韓國經營科學會誌
第29卷 第3號
2004年 9月

상수 서비스 시간을 갖는 개방형 대기행렬의 종대종 지연과 윈도우 구조의 영향 분석

이 영*

An Impact Analysis of Window Mechanism and End to End Delay of Tandem Queueing Network with Constant Service Times

Young Rhee*

■ Abstract ■

In this paper, we investigate the impact of window mechanisms on end to end delay in a series of nodes with constant service times. It is shown that arbitrary combinations of window mechanisms, each applying to an arbitrary subset of data, can be embedded on the nodes without affecting the departure instants from the system if the windows are at least as large as the number of nodes spanned. The window mechanisms are shown to have no impact on the average end to end delay of data. As the condition on the windows is a minimal necessary requirement for full parallelism, the results show that the transparent operation from viewpoint of data transmission can be achieved with minimal resources.

Keyword : Constant Service Times, End to End Delay, Window Mechanism

논문접수일 : 2003년 12월 3일 논문게재확정일 : 2004년 7월 31일
* 계명대학교 산업시스템공학과 조교수

1. Introduction

Communication network protocols are designed a layered structure along the line of the OSI model[2, 4]. A common means of implementing the required sequencing, pacing, congestion control, and data integrity functions of protocol layer is a multiple window mechanism, say a sliding window mechanism. As it were, the window mechanism is a means for regulating the flow of active customers through the service nodes in distributed networks. This mechanism is widely applied in data transmission network, as it is the standard level 2 data link flow control, and for level 3 end-to-end flow control. Even if the remote process call and the buffer management schemes are conventionally encountered and well modeled as the window mechanism. The word window in the term window mechanism refers to an extra buffer, created by both the sender and the receiver in the communication network. A window mechanism limits the total number of frames or packets within the set of nodes that it controls. Specifically, it blocks arriving frames in front of the node, preventing that node from accepting further work when the number of active frames reaches a predefined threshold. The threshold is usually called the window size or window. As active frames consume resources while waiting in the buffers, the window mechanisms simplify and reduce the cost of provision the service nodes because they limits the resources required to manage the frames.

The window mechanism has been extensively studied, especially in the context of communication networks. Reiser[8] modeled a computer communication system consisting of many virtual routes with end to end window flow control, as a closed multichain queueing network under the assumption of a loss system. Reiser [7] and Thomasian and Bay [12] use a flow equivalent server technique to model the sliding window link as a single server queue with state dependent service rate. In this approach, the effect of delays due to all sequence numbers in use is accounted for in the delivery service time of the equivalent server. Varghee, Chou and Nilsson [13] analyzed an open queueing network without an acknowledgment delay using approximation method. The performance of the mechanism under the nominal operation, and in the presence of stress conditions such as transmission errors, loss or the delay of frames are well fitted [2]. Fdida, Perros and Wilk [4] present a methodology for analyzing arbitrary configurations of sliding window controlled networks. Each layer of sliding window protocol is reduced to a state dependent infinite server queue without explicit acknowledgements, using a flow-equivalent methodology. A different approach proposed by Dallery [3] is about a single class open queueing network with general service times. The basic open model is transformed into an equivalent closed model. Also, a closed queueing network with general service times and population constrained subnetwork is analyzed [1].

Shapiro and Perros [11] present a hierarchical method for analyzing nested sliding window controlled layers. Each layer with sliding window protocol is reduced to a single queue without explicit acknowledgements. The analysis is also restricted to a single hop (no intermediate stations) connection. Rhee and Perros [9, 10] present a multilayered window flow control, where the population constraint within each subnetwork is

controlled by a semaphore queue. A major characteristic of multilayered window flow control is that the lower layer flow is halted by the state of the high layer. It is shown that the queueing network can be transformed into a simplified queueing network. A single-hop OSI structured network with multiple layers of sliding window flow control and packet fragmentation between layers is analyzed by Shapiro and Perros [11]. These types of queueing networks have applications in the manufacturing system. Liberopoulos and Dallery [6] analyzed a multi-stage manufacturing systems in which several production activities have been functionally aggregated into different production stage so that pull production control can be exercised in between stages.

However, due to the complexity of the interferences, the previous analysis generally does not carry to more complex, yet frequently encountered, system where multiple window mechanisms interfere with each other. The goal of this paper is to study the dynamics of the window mechanisms in such systems, with emphasis on the window choice. The primary concern of this paper is to analyze the impact of the window flow mechanisms on performance : that is, throughput and end-to-end delay under the typical conditions. It will not be concerned by performance under the exceptional circumstances such as data transmission errors or the loss of data. To minimize the resources with which the service nodes must be provisioned, the window flow mechanisms should have the smallest possible windows. The windows can not be so small as to interfere the full operation of the service nodes, since this may degrade the performance of the communication system. The selection of the windows is made through a detailed analysis of the internal dynamics of the system, accounting for the impact of the window flow mechanisms on the movements of data. As the window mechanisms strongly correlate the nodes that they control, this analysis is often not trivial. One must convert to Petri networks, blocking networks, or simulation to tackle the correlation between windows and performance instead of applying a typical queueing network analysis.

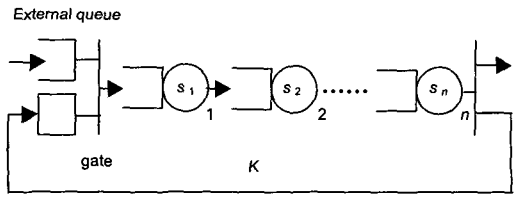
In this paper, we consider a system consisting of a series of sequentially accessed service nodes in which data have identical requirements at each node. For the reasons of simplicity, modularity and efficiency, this kind of system is frequently encountered, either independently or as a component in a complex communication system. Although window mechanisms may strongly influence where data or frames wait internally in the system, it is shown that they have no influence on their overall delay when the window size meet a certain condition, called the minimal condition. For engineering purposes, the condition allows to neglect the modeling of the window mechanisms without loss of accuracy, and also allows wide application of the simpler modeling tools. The condition states the window of every mechanism should be at least as large as the number of servers controlled by the mechanism. This minimal requirement to provide full resource possession is proved partially by Rhee and Perros [9]. This paper considers the applicability of internal dynamics of window mechanism in the wide communication network.

This paper is organized as follows. Section 2 presents the basic window mechanism, notation and assumption to develop network modeling. In section 3, we show some properties to provide the main results. Section 4 illustrates and ex-

tends the properties of section 3 as an example. Section 5 concludes with comments and suggestions for further interests.

2. A window model under the study

Let us consider a basic window mechanism as shown in [Figure 1]. Data or frames arrive at the external queue first. They access in sequence nodes, $1, 2, \cdots, n$ and leave. To join the first queue, every frame must grab a free token. Once they have one, they keep it until they leave the n^{th} queue. Frames at the external queue may grab tokens as soon as they are released by preceding frames departing from the final n^{th} node. The total number of tokens, called the window, is the control parameter of the mechanism.



[Figure 1] Basic window model

In general, many window mechanisms may be embedded on the series of nodes such as [Figure 1]. The mechanisms may differ on the location of their gate, on the point where their tokens are released, or on their windows. In addition, data may not all be regulated by the same mechanisms. Data may be discriminated based on some attributes, and different data may be regulated on the other way. The scheduling disciplines at the external queue and at each queue are work-conserving and non-preemptive. This means that free token can not be withheld when there are data waiting at the external queue, the server can not be idle when there are data waiting and the processing of data can not be interrupted once started. The time for grabbing a token is negligible at the external queue. The service time at each node is the same for all data, which means a constant service time. However, the service times may differ for different nodes.

Let K be the windows in [Figure 1]. The identity of the i^{th} data arriving to the system is denoted c_i, its arrival time, a_i. Finally, s_j denotes the service time at the j^{th} node, $j = 1, 2, \cdots, n$. It is assumed that a_i may explicitly depend only on the departure instants from the system up to the arrival of c_i. This is consistent with the view that whatever generates a_i sees the system as a blackbox. It has no knowledge of the internal dynamics of the system, and hence cannot make a_i depend on it. However, it can observe the departure process, and make a_i depend explicitly on the departure instants. Note that the assumption is still somewhat restrictive because the dependancy may only be through the departure instants, but not the particular identities of the data.

3. Some properties on transparent operation.

In this section, we prove a sufficient condition for achieving some sort of transparent operations from an input and output point of view.

Property 1 : Let us consider a series of nodes with embedded window mechanisms All assumptions are already defined in section 2. The arrangement of window mechanisms is arbitrary. The subset of the window mechanisms that

apply to each data is also arbitrary, and may differ for different arriving data. Let m be the node whose service time is the largest among the all nodes. Let d_i be the departure time of the i^{th} data. After the node m, the data do not wait, neither at the node nor at the server.

proof : By induction on the departure instants, we assume that the first $(i-1)$ departure instants are independent of the specific windows. It is shown that this holds for the i^{th} departure instant. Given the assumptions of the proposition, it is readily established that the data which arrives first is the one which depart first. This is readily verified for c_1. Assume by induction that it holds for the first $(j-1)$ data which departs from the node m. Then, these data are respectively $c_1, c_2, \cdots, c(j-1)$. Let c be the data which leaves the node m after $c_{(j-1)}$. When c leaves the node m, all external queues after the node m are empty and all noders after the node m contains at most one data. In addition, $c_{(j-1)}$ departs from the node $(m+1)$ is no later than when c departs from the node m, since the minimum inter-departure time from the node m is at least as large as the service time at the node $(m+1)$.

Suppose that there is a window mechanism whose gate is in front of node $(m+1)$ and that is applied to c. As the window is at least as large as the number of nodes spanned, as all external queues and node $(m+1)$ are empty and as node $(m+2)$, $(m+3)$, $\cdots$, and n contain at most 1 data, c must find a free token of the mechanism immediately upon departing from the node $(m+1)$. Since this relationship applies generically to all window mechanisms whose gate is in front of the node $(m+1)$ and that applies to c, it follows that c starts service at the node $(m+1)$ immediately upon departing from the node m. This is the reason that c does not wait at the node $(m+1)$ nor at any external queue between the node m and the node $(m+1)$.

If $c_1, c_2, \cdots, c_{(j-1)}$ and c do not wait between the node m and the node $(m+1)$, their minimum inter-departure time from the node $(m+1)$ is the same as for the node m, and c is the next data to depart from the node after $c_{(j-1)}$. From this observation, the argument can be repeated to show that c does not wait between its departure from the node $(m+1)$ and the node $(m+2)$, and so no for all subsequent nodes. Therefore, c does not wait after the node m and the minimum inter-departure time at the node m, $(m+1)$, $\cdots$, n is s_m. And, c_k, $k=1,2,\cdots,i-1$joins the node q at time $d_{(i-1)}-(i-k-1)s_m - \sum_{j=q}^{n} s_j$. This is because c_k departs from the node n at least $(i-k-1)s_m$ before $c_{(i-1)}$, and enters the node q at least that time minus all the service times from the node q to the node n. ■

Property 2 : If the size of every window mechanisms is at least as large as the number of tokens spanned by the mechanism, then the departure instants from the system are independent of the specific values of the windows.

proof : Let assume that there is a window mechanism of size K whose gate is in front of the node q and whose tokens are released in front of the node r. If the tokens are released after the node n, then set $r=n+1$. By assuming $i>K$, we have to prove that the first to release its token does so at most at time $d_{(i-1)}-(K-1)s_m - \sum_{j=r}^{n} s_j$ among the K data in $c_1, c_2, \cdots, c_{(i-1)}$

who last had a token. In the worst case for the bound, the K data who last had a token among $c_1, c_2, \cdots, c_{(i-1)}$ are $c_{(i-W)} \cdots c_{(i-1)}$. The $(i-w)^{th}$ data, $c_{(i-W)}$ joins the node r at most at time $d_{(i-1)} - (K-1)s_m - \sum_{j=r}^{n} s_j$, and must have relinquished its token by then. Two cases are distinguished depending if $a_i \geq d_{(i-1)} - s_m - \sum_{j=1}^{n} s_j$, and hence must have relinquished its token by then. Two cases can be classified depending on if $a_i \geq d_{(i-1)} + s_m - \sum_{j=1}^{n} s_j$ or not.

• **Case** : $a_i \geq d_{(i-1)} + s_m - \sum_{j=1}^{n} s_j$

From the case assumption, we need to prove that $d_i = a_i + \sum_{j=1}^{n} s_j$, independent of the specific windows. It may first noted that c_i joins the node q, $q = 1,2,\cdots,n$ at the earliest at time ;

$$a_i - \sum_{j=1}^{q-1} s_j \geq d_{(i-1)} + s_m - \sum_{j=q}^{n} s_j \qquad (1)$$

And, $c_1, c_2, \cdots$, and $c_{(i-1)}$ have by then departed the node q. Suppose there is a window mechanism whose gate is in front of the node 1 and whose tokens are released in front of the node r. And suppose further that the size of its window is K and that it applies to c_i. Noting that a token for c_i is initially available if $i \leq K$and a token for c_i must become available at most at time $max\{0, d_{(i-1)} - (K-1)s_m - \sum_{j=r}^{n} s_j\}$. By assumption, $K \geq r-1$, i.e., the window size at least as large as the number of nodes spanned, this time is less than the right hand side of (1). This implies that the i^{th} data, c_i finds a free token of the mechanism immediately upon arrival.

As the window mechanism is generic, it follows that the i^{th} data, c_i finds a free token of every window mechanism whose gate is in front of the node 1. Hence, c_i joins the first node immediately upon arrival. As $c_1, c_2, \cdots, c_{(i-1)}$ are gone upon its arrival and enter into the service. Accordingly, c_i leaves the first node at time $a_i + s_1$. Repeating this argument, it is shown that c_i never waits at the external queue nor at a server when the condition are met. It follows that the i^{th} departure instant is that of c_i, and that $d_i = a_i + \sum_{j=1}^{n} s_j$, independent of the specific windows. ■

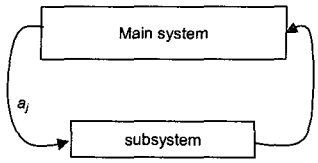
• **Case** : $a_i < d_{(i-1)} + s_m - \sum_{j=1}^{n} s_j$

The objective of the second case is to prove that $d_i = d_{(i-1)} + s_m$, independent of specific windows. Assume there is a window mechanism whose gate is in front of the first node and whose tokens are released in front of the node r. Further assume that the size of its window is K. For the same reason as shown the above, a token of the mechanism becomes available to one of the data in $c_i, \cdots$ at most at time $max(0, d_{(i-1)} - (K-1)s_m - \sum_{j=r}^{n} s_j)$ which, as $K \geq r-1$, is no greater than $d_{(i-1)} + s_m - \sum_{j=1}^{n} s_j$. As the window mechanism is generic, it follows that a token of every window mechanism whose gate is in front of the first node is available to one of the data in $c_i, \cdots$ at most by time $d_{(i-1)} + s_m - \sum_{j=1}^{n} s_j$.

Now, by the case assumption, c_i and possibly many following data are arrived by time $d_{(i-1)}$

$+s_m-\sum_{j=1}^{n} s_j$, and at least one of these data must be among $c_i, \cdots$. Hence, at least one data among $c_i, \cdots$, must by then at the first node and $c_i, \cdots, c_{(i-1)}$ have left the first node by time $d_{(i-1)}-\sum_{j=1}^{n} s_j$. It follows that the i^{th} departure from the first node occurs no later than $d_{(i-1)}+s_m-\sum_{j=2}^{n} s_j$. Repeating this argument, it is readily established that the i^{th} departure from the node q, $q-1, \cdots, n$, occurs at most at time $d_{(i-1)}+s_m-\sum_{j=q+1}^{n} s_j$, and hence that $d_i \geq d_{(i-1)}+s_m$. It follows that $d_i = d_{(i-1)}+s_m$, independent of the specific windows as long as the condition is satisfied. In addition to the assumption of Property 2, let assume that the queueing discipline at every external queue is FCFS(first come first service) and the flow conservation is secured. Then as long as the window of every window mechanism is at least as large as the number of nodes spanned the mechanism, the end to end delay of every data is independent of the specific value of the windows. The above statement can be proved that the arriving data leave the system in the same order. Therefore, the departure instants do not vary and the end to end delay of every data does not changed. ■

Corollary 1 : Let consider a system which can be divided into the main system and the subsystem in [Figure 2]. Then, as long as the window of every window mechanism in the subsystem is at least as large as the number of nodes spanned by the mechanism, the operation of the main system is independent of the specific values of the windows in the subsystem.



[Figure 2] Decomposed system

proof : The proof is closely related with an application of Property 2.

In the subsystem, the arrival process needs not to be independent from the departure process. The dependency may only be on the departure instants but not on the identity of the departing data. Corollary 1 asserts that even if a complex system does not satisfies the conditions of Property 2, its modeling can be simplified if a subsystem satisfies the conditions. Then, without loss of accuracy, the performance of the system can be analyzed assuming arbitrarily large windows in the subsystem. Otherwise, the window mechanisms in the subsystem can be neglected.

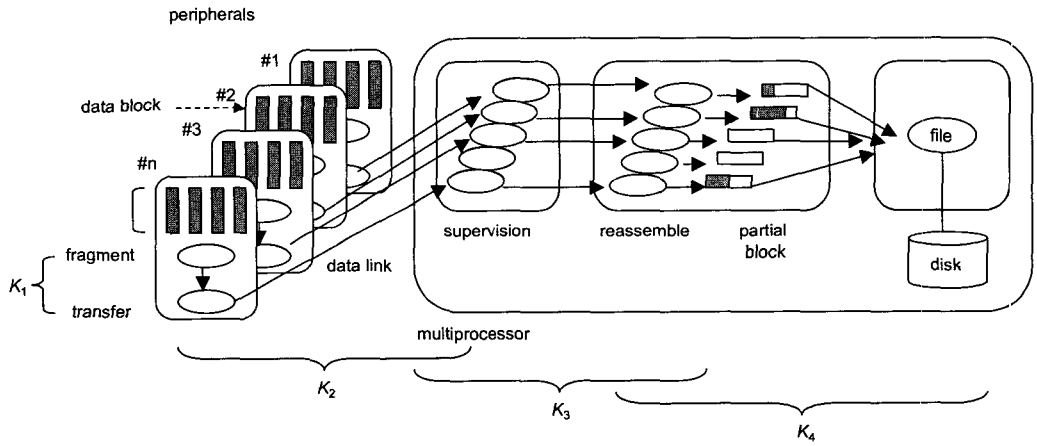
4. Numerical examples

In this section, it is shown that the results of the preceding section by means of an example. The example is a data collection system where large blocks of data are transferred from n peripherals to a multiprocessor as shown in [Figure 3]. Generally, this kinds of models are applied to the WAN standards such as X.25 and frame relay. Due to their sizes, the blocks are first fragmented into small identical units, $u \geq 1$ and then transmitted. Communication from each peripheral to the multiprocessor is used over the dedicated WAN link.

In the multiprocessor, the units of a particular peripheral are received at the WAN processor by

a dedicated supervision task. After ensuring that they have been properly transmitted, the supervision task forwards the units to the reassembly processor. The units are reassembled into data block with some intermediate formatting and processing. Each peripheral has a dedicated reassembly task which is responsible for the reconstruction of its data blocks. When a block is completely reconstructed, the reassembly task is responsible for its transfer to the file server. At the file server, the block is received by the file task which writes it to the disk and discards it.

There are no end to end flow control mechanisms, i.e., from the peripherals to the disk, to explicitly prevent the peripherals from swamping the multiprocessor with data. Instead, several window mechanisms implicitly regulate the overall flow by locally regulating the flow of the blocks and units in each element. In [Figure 3], K_2 is the level 3 flow control mechanism. And also, K_1, K_3 and K_4 limit the number of blocks and units respectively in the peripherals, the reassembly processor and the file server. The tokens of these mechanisms can be viewed as buffer.



[Figure 3] Location of the window mechanisms

The details of the window mechanisms are given below, To emphasize the distinction between the flow control at the block and the unit level, we explicitly associate block and unit tokens with the mechanism.

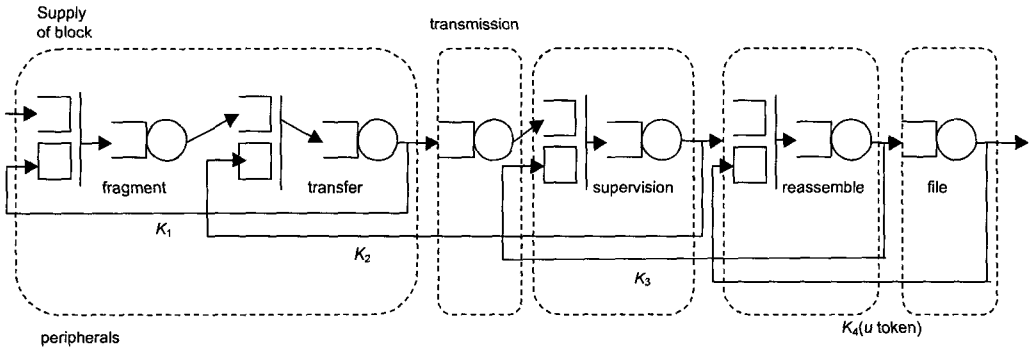
K_1 : When a block is fragmented, its block token is given to the last unit to be transferred. When the transfer of the last unit occurs, the block token of K_1 is released.

K_2 : Each instance limits the number of units which can be transferred to the processor before an acknowledgement must be received. The units must be completely processed by the processor before they are acknowledged, and that time for acknowledgement message to return is negligible.

K_3 : Each unit must hold a token of the instance of K_3 corresponding to its peripheral to be processed at the processor. The unit frees its token upon leaving the reassembly processor.

K_4 : When a block is completely reassembled and processed by the file server, all the unit tokens of K_4 held by its units are released.

The windows must be specified to determine the minimal amount of memory for the data that the processors need. To help in this regard, the model has been redrawn in [Figure 4] with the window mechanisms represented in [Figure 1]. To satisfy the requirement of one task per processor, it is assumed that the processing associated with the transferred task on the peripherals is small compared to the processing required for the fragmented task. The processing of the transferred task is ignored, and the transferred task is only considered with respect to the synchronization that it introduces in the acquisition and in the release of tokens.



[Figure 4] Model for window mechanisms

〈Table 1〉 Simulation results for window mechanisms

simulation #	block I.D.	given window	infinite window
end to end delay for 1 peripheral	20	53.63	53.63
	40	61.37	61.37
	60	70.08	70.08
	80	85.17	85.17
	100	99.18	99.18
end to end delay for 3 peripherals	···	70.17	70.17

For the numerical example, we assume first that $n=1$, $u=1$ and all processing times are constant. Property 1 and 2 can be used to determine windows which achieve transparent operation. It states that $K_1=1$ is sufficient, since K_1 spans only one processor. For the same reason, K_2, K_3 and K_4 is set to 2. To verify that the transparent operation is achieved, the system shown in [Figure 3] is simulated and the parameters of the simulations and the outputs are given in <Table 1>.

The arrival process is poisson with rate 1/12.5 and the processing time for fragment = 5, data link = 6, supervision = 7, reassemble = 8 and file = 10. The parameters are chosen to ensure that all window mechanisms are frequently activated, and to reasonably load the system to 50% to 80%. The outputs are the end to end delays for transferring several arbitrarily chosen data blocks. Under transparent operation, the end to end delays should be the same in both systems. The same systems are simulated but with the arrival process splits into three peripherals and the

average delay should be the same in the systems with finite and with infinite windows.

〈Table 2〉 simulation results for $u = 5$ and with the processing times adjusted so as to generate the same overall loads

Block I.D.	end to end delay	
	given window	infinite windows
20	41.23	41.23
40	50.97	50.97
60	64.68	64.68
80	75.31	75.31
100	88.78	88.78

The second example is suggest to compare with the windows when $n=1$ and $u>1$. Each window could be heuristically chosen to be at least sufficiently large to allow the controlled elements to fully work locally in parallel and has a different dynamics. <Table 2> presents the results of simulations of the system using the same model of <Table 1>, but with $u=5$ and with the processing times adjusted so as to generate the same overall loads. The processing time is set to data link = 1.2, supervision = 1.4, reassemble = 1.6 . Clearly, the end to end delays are still independent of specific windows. The last comment within the context of the example concerns the impact of the service time distributions. So far, it has been assumed that all service times are constant. To see what happens if they are not, the data collection system which produced <Table 2> is simulated. The results are summarized in <Table 3>. The first observation is that the choice of windows made in <Table 2> does not lead anymore to transparent end to end delays. However, <Table 3> shows that the delays converge rapidly as the window size are increased. This suggests that although the variability of the service times does not impact delay when window mechanisms regulate the flow, the impact rapidly becomes negligible as the windows increase.

〈Table 3〉 convergency results

Block I.D.	end to end delay			
	additional tokens			
	0	1	2	∞
20	60.48	57.48	55.49	55.33
40	99.14	89.11	86.77	86.61
60	106.33	89.70	89.36	89.19
80	134.7	90.57	90.07	89.91
100	178.6	105.8	103.3	103.1

5. Conclusion

It has been shown that in a series of nodes with constant service times, arbitrary combinations of window mechanisms, each applying to an arbitrary subset of data, can be embedded on the servers without affecting the departure instants from the system if the windows are at least as large as the number of nodes spanned. The result implies that the end to end delay of every data is independent of the specific windows. Under general non preemptive work conserving scheduling, the result implies that the average end to end delay of data is independent of the specific windows. As the condition on the windows is a minimal necessary requirement for full parallelism, the results show that transparent operation can be achieved with minimal resources.

With variable service times, the results do not strictly hold, but transparent operation seems to be rapidly achieved as the windows are increased above the minimal requirements. In addition, to

increase the windows is not the only means to achieve full parallelism. This paper suggests that, although window mechanisms affect internal dynamics, they have a weak impact on overall performance when minimal conditions are met.

REFERENCES

[1] Baynat, B. and Dallery, Y., "On product form approximation techniques for general closed queueing networks," *Tech. Rep. MASI*, 1987.

[2] Bertsekas, D. and Gallager, R., *Data Networks*, Prentice-Hall, 1992.

[3] Dallery, Y., "Approximate analysis of general open queueing networks with restricted capacity," *Tech. Rep., LAG*, N 87-08, 1987, revised Feb. 1989.

[4] Fdida, S., Perros, H. and Wilk, A., "Semaphore queue: modeling multilayered window flow mechanisms," *IEEE Trans. on comm.*, Vol.38, No.3(1990), pp.309-317.

[5] Kant, K., *Introduction to computer performance evaluation*, McGraw-Hill, New York, 1992.

[6] Liberropoulos, G. and Dallery, Y., "A unified framework for pull control mechanism in multi-stage manufacturing systems," *Annals of OR.*, Vol.93(2000), pp.325-355.

[7] Reiser, M., "A queueing network analysis of computer communication networks with window flow control," *IEEE Trans. on comm.*, Vol.23(1975), pp.1199-1209.

[8] Reiser, M., "Performance evaluation of data communication systems," *Proceeding of the IEEE*, Vol.70-2(1982), pp.171-182.

[9] Rhee, Y. and Perros, H., "Analysis of an open tandem queueing network with population constraint and constant service times," *European Journal of O.R.*, Vol.92 (1996), pp.99-111.

[10] Rhee, Y. and Perros, H., "On the mean waiting time of a population constrained open tandem queueing network with constant service times," *IIE Transaction*, Vol.30(1998), pp.973-979.

[11] Shapiro, G. and Perros, H., "Nested Sliding window protocols with packet fragmentation," *IEEE Trans. on comm.*, Vol.41, No.1 (1993), pp.99-109.

[12] Thomasian, A. and Bay, P., "Analysis of queueing network models with population size constraints and delayed block customers," *ACM SIGMETRICS* conf., Cambridge, 1984, pp.202-216.

[13] Varge, G., Chou, W. and Nilsson, A., "Queueing delays on circuits using a sliding window flow control scheme," *ACM SIGMETRICS* conf., Minneapolis, 1983, pp.275-281.