiently through any maximum cycle mean algorithm such as Karp’s algorithm [4].

4 General LIS

Now, we extend the results to consider general LISs with arbitrary queue sizes. For a channel \((v_i,v_j)\) with queue size \(Q(v_i,v_j)\), the IE state now is divided into \(Q(v_i,v_j)\) sub-states: IE\((k)\), \(0 \leq k \leq Q(v_i,v_j) - 1\). Here, “\(k\)” can be viewed as the number of used entries in the queue. When the source unit is in SS state and the sink unit is NS, only IE\((0)\) will go to the ED state, and other substate IE\((k)\) would just become IE\((k - 1)\). Similarly, when the states of the source unit and the sink unit are NS and SS respectively, only IE\((Q(v_i,v_j) - 1)\) become SR, and the next state for other state IE\((k)\) is IE\((k + 1)\). Other transition rules can be obtained similarly.

Note that Corollary 1 cannot applied to general LIS directly. For a channel \((v_i,v_j)\) with queue size \(Q(v_i,v_j)\) and in state IE\((0)\), the first \((Q(v_i,v_j) - 1)\) SRs that propagate to it would not change it to SR state. Conceptually, those SRs can be viewed as being “trapped” in the queue. In addition, when the queue has trapped SRs, the ED state that propagates to it would simply cancel with one of the trapped SRs. Given an induced cycle \(C\), let \(N_{SR,F}^T\) and \(N_{SR,B}^T\) be the number of trapped SRs in the forward channels and that of the backward channels, respectively. It is trivial to show that \(\delta(C,t) + N_{SR,B}^T - N_{SR,F}^T\) is still a constant equal to \(\Delta(C)\). Therefore, we have:

\[
\delta(C,t) \geq \Delta(C) - N_{SR,B}^T \geq \Delta(C) - \sum_{e \in E_B(C)} (Q(e) - 1)
\] (3)

The right hand side of this inequality is simply the sum of edge weight of the cycle acquired by mapping the induced cycle \(C\) into the extended lis-graph. Therefore, Theorem 2 can be extended to general LIS easily and the performance upper bound becomes:

\[
1 - \max_{C' \in SC'}\left(\frac{W(C')}{|C'|}\right).
\] (4)

where \(SC'\) is the set of all cycles in the extended lis-graph and \(W(C')\) the sum of edge weight of cycle \(C'\).

Note that this result is consistent with that in [3], which consider the system with unlimited queue size. In such a system, the weight of all mirror edges in the extended lis-graph is \(-\infty\). Therefore only cycles that do not include mirror edges in the extended lis-graph can limit the system performance; these cycles corresponds to feedback cycles in the system.

5 Circuit implementation

In this section, we shall introduce an efficient implementation of LISs, which follows the two basic characteristics presented in Section 2. In particular, we focus on the implementation of communication channels. In our implementation,
a relay station is simply a register, and usually can be just viewed as a circuit block. The only difference is that, at the first timestamp, relay stations do not produce any informative data as other circuit blocks.

A channel can be divided into two parts: the source part identifies whether the data is informative and determines whether the channel should issue a stall request to the source unit; the sink part maintains a queue for the data and performs input data synchronization.

Our implementation is shown in Figure 4. For a circuit unit, channel $i$ and channel $m$ are one of its input channels and one of its output channels, respectively. Only the sink part of channel $i$ and the source part of channel $m$ are shown in this figure.

In the sink part of channel $i$, signal $\text{QueueFl}_i$ indicates whether the queue is full and signal $\text{EmpData}_i$ is asserted if and only if the data is non-informative. Signal $\text{ED}_i$ is asserted when both the queue is empty and the data from the source unit is non-informative. The controller, which is simply an OR gate, outputs the signal $\text{Stall}$ to indicate whether the unit should stall in the next timestamp. If $\text{Stall}$ is negated, the queue will discard in the next timestamp the oldest data that has been placed on its output $\text{RealIn}_i$; otherwise, the output would remain unchanged. $\text{Data}_i$ will be pushed into the queue only if $\text{EmpData}_i$ is negated and the queue is not full. Note that the assertion of $\text{QueueFl}_i$ does not imply that the channel issues a stall request. In fact, based on the analysis in Section 2, a stall request should be issued only if the queue is full and the incoming data is informative. However, we cannot use $\text{EmpData}_i$ to determine whether a stall request should be issued to the source unit. If $\text{EmpData}_i$ was used, it would take a long delay for $\text{EmpData}_i$ from the source unit to reach the sink and for the feedback from sink unit to reach the source. Such a long delay may not be acceptable if the unit involved is a relay station. Therefore, the decision on stall request is left to the source part of the channel.

The source part includes only one sequential unit: a flip-flop to store signal $\text{EmpData}_m$. Signal $\text{SR}_m$, which is equal to $\text{QueueFl}_m \cap \text{EmpData}_m$ is for requesting the sink unit to stall. When $\text{Stall}$ is asserted, the circuit block would keep the previous state, and of course, all its outputs are unchanged. The signal $\text{EmpData}_m$ in the next timestamp would usually be set to the value of $\text{Stall}$. However, if the $\text{SR}$ is asserted, the informative data in the last timestamp is not yet pushed into the the full queue. Therefore, the unchanged output value in the next timestamp is still informative.
for the sink unit. Hence, $EmpData_m$ of the next timestamp is $Stall \cap \overline{SR}_m$. The controller is just an OR gate, because the unit needs to stall if any input channel is empty or any output channel requests it to stall.

6 Experimental results

A “progressive trace” [3] based simulator for the proposed circuit implementation was constructed to verify the system behavior and measure its throughput. The main advantage of using progressive trace in the simulation is that it makes possible the analysis of system behavior without the detailed information of the specific logic function of circuit blocks. We also implemented the performance analysis method proposed in this paper based on Karp’s algorithm [4]. For an LIS, the extend lis-graph was extracted; Karp’s algorithm was then performed to find the maximum cycle mean and the performance upper bound.

In order to evaluate the efficiency of our implementation and the accuracy of the performance analysis, a large number of randomly generated LISs were used. Five parameters guided the generation of the random LISs: $N_u$, $P_E$, $P_R$, $RN_{max}$, and $Q_{max}$. $N_u$ is the number of circuit units; $P_E$ is the probability that there is communication channel between any two units, whose communication direction is randomly selected; $P_R$ is used to decide the number of relay stations inserted into a channel, and the probability of inserting $n$ ($0 \leq n \leq RN_{max}$) relay stations into a channel is $P_R^n - P_R^{n+1}$; $Q_{max}$ is the maximum queue size for any channel, and the queue size of any channel is a randomly selected integer between 1 and $Q_{max}$. Since such a randomly generated LIS may not be connected, two extra units are added as the primary source unit and primary output unit. A channel is added from the primary source unit to any unit without predecessor. Similarly, a channel is added from any unit without successor to the primary output unit.

First, we observed that the progressive traces from the simulation of all randomly generated lis-graphs always have the correct behavior in terms of latency insensitive protocol. This shows the robustness of our circuit implementation. In addition, the performance acquired from simulator is always equal to the performance upper bound obtain from the performance analyzer. This illustrates the efficiency of our proposed implementation. It also suggest that the performance upper bound we have is really tight. Our conjecture is that this upper bound is actually the exact performance.

Due to the page limit, only a small part of the experimental results are shown in Table 2. For each configuration, the throughput is the average value of 5000 random LISs. $T_{MQ}$ is the average throughput for MQ-LISs, whereas $T_{RQ}$ is the average throughput for general LISs with random queue sizes ($Q_{max} = 4$). Only one average throughput is given because, for any random LIS tested, the simulator and the performance analyzer always gave the same result. Although the difference between $T_{MQ}$ and $T_{RQ}$ is minor for some configurations, it does not imply that the benefit of having a large queue size is marginal. The main reason for those minor difference is that the queue sizes were randomly assigned.
Table 2: Experimental results.

<table>
<thead>
<tr>
<th>$N_u$</th>
<th>$P_E$</th>
<th>$P_R$</th>
<th>$RN_{max}$</th>
<th>$T_{MQ}$</th>
<th>$T_{RQ}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.2</td>
<td>0.1</td>
<td>5</td>
<td>0.957</td>
<td>1.000</td>
</tr>
<tr>
<td>30</td>
<td>0.2</td>
<td>0.1</td>
<td>5</td>
<td>0.586</td>
<td>0.603</td>
</tr>
<tr>
<td>50</td>
<td>0.2</td>
<td>0.1</td>
<td>5</td>
<td>0.500</td>
<td>0.502</td>
</tr>
<tr>
<td>90</td>
<td>0.2</td>
<td>0.1</td>
<td>5</td>
<td>0.416</td>
<td>0.420</td>
</tr>
<tr>
<td>150</td>
<td>0.2</td>
<td>0.1</td>
<td>5</td>
<td>0.368</td>
<td>0.368</td>
</tr>
<tr>
<td>50</td>
<td>0.3</td>
<td>0.3</td>
<td>5</td>
<td>0.500</td>
<td>0.502</td>
</tr>
<tr>
<td>50</td>
<td>0.1</td>
<td>0.05</td>
<td>5</td>
<td>0.663</td>
<td>0.683</td>
</tr>
</tbody>
</table>

Figure 5: Enumeration of empty event transformation

7 Conclusion

The performance of latency insensitive systems with limited queue size is formally studied in this paper. A theoretical performance upper bound, which can be computed efficiently, is given. This paper also proposes an efficient implementation of latency insensitive systems that can fully exploit the channel queue with various sizes. The experimental results show that the implementation can always reach the theoretical performance upper bound. This also validates the claim that the performance bound is tight.

Appendix: Proof of Corollary 1

Corollary 1 is proved by the enumeration of all possible empty event transformations in which $N_{ED,F}$, $N_{SR,B}$, $N_{ED,B}$ or $N_{SR,F}$ may change. Figure 5 lists all possible basic instances. Each instance is part of an induced cycle whose direction is assumed to be clockwise. Channels are labeled with numbers and units are labeled by alphabets. Instances A through D illustrate the transformation of empty events inside the induced cycle, and instances E through H illustrate the influence on the induced cycle from the empty events outside the induced cycle.

Instance A: We study the propagation of empty events without interaction between empty events. Clearly an ED
state of channel 1 will be propagated to channel 2 in the next state if and only if channel 2 is not in SR state. Similarly
the SR state of channel 2 will move to channel 1 if and only if channel 1 is not in ED state. Otherwise, there would be
some interaction between empty events, which would be shown in other instances. The above two cases show that if
consecutive channels are in the same direction, ED states will propagate along that direction and SR states will
propagate in the opposite direction. Now consider the case when channel 2 is in ED state and channel 3 is not in ED
state, the next state of channel 3 will be SR. A similar case is that channel 3 is in SR state while channel 4 is not in
SR state, and the next state of channel 4 will become ED. In the above two cases, ED state on forward edges and SR
state on backward edges are swapped, again the sum of \( N_{ED,F} \) and \( N_{SR,B} \) remains unchanged. Similar conclusion can
be drawn for ED state on backward edges and SR state on forward edges.

Instance B: We investigate the interaction of the ED state and the SR state. The ED state of channel 1 no longer
propagates to channel 2 because the next state of channel 2, whose current state is SR, can only be IE or SR depending
on the state of unit \( z \). The SR state of channel 2 cannot propagate to channel 1 either because the next state of channel
1 can only be IE or ED depending on the state of unit \( x \). Therefore, both the ED state of channel 1 and the SR state of
channel 2 cannot be propagated. Thus, both \( N_{ED,F} \) and \( N_{SR,F} \) decrease by 1, i.e., the relative number of empty events
remains unchanged. There is another variation of instance B, in which channel 3 is in ED state, channel 1 is in IE
state, and channel 2 is still in SR state. The next state of channel 1 will be IE in this case. The ED state and SR state
also cancel the propagation of each other.

Instances C and D: Similar to instance B.

Instance E: We can assume that the state of channel 2 is not ED and the state of channel 3 is not SR. Otherwise,
there will be no influence on the next state of channel 2 and 3 if we change the state of channel 1 to IE, then instance
E can be viewed as one of instances A through D. Therefore, the state of unit \( y \) should be NS other than SS without
the influence of channel 1. Now we consider the influence of unit \( y \) on the next state of channel 2 (The corresponding
contribution of channel 2 to the relative sum of empty events is listed in the parentheses immediately after the next
state):

(I) If state of unit \( x \) is NS, the state of channel 2 can only be IE, therefore, the next state of channel 2 is IE (0) if
the state of unit \( y \) is NS, and it is SR (-1) if the state of unit \( y \) is SS.

(II) If the state of unit \( x \) is SS, the state of channel 2 can be either IE or SR, so the next state of channel 2 is ED (1)
or IE (0) if the state of unit \( y \) is NS, otherwise it is IE (0) or SR (-1).

In both cases, it is evident that contribution of channel 2 to the relative sum decreases by 1 due to the change of the
state of unit \( y \) from NS to SS. Similarly, the contribution of channel 3 to the relative sum increases by 1. Therefore, The
ED state of channel 1 has no effect to the relative sum of empty events of the induced cycle, though it may influence
the next states of channel 2 and 3.

Instances F, G, and H: Similar to instance C.

All these instances show that the relative number of empty events does not change in any situation.

In the initial state, all output channels of relay stations are in ED state, and all other channels are in IE state,
therefore \( \delta(C,1) \) is equal to \( \Delta(C) \). Thus, the relative number of empty events at any time stamp \( t \) is always \( \Delta(C) \). \( \square \)
References


