ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

PhD THESIS

Çiğdem ACI

DEVELOPMENT OF NEW CONGESTION CONTROL ALGORITHMS FOR BROADCAST BASED MULTIPROCESSOR ARCHITECTURES WITH MULTIPLE INPUT QUEUES

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ADANA, 2013

ABSTRACT

PhD THESIS

DEVELOPMENT OF NEW CONGESTION CONTROL ALGORITHMS FOR BROADCAST BASED MULTIPROCESSOR ARCHITECTURES WITH MULTIPLE INPUT QUEUES

Çiğdem ACI

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

Supervisor : Asst.Prof.Dr. M. Fatih AKAY
Year: 2013, Pages: 123
Jury : Asst.Prof.Dr. M. Fatih AKAY
: Assoc.Prof.Dr. Ramazan ÇOBAN
: Assoc.Prof.Dr. Mustafa GÖK
: Assoc.Prof.Dr. S. Ayşe ÖZEL
: Assoc.Prof.Dr. A. Bedri ÖZER

In this thesis, a congestion control algorithm to improve 64-node, 2-Dimensional (2-D) Simultaneous Optical Multiprocessor Exchange Bus (SOME-Bus) performance is proposed. The congestion control algorithm is tested via simulation under several synthetic traffic patterns using Optimized Network Engineering Tool (OPNET) Modeler. Performance measures such as average processor utilization, average channel waiting time, average input waiting time and average network response time have been collected before and after applying the algorithm. For Client-Server traffic model, the proposed algorithm is able to increase the average processor utilization, between 1.59% and 12.57% with 4 threads, and between 1.21% and 13.91% for 8 threads; decrease the average channel waiting time between 3.47% and 16.17% with 4 threads, and between 6.18% and 25.02% for 8 threads; the average input waiting time between 2.41% and 19.29% with 4 threads, and between 4.38% and 27.85% for 8 threads; the average network response time between 2.85% and 21.07% with 4 threads, and between 4.99% and 30.27% for 8 threads. For Asynchronous Message Passing traffic model, the proposed algorithm is able to increase the average processor utilization between 1.86% and 13.75% with 4 threads, and between 1.04% and 13.91% for 8 threads; decrease the average channel waiting time between 4.26% and 18.07% with 4 threads, and between 10.55% and 29.53% for 8 threads; the average input waiting time between 4.63% and 22.06% with 4 threads, and between 5.67% and 30.22% for 8 threads; the average network response time between 17.53% and 44.71% with 4 threads, and between 31.16% and 52.09% for 8 threads.

Key Words: 2-Dimensional Multiprocessor Architectures, Congestion Control, Simulation, Optical Interconnects

ÖZ

DOKTORA TEZİ

BİRDEN FAZLA GİRİŞ KUYRUĞUNA SAHİP YAYIM TABANLI VE ÇOKLU MİKROİŞLEMCİLİ MİMARİLER İÇİN YENİ TIKANIKLIK KONTROLÜ ALGORİTMALARININ GELİŞTİRİLMESİ

Çiğdem ACI

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

Danışman : Yrd.Doç.Dr. M. Fatih AKAY
Yıl: 2013, Sayfa: 123
Jüri : Yrd.Doç.Dr. M. Fatih AKAY
: Doç.Dr. Ramazan ÇOBAN
: Doç.Dr. Mustafa GÖK
: Doç.Dr. S. Ayşe ÖZEL
: Doç.Dr. A. Bedri ÖZER

Bu tezde, 64 düğümlü SOME-Bus ağının performansını arttıracak bir tıkanıklık kontrolü algoritması önerilmiştir. Tıkanıklık kontrolü algoritması, benzetim metodu ve OPNET Modeler kullanılarak birçok yapay trafik modeli altında test edilmiştir. Ortalama işlemci verimi, kanal kuyruğunda ortalama bekleme süresi, giriş kuyruğunda ortalama bekleme süresi ve ortalama ağ cevap zamanı gibi performans kriterleri, algoritmadan önce ve sonra olmak üzere iki durum için de ölçülmüştür. Sunucu-İstemci trafik modelinde önerilen algoritma ortalama işlemci verimini, 4 adet iş parçacığı kullanıldığında, %1.59 ile %12.57; 8 adet iş parçacığı kullanıldığında, %1.21 ile %13.91 arasındaki oranlarla yükseltmiş olup; kanal kuyruğunda ortalama bekleme süresini ise 4 adet iş parçacığı kullanıldığında, %3.47 ile %16.17; 8 adet iş parçacığı kullanıldığında, %6.18 ile %25.02 arasındaki oranlarla düşürmüştür. Giriş kuyruğunda ortalama bekleme süresini 4 adet iş parçacığı kullanıldığında, %2.41 ile %19.29; 8 adet iş parçacığı kullanıldığında, %4.38 ile %27.85; ortalama ağ cevap zamanını ise 4 adet iş parçacığı kullanıldığında, %2.85 ile %21.07; 8 adet iş parçacığı kullanıldığında, %4.99 ile %30.27 arasındaki oranlarla düşürmüştür. Asenkron Mesaj Geçişi trafik modelinde ise önerilen algoritma ortalama işlemci verimini, 4 adet iş parçacığı kullanıldığında, %1.86 ile %13.75; 8 adet iş parçacığı kullanıldığında, %1.04 ile %13.91 arasındaki oranlarla arttırmış olup; kanal kuyruğunda ortalama bekleme süresini ise 4 adet iş parçacığı kullanıldığında, %4.26 ile %18.07; 8 adet iş parçacığı kullanıldığında, %10.55 ile %29.53 arasındaki oranlarla azaltmıştır. Giriş kuyruğunda ortalama bekleme süresini 4 adet iş parçacığı kullanıldığında, %4.63 ile %22.06; 8 adet iş parçacığı kullanıldığında, %5.67 ile %30.22; ortalama ağ cevap zamanını ise 4 adet iş parçacığı kullanıldığında %17.53 ile %44.71; 8 adet iş parçacığı kullanıldığında, %31.16 ile %52.09 arasındaki oranlarla azaltmıştır.

Anahtar Kelimeler: 2 Boyutlu Çoklu İşlemcili Mimariler, Tıkanıklık Kontrolü, Benzetim, Optik İnterkoneksiyonlar.

ACKNOWLEDGEMENTS

I am happy to have the opportunity to thank people who have helped me throughout this study.

The biggest share of my gratitude is reserved for my supervisor, *Assist.Prof. Dr. Mehmet Fatih AKAY*, for patience and help he provided to me. I will always feel lucky for having been able to work with such a distinguished scholar.

I would like to acknowledge my committee members *Assoc.Prof.Dr. Ramazan ÇOBAN, Assoc.Prof.Dr. Mustafa GÖK, Assoc.Prof.Dr. Selma Ayşe ÖZEL* and *Assoc.Prof.Dr. Ahmet Bedri ÖZER* for their time and suggestions.

I would like to thank to *Dr. Constantine KATSINIS* from Drexel University (Philadelphia, USA) for his permission about using figures and information of the SOME-Bus multiprocessor architecture.

I would also like to thank OPNET Tech. Inc. for letting me use the OPNET Modeler as a simulation environment.

I would like to express my thanks to *Hasan AHKEMOĞLU* for his feedbacks and corrections.

I am grateful to all staff members in the *Department of Computer Engineering* and *Electrical and Electronics Engineering*, *Çukurova University*, for the constant support and encouragement they provided.

I owe special thanks to all scholars in the *Department of Computer Engineering, Fırat University,* who have made great contributions to my professional and personal development.

Special thanks to my family members: my mother *Esma İNAN* and my dear brother *Savaş İNAN* for their endless love, support and patience.

Finally, I would like to express my deepest gratitude to my husband *Mehmet ACI* who has always supported, encouraged and believed in me, in all my endeavors.

CONTENTS **PAGE**

LIST OF TABLES **PAGE**

LIST OF FIGURES **PAGE**

LIST OF ABBREVIATONS

1-D	: One-dimensional
2-D	: Two-dimensional
BR	: Bit reverse
C	: Channel
def.	: Default
dest_addr	: Destination address
dest_cnt	: Destination count
dest_input_q_no	: Destination input queue number
dest_ptr_pos	: Destination pointer position
H	: Horizontal bus
HPC	: High performance computing
HR	: Hot region
inter_node	: Intermediate nodes
K	: Number of threads in the system
N	: Node number for 1-D SOME-Bus
N^2	: Node number for 2-D SOME-Bus
node_addr	: Node address
OPNET	: Optimized network engineering tool
P	: Processor
pk_pass_fld	: Packet pass field
R	: Mean thread run time
recv_op	: Receive operation
send_op	: Send operation
SOME-Bus	: Simultaneous optical multiprocessor exchange bus
src_output_q_pk_cnt	: Source output queue packet count
svc_cmp	: Service complete
svc_completion	: Service completion
svc_strt	: Service start

T	: Message transfer time
UN	: Uniform
V	: Vertical bus

1. INTRODUCTION

1.1. Components of High Performance Computing

Parallel computing is the method of breaking large problems down into smaller constituent components, tasks or calculations that are solvable in parallel. Parallel computing has emerged as a key enabling technology in modern computing (Geist et al., 1994). The use of parallel computing is essential for solving practical problems in science and engineering. Parallelism is a way of speeding up computations which make high time and memory demands. Some important domains for parallel computing include scientific applications that model physical phenomena; engineering applications such as those in computer-aided design, digital signal processing, automobile crash simulation and even simulations used to evaluate architectural tradeoffs; graphics and visualization applications that render scenes or volumes into images; optimization applications such as crew scheduling for an airline and transport control; artificial intelligence applications such as expert systems and robotics; multi-programmed workloads; and the operating system itself, which is a particularly complex parallel application (Culler et al., 1998). Parallel computing involves three interactive disciplines as shown in Figure 1.1. Theoretical scientists develop mathematical models that computer engineers solve numerically; the numerical results may then suggest new theories (Morrison, 2003).

High performance computing (HPC) is an evolving field that usually involves the use of parallel computing systems to solve difficult computational problems (Park et al., 2007). HPC systems are computer systems consisting of multiple processing units connected via some interconnection network plus the software needed to make the processing units work together. With ever-increasing computing power demands for ultrascale applications, the HPC community has been deploying modern computing systems with increasing size and complexity (Huang et al., 2007). The unquenchable desire of today’s scientists to run ever larger simulations and analyze ever larger data sets drives the size of high performance computers from hundreds, to thousands, and even tens of thousands of processors (Chen et al., 2005).

The rapidly increasing number of processors in modern microprocessors is pushing the current HPC systems into the petascale and exascale era (Jin et al., 2011).

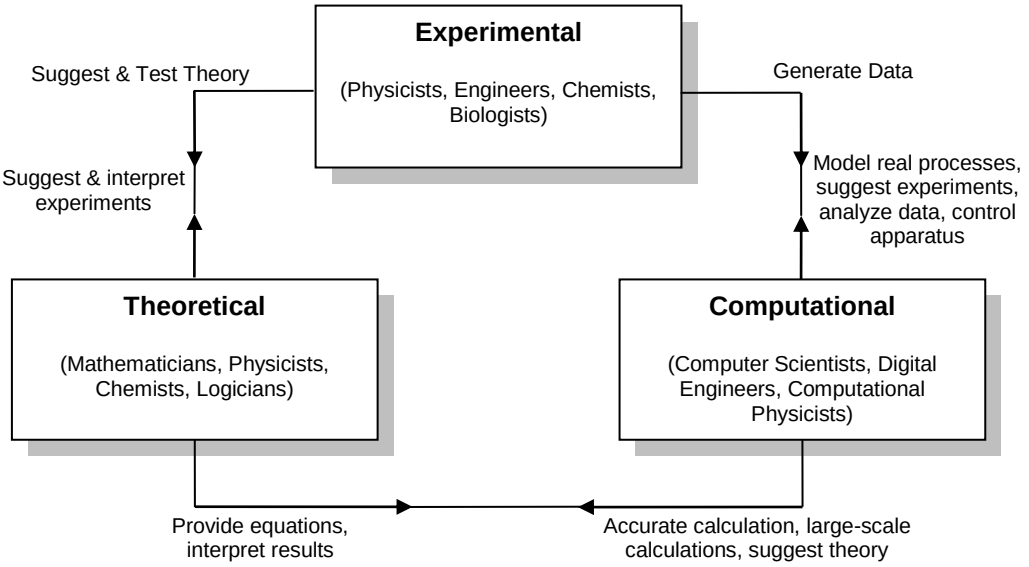

Figure 1.1. Interaction among Experiment, Theory and Computation (Morrison, 2003)

HPC systems require some kind of communication subsystem to interconnect processors, memories, disks, and other peripherals. An interconnection network is a programmable system that transports data between terminals. They are used in almost all digital systems that are large enough to have two components to connect. The most common applications of interconnection networks are in computer systems and communication switches. In computer systems, they connect processors to memories and input/output devices to input/output controllers. They connect input ports to output ports in communication switches and network routers. They also connect sensors and actuators to processors in control systems. Anywhere that bits are transported between two components of a system, an interconnection network is likely to be found. The interconnection network between processor and memory largely determines the memory latency and memory bandwidth, two key performance factors, in a HPC system (Dally and Towles, 2004). The goal of the interconnection network is to serve as the communications infrastructure for transferring data among the growing massive numbers of processing and memory

elements. The performance of advanced computing systems clearly relies upon the efficiency and capabilities of the interconnect (Hennessy and Patterson, 2006).

An interconnection network could be either static or dynamic (Figure 1.2). Connections in a static network are fixed links, while connections in a dynamic network are established on the fly as needed. Static networks can be further classified according to their interconnection pattern as one-dimensional (1-D), two-dimensional (2-D), or Hypercube. Dynamic networks, on the other hand, can be classified based on interconnection scheme as bus-based versus switch-based. Switch-based dynamic networks can be classified according to the structure of the interconnection network as single-stage, multistage, or crossbar networks (El-Rewini and Abd-El-Barr, 2005).

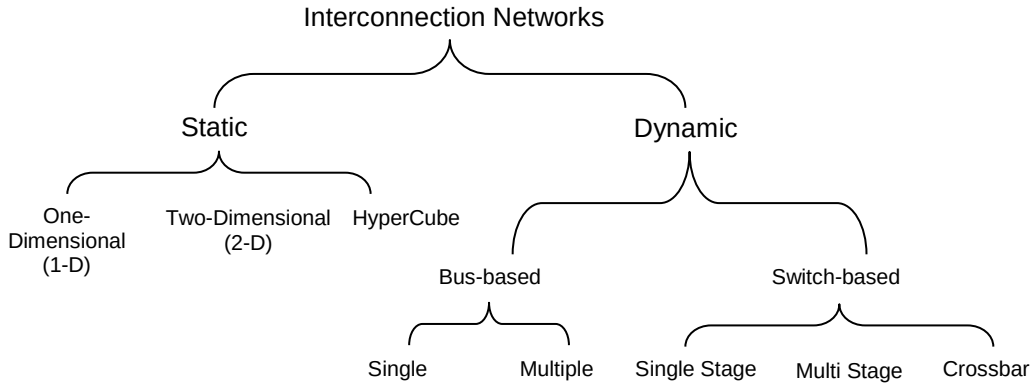

Figure 1.2. A topology-based taxonomy for interconnection networks (El-Rewini and Abd-El-Barr, 2005)

Bus-based networks can further be classified as single bus or multiple buses. A single bus is considered the simplest way to connect multiprocessor systems. Figure 1.3 shows an illustration of a single bus system. In its general form, such a system consists of N processors, each having its own cache, connected by a shared bus. The use of local caches reduces the processor–memory traffic. All processors communicate with a single shared memory. Although simple and easy to expand, single bus multiprocessors are inherently limited by the bandwidth of the bus and the fact that only one processor can access the bus, and in turn only one memory access can take place at any given time (El-Rewini and Abd-El-Barr, 2005).

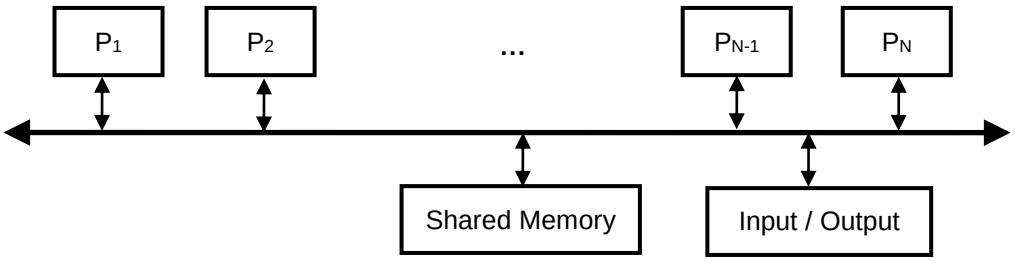

Figure 1.3. A single bus system (P=Processor, N=number of processors) (El-Rewini and Abd-El-Barr, 2005)

The use of multiple buses to connect multiple processors is a natural extension to the single shared bus system. A multiple bus multiprocessor system uses several parallel buses to interconnect multiple processors and multiple memory modules. A number of connection schemes are possible in this case. Among the possibilities are the multiple bus with full bus–memory connection (Figure 1.4), multiple bus with single bus memory connection, multiple bus with partial bus–memory connection, and multiple bus with class-based memory connection. In general, multiple bus multiprocessor organization offers a number of desirable features such as high reliability and ease of incremental growth (El-Rewini and Abd-El-Barr, 2005).

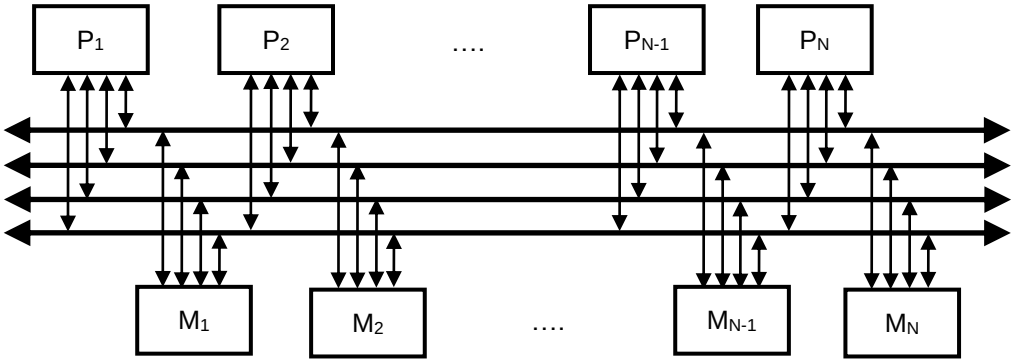

Figure 1.4. Multiple bus with full bus–memory connection (El-Rewini and Abd-El-Barr, 2005)

Many interconnection network topologies have been proposed in terms of their graph-theoretical properties. However, very few of them have ever been implemented. Most of the implemented networks have an orthogonal topology. A

network topology is orthogonal if and only if nodes can be arranged in an orthogonal N-dimensional space, and every link can be arranged in such a way that it produces a displacement in a single dimension. Orthogonal topologies can be further classified as strictly orthogonal and weakly orthogonal. In a strictly orthogonal topology, every node has at least one link crossing each dimension. In a weakly orthogonal topology, some nodes may not have any link in some dimensions. Hence, it is not possible to cross every dimension from every node. Crossing a given dimension from a given node may require moving in another dimension first. Direct and in-direct network topologies have been proposed for this reason. Each link in direct networks, traverses a single dimension and every node has at least one link crossing each dimension, the distance between two nodes can be computed as the sum of dimension offsets. Also, the displacement along a given link only modifies the offset in the corresponding dimension. Taking into account that it is possible to cross any dimension from any node in the network, routing can be easily implemented by selecting a link that decrements the absolute value of the offset in some dimension. Several examples of parallel computers with direct networks can be given as (Duato et al., 1997): Cray T3E (Anderson et al., 1997), Cray T3D (Kessler and Schwarzmeier, 1993), Chaos Router (Konstantinidou and Snyder, 1994), Intel iPSC-2 Hypercube (Bomans and Roose, 1989).

Indirect or switch-based networks are another major class of interconnection networks. Instead of providing a direct connection among some nodes, the communication between any two nodes has to be carried through some switches. Each node has a network adapter that connects to a network switch. Each switch can have a set of ports. Each port consists of one input and one output link. A (possibly empty) set of ports in each switch is either connected to processors or left open, whereas the remaining ports are connected to ports of other switches to provide connectivity between the processors. The interconnection of those switches defines various network topologies. Similar to direct networks, an indirect network is mainly characterized by three factors: topology, routing, and switching. The topology defines how the switches are interconnected by channels and can be modeled by a graph. For indirect networks with N nodes, the ideal topology would connect those

nodes through a single N × N switch. Such a switch is known as a crossbar. Crossbar networks allow any processor in the system to connect to any other processor or memory unit so that many processors can communicate simultaneously without contention. A new connection can be established at any time as long as the requested input and output ports are free. Crossbar networks are used in the design of high-performance small-scale multiprocessors, in the design of routers for direct networks, and as basic components in the design of large-scale indirect networks. Several examples of parallel computers with indirect networks and commercial switches to build indirect networks can be given as (Duato et al., 1997): Myricom Myrinet (Pakin et al., 1995), IBM SP2 (Hwang et al., 1996), SGI SPIDER (Galles, 1997).

The processing units in an interconnection network can communicate and interact with each other using either shared-memory or distributed-memory as shown in Figure 1.5. Shared-memory systems typically accomplish interprocessor coordination through a global memory shared by all processors. These are typically server systems that communicate through a bus and cache memory controller. The bus/cache architecture alleviates the need for expensive multiported memories and interface circuitry as well as the need to adopt a message-passing paradigm when developing application software. Each processor has equal opportunity to read/write to memory, including equal access speed. A number of basic issues in the design of shared memory systems have to be taken into consideration. These include access control, synchronization, protection, and security. Access control determines which process accesses are possible to which resources. Synchronization constraints limit the time of accesses from sharing processes to shared resources. Protection is a system feature that prevents processes from making arbitrary access to resources belonging to other processes (El-Rewini and Abd-El-Barr, 2005).

In distributed-memory systems (also referred to as message passing), each processor has a local, private memory. Distributed-memory systems are a class of multiprocessors in which each processor has access to its own local memory. Unlike shared memory systems, communications in these systems are performed via send and receive operations. A node in such a system consists of a processor and its local memory. Nodes are typically able to store messages in buffers (temporary memory

locations where messages wait until they can be sent or received), and perform send/receive operations at the same time as processing. Simultaneous message processing and problem calculating are handled by the underlying operating system. Processors do not share a global memory and each processor has access to its own address space. The processing units of a distributed-memory system may be connected in a variety of ways ranging from architecture-specific interconnection structures to geographically dispersed networks. Distributed-memory multiprocessors employ a variety of static networks in local communication. Of importance are Hypercube networks (Bhuyan and Agrawal, 1984), which have received special attention for many years. It was also apparent that distributed memory is the only way efficiently to increase the number of processors managed by a parallel and distributed system. If scalability to larger and larger systems (as measured by the number of processors) was to continue, systems had to use distributed memory techniques (El-Rewini and Abd-El-Barr, 2005).

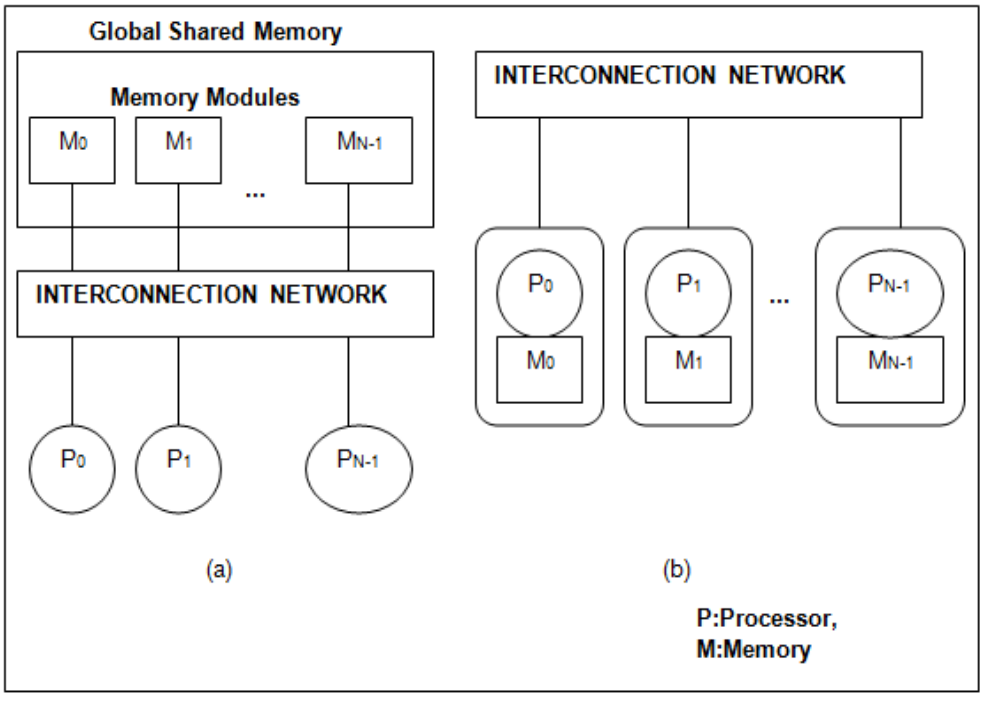

Figure 1.5. Shared-memory vs. distributed-memory multiprocessor (El-Rewini and Abd-El-Barr, 2005)

Both shared-memory and distributed-memory systems are commercially successful and both the interconnection network and the programming model play a major role in determining the communication speed (El-Rewini and Abd-El-Barr, 2005).

1.2. Message Passing Programming Model

In all parallel processing, data must be exchanged between cooperating tasks (Morrison, 2003). Message passing systems provide alternative methods for communication and movement of data among multiprocessors (El-Rewini and Abd-El-Barr, 2005). Current parallel programming paradigms for HPC systems are mainly relying on message passing (Fagg et al., 2004). Message passing programming model and distributed memory systems are common platforms for running large parallel scientific and engineering applications (Gursoy and Kale, 2004).

The message passing model has become the paradigm of choice, from the perspective of the number and variety of multiprocessors that support it, as well as in terms of applications, languages, and software systems that use it (Morrison, 2003). The message-passing paradigm provides a mean to write highly scalable algorithms, abstracting and hiding many architectural decisions from the application developers (Fagg et al., 2004).

A message passing programming model uses a set of primitives that allows processes to communicate with each other. These include the send, receive, broadcast and barrier primitives. The send primitive takes a memory buffer and sends it to a destination node. The receive primitive accepts a message from a source node and stores it in a specified memory buffer. The basic programming model used in message passing architectures is based on the idea of matching a send request on one processor with a receive request on another. In such scheme, send and receive are blocking; that is, send blocks until the corresponding receive is executed before data can be transferred (Grama et al., 2003).

Message passing communication protocol supports end-to-end packet acknowledgment. For every packet sent by a source node, there is a returned acknowledgment after the packet has reached the destination node. This allows source nodes to discover packet loss. Automatic retransmission of a packet is made if the acknowledgment is not received within a preset time interval. A message passing programming style is the preferred style for performance on such model. Also message passing without acknowledgement protocol can be defined as neglecting the fact that the source is not in need to learn whatever the sent packet has arrived or not. The main drawback of message passing is the programmer's responsibility for determining and orchestrating all parallelism (Grama et al., 2003).

Figure 1.6 shows an example message passing system consisting of four processes. In this figure, a horizontal line represents the execution of each process and lines extended among processes represent messages exchanged among these processes. A message is defined as a logical unit for inter-node communication; it is considered as a collection of related information that travels together as an entity. A message can be an instruction, data, synchronization, or interrupt signals. A message passing system interacts with the outside world by receiving input message(s) and/or outputting message(s). It is essential that the outside world perceives a consistent behavior of a given message passing system (El-Rewini and Abd-El-Barr, 2005).

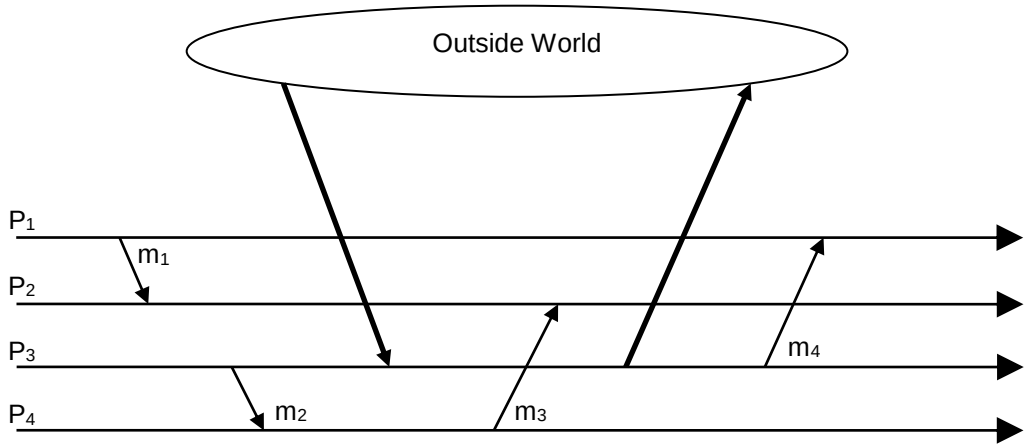

Figure 1.6. An example of a message passing system (El-Rewini and Abd-El-Barr, 2005)

Parallel scientific applications have been traditionally parallelized using a combination of a traditional sequential programming language such as Fortran of C/C++ with message passing facilities, such as Message Passing Interface (Gropp et al., 1998) allowing virtually complete control over the structure and degree of parallelism as well as the distribution and alignment of data across the architecture (Chan et al., 2003). Message Passing Interface provide the programmer with communication primitives to establish the necessary communications among tasks belonging to a whole parallel application (Roig et al., 2007). The Message Passing Interface standard is an application programming interface, not an implementation and it defines the user interface and functionality for a wide range of message passing capabilities (Gropp et al., 1998). In defining the syntax and semantics of the interface, the designers had to balance the impact of the specification on portability, efficiency, and ease of use (Duato et al., 1997). The main disadvantage of Message Passing Interface; low-level, cumbersome and error-prone communication management using the pair wise primitives send and receive has been known and criticized for years (Gorlatch, 2002).

1.3. Review of Congestion Control

An efficient interconnection network for HPC systems must provide low latency memory access and message transmission (Chan et al., 2003). While some communication latency can be tolerated by overlapping computation with communication, latency imposes fundamental limits on the effectiveness of multiprocessors (Miguel-Alonso et al., 2008). The latency depends not only on the programming paradigm of the network, but also congestion on the interconnection network, which is a significant problem due to the growth of networks and increased link speeds. Both electrical and optical interconnects may suffer from severe congestion problems with high traffic loads, which prevent reaching the wished performance (Baydal et al., 2005). If traffic increases and reaches (or surpasses) a certain level (the saturation point), accepted traffic falls down and message latency considerably increases. To keep the network below saturation and avoid the resulting

performance degradation, it is necessary to implement a congestion control mechanism (Gu et al., 2007).

There are two strategies to solve the congestion problem: congestion recovery and prevention. Congestion recovery techniques (Gu et al., 2007; Baydal et al., 2000; Dally and Aoki, 1993; Lopez et al., 1998; Lopez et al., 1997; Smai and Thorelli, 1998) are based on monitoring the network and triggering some actions when they detect congestion. In this case, the solution has three steps. First, congestion has to be detected. Second, congestion has to be notified to the network nodes, and third, some actions have to be applied to solve network congestion.

Congestion prevention techniques require some kind of authorization from the network before injecting a packet. Avoiding the congestion before it occurs is a superiority that congestion prevention methods have and these techniques are more preferred by the researchers for that reason (Baydal et al., 2005). Local and global congestion control mechanisms are proposed previously. It is understood that notifying only the local node has the disadvantage that applied actions to avoid network congestion may punish only the nodes that detect the congestion independently if they are mainly responsible for the problem. Thus, global congestion prevention mechanisms have become more popular. Thottethodi et al. (2004) use the number of busy output virtual channels in a node where as Baydal et al. (2000) propose an approach that counts a subset (free and useful) of virtual channel buffers to estimate congestion.

The actions that are triggered to avoid network congestion can be divided into three groups. First, and more studied, is message throttling. The second group uses non-minimal routing to balance network load along alternative paths (Dally and Aoki, 1993). Finally, the third group (Smai and Thorelli, 1998) adds additional buffers (congestion buffers) to slow down messages and reduce network traffic. Message throttling has been the most frequently used method. If the network is becoming congested, it seems reasonable that nodes stop, or at least slow down their injection rate of messages.

Some proposals completely stop message injection when congestion is detected (Jain, 1998; Petrini and Vanneschi, 1996; Thottethodi et al., 2001). Another

possibility is to reduce the available bandwidth to inject new messages. If several injection channels are available, a way may consist of progressively disabling them. Also, message transmissions may be delayed during increasing intervals (Kim et al., 1994). Message throttling can be applied for a predefined time (Smai and Thorelli, 1998; Kim et al., 1994), or until the traffic falls enough (Lopez et al., 1998; Baydal et al., 2000; Thottethodi et al., 2001).

The 2-D interconnection networks are common topologies due to its feasibility and low dimensionality. The 2-D interconnection networks have been mostly used as the nearest neighbor 2-D Mesh (Bani-Mohammad et al., 2011; Goh and Veeravalli, 2008), Torus (Darwish et al., 2008; Ju and Yang, 2012) and Crossbar (Chang, 2004; Kornaros and Orphanoudakis, 2010). Although advantages and common usage of 2-D interconnection networks, network congestion affects the system performance considerably. Several mechanisms have been presented to deal with the network congestion problem in 2-D interconnection networks:

In (Daneshtalab et al., 2012a), in order to reduce the network congestion, a streamlined method named Global Load Balancing is presented. The ideas behind the Global Load Balancing method are twofold: The first idea is to use the global congestion information as a metric for arbitration in routers to reduce the congestion level of highly congested areas. The second idea is to use an adaptive scheduler in network interfaces based on the global congestion information to avoid additional traffic to congested areas.

Li et al. (2006) proposed an adaptive deadlock free routing algorithm called Dynamic XY for 2-D Mesh interconnection network. In this algorithm, which is based on the static XY algorithm, a packet is sent either to the X or Y direction depending on the congestion condition. It uses local information which is the current queue length of the corresponding input port in the neighboring routers to decide on the next hop. It is assumed that the collection of these local decisions should lead to a near-optimal path from the source to the destination. The main weakness of Dynamic XY is that the use of the local information in making routing decision could forward the packet in a path which has congestion in the routers farther than the current

neighbors. This situation could happen when the routing unit is one unit apart from the destination in X or Y dimension.

Lotfi-Kamran et al. (2010) solved the problem of the Dynamic XY routing algorithm with little area overhead. It is congestion-aware and more link failure tolerant compared to the Dynamic XY algorithm which is called Enhanced Dynamic XY. On contrary to the Dynamic XY algorithm, it can avoid the congestion when routing from the current switch to the destination whose X position (Y position) is exactly one unit apart from the switch X position (Y position). This is achieved by adding two congestion wires (one in each direction) between each two cores which indicate the existence of congestion in a row (column). The same wires may be used to alarm a link failure in a row (column). These signals enable the routing algorithm to avoid these paths when there are other paths between the source and destination pair. In addition, the proposed technique increases tolerance against single link failure compared to the Dynamic XY technique.

In (Daneshtalab et al., 2010) a router architecture, which utilizes both input and output selections, is proposed to avoid congested areas in regular 2-D on-chip networks. The output selection of the presented router utilizes an adaptive routing algorithm supporting both unicast and multicast traffic while the input selection part of the router uses the weighted round robin arbitration. The input selection, profits from the advantages of both priority (unfair) and non-priority (fair) arbitration policies. The adaptive output selection algorithm uses congestion flags to route packets through non-congested paths and consequently helps balance the traffic, whereas the weighted round robin input selection assist in relieving nodes where congestion is formed. Under the multicast, unicast, and mixed traffic models and in high flit injection rates, the proposed model has the lowest average communication delay.

Trumler et al. (2008) present an approach to increase the network throughput and to lower the average latency based on the local load information of the nodes. They used a simple 2-D Mesh for communication infrastructure. The nodes exchange their local load values with their neighboring nodes, which route incoming packets based on this information. The basic idea of this self-organizing, adaptive routing

algorithm is inspired by the self-optimization algorithm for load balancing in large scale networks, which is based on the notion of the human hormone system. The throughput gain is up to 127% compared to the original algorithm. The throughput gain is highest for the uniform random traffic pattern.

Congestion control parameters which are the main part of the congestion control mechanisms are as important as the techniques for solving the congestion problem. Input, output and hybrid parameters have been proposed in previous studies. Occupied input (Daneshtalab et al., 2012a; Wang et al., 2012) and output (Daneshtalab et al., 2012b) channel count, input flit count (Lotfi-Kamran et al., 2010), buffer utilization (Trumler et al., 2008) and link utilization (Van den Brand et al., 2007) are used as parameters of congestion control mechanisms.

Previous congestion control mechanisms can not solve the congestion in multiple input queues of broadcast-based optical interconnection networks (Hemenway et al., 2004; Katsinis and Hecht, 2004). (Aci and Akay, 2010) showed that the congestion problem is possible for input queues of broadcast-based optical interconnection networks and proposed a congestion control algorithm to prevent congestion in the input queues of a broadcast-based optical multiprocessor architecture. The performance of the algorithm is evaluated by Optimized Network Engineering Tool (OPNET) Modeler (Chang, 1999) with various synthetic traffic patterns on a 64-node 1-D Simultaneous Optical Multiprocessor Exchange Bus (SOME-Bus) architecture employing the message passing (Gropp et al., 1998) protocol. Only input parameters have been implemented and the proposed congestion control algorithm is only simulated on 1-D SOME-Bus architecture. Performance measures such as average input waiting time, average network response time and average processor utilization have been collected before and after applying the algorithm. The results show that the proposed algorithm is able to decrease the average input waiting time by 13.99% to 20.39%, average network response time by 8.76% to 20.36% and increase average processor utilization by 1.92% to 6.63%.

1.4. Contributions of the Thesis

In this work, we propose a 2-D congestion control algorithm to prevent congestion in broadcast-based optical multiprocessor architecture employing the message passing protocol. The 1-D congestion control algorithm (Aci and Akay, 2010) is used to design the 2-D congestion control algorithm. In order to take the advantage of 2-D, architectural and communicational features (i.e. *bypass operation* and *intermediate node selection procedure*) are considered in the algorithm. Input, output, and hybrid parameters are utilized for performance improvement. The 2-D SOME-Bus congestion control algorithm is tested via simulation under Uniform (UN), Hot Region (HR) and Bit Reverse (BR) synthetic traffic patterns using Asynchronous and Client-Server traffic models. To get a better understanding of the algorithm, four evaluation cases such as 4 threads and bypass operation, 8 threads and bypass operation, 4 threads and no-bypass operation and 8 threads and no-bypass operation are yielded. Performance measures such as average processor utilization, average channel waiting time, average input waiting time and average network response time have been collected before and after applying the algorithm.

For Client-Server traffic model, the proposed algorithm is able to increase the average processor utilization 1.59% to 12.57% with 4 threads and 1.21% to 13.91% for 8 threads, decrease the average channel waiting time 3.47% to 16.17% with 4 threads and 6.18% to 25.02% for 8 threads, the average input waiting time 2.41% to 19.29% with 4 threads and 4.38% to 27.85% for 8 threads, the average network response time 2.85% to 21.07% with 4 threads and 4.99% to 30.27% for 8 threads. For Asynchronous Message Passing traffic model, the proposed algorithm is able to increase the average processor utilization 1.86% to 13.75% with 4 threads and 1.04% to 13.91% for 8 threads, decrease the average channel waiting time 4.26% to 18.07% with 4 threads and 10.55% to 29.53% for 8 threads, the average input waiting time 4.63% to 22.06% with 4 threads and 5.67% to 30.22% for 8 threads, the average network response time 17.53% to 44.71% with 4 threads and 31.16% to 52.09% for 8 threads.

1.5. Organization of the Thesis

The rest of the thesis is organized as follows: Section 2 presents an overview of the SOME-Bus architecture. Section 3 presents details of the congestion control algorithm. Simulation framework and traffic models are described in Section 4. Results and a detailed discussion of the findings are presented in Section 5. Finally, Section 6 concludes the thesis. Appendices show graphical representation of the results.

2. OVERVIEW OF THE SOME-BUS ARCHITECTURE

2.1. Optical Interconnection Networks

As both silicon and gallium arsenide based technologies drive device speed into the gigahertz frequency range, and rise times down to small fractions of nanoseconds, communications, not device speed, between subsystems and chips at higher and higher data rates are the limiting and decisive factor for performance and cost of high-speed computing systems. These limitations are manifested in both high-speed uni-processors, as well as HPC systems. For uni-processors, higher performance is obtained from faster clock speeds. However, for systems operating in the gigahertz range, electrical connections have to be treated as high frequency transmission systems. Skin effect, crosstalk, interference, and dielectric imperfections cause severe pulse distortions and attenuation, clock skew and random propagation delays. For multiprocessor design, the technological limitations of electronic interconnections are resulting in very limited communication bandwidth, low interconnect density, high power requirements, and long network latencies (Cohen et al., 2000).

Optical technology has long been considered appealing for constructing high-speed interconnects in multiprocessor systems (Tan et al., 2008). Optical interconnection networks offer a potentially disruptive technology solution that can provide ultrahigh throughput, minimal access latencies, and low-power dissipation that remains independent of capacity. By realizing the enormous bandwidth capacities and stringent latency requirements of interconnecting the growing number of compute elements in a scalable fashion, optical interconnection networks can become key to relieving what is perhaps the most daunting challenge to performance of future computing systems (Hawkins et al., 2007).

2.2. The SOME-Bus Architecture

SOME-Bus is a processor interconnection scheme that uses the properties of optics to provide the benefits of small interconnection distances and high data rates (Katsinis, 2001). It is a proposed optical interconnection architecture for over a hundred processors which contains a dedicated transmission channel for each processor to eliminate global arbitration and to provide bandwidth that scales with the number of processors in the machine. Unlike electrical buses in which the limits are due to the electrical characteristics of the wire, the bandwidth of optical interconnects is not limited by the fiber optics used to connect the transmitters and receivers; the bandwidth limitations are due to the transmitter and receivers (Cohen et al., 2000).

SOME-Bus is low-latency, high-bandwidth, fiber-optic network which directly connects each processing node to all other nodes without contention. One of its key features is that each of P nodes has a dedicated broadcast channel which can operate at several Gbytes/s, depending on the configuration. In general, the SOME-Bus contains K fibers, each carrying M wavelengths organized in M/W channels, where each channel is composed of W wavelengths. The total number of fibers is K = PW/M. A simple configuration with 128 nodes (P = 128 channels) and W = 1 wavelength per channel would require K = 32 fibers with M = 4 wavelengths per fiber, and a receiver array at each node containing 128 detectors organized as 32 × 4 over the surface of a single chip. Each of P nodes also has an input channel interface based on an array of P receivers (each with W detectors) which simultaneously monitors all P channels (Katsinis, 2004).

The physical implementation of SOME-Bus is motivated by recent progress in optical communication, dense wavelength-division-multiplexing, and optoelectronics. Slant Bragg gratings (Bouzid and Abushagur, 1996) are written directly into the fiber core and are used as narrow-band, inexpensive output couplers. This coupling of the evanescent field allows the traffic to continue and eliminates the need for regeneration. Figure 2.1 shows the parallel receiver array and output coupler. The SOME-Bus also uses amorphous silicon (a-Si) photodetectors built as

superstructures on the surface of electronic processing devices. Due to the low conductivity of the a-Si layer, no subsequent patterning is required, and therefore the yield and cost of the receiver is determined by the yield and cost of the CMOS device itself (Katsinis, 2004).

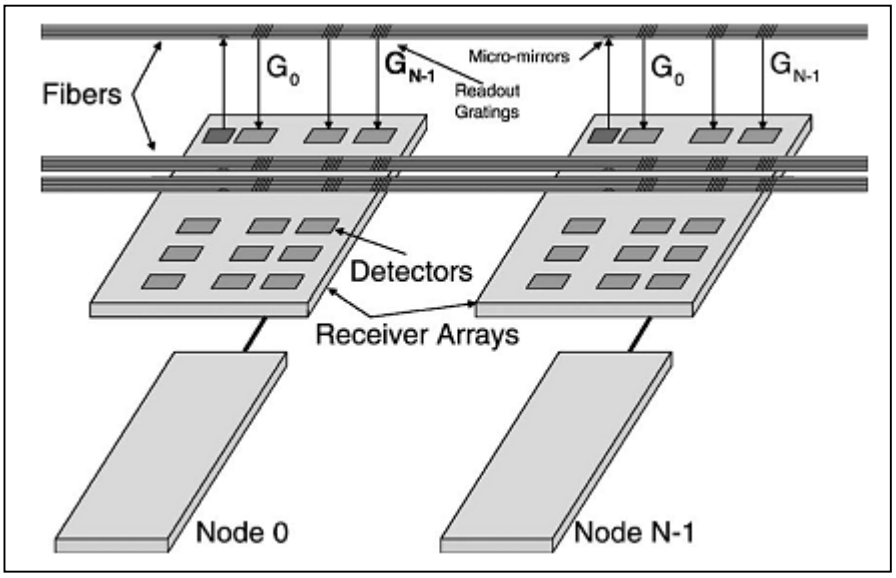

Figure 2.1. Parallel receiver array (Katsinis, 2004)

The receiver array does not perform any routing and consequently its hardware complexity is small. It contains an optical interface which performs address filtering, barrier processing, length monitoring and type decoding. The address filter can recognize multicast group addresses as well as broadcast addresses in addition to recognizing the address of the host node. The receiver array also contains a set of queues such that one queue is associated with each input channel, allowing messages from any number of nodes to arrive and be buffered simultaneously. This organization supports multiple simultaneous broadcasts, provides bandwidth that scales directly with the number of nodes in the system and eliminates the need for global arbitration. Arbitration may be required only locally in the receiver array when multiple input queues contain messages (Hecht and Katsinis, 2003).

Once the logic level signal is restored from the optical data, it is directed to the input channel interface which consists of two parts: the optical interface and the processor interface. Figure 2.2 shows the optical interface which includes physical

signaling, address filtering, barrier processing, length monitoring and type decoding (Zhu et al., 2004). Each receiver generates a data stream which is examined to detect the start of the packet and the packet header. The header decode circuitry examines the header field, which includes information on the message type, destination address (or addresses) and length, to determine whether or not the message is a synchronization message. If the message is a synchronization message, it is handled by the barrier circuitry, otherwise the destination address is compared to the set of valid addresses contained in the address decode circuitry. In addition to recognizing the local node address, the address filter can recognize multicast group addresses as well as broadcast addresses. Once a valid address has been identified, the message is placed in a queue. If the address does not match, the message is ignored (Katsinis, 2004).

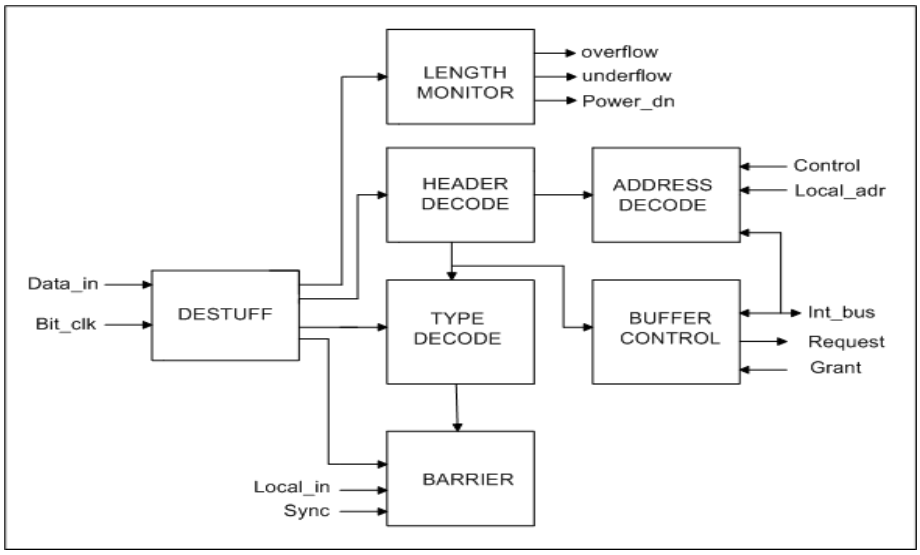

Figure 2.2. SOME-Bus optical interface (Zhu et. al., 2004)

Figure 2.3 shows the processor interface which includes a routing network (resolver circuit) and a queuing system. One queue is associated with each input channel, allowing messages from any number of processors to arrive and be buffered simultaneously, until the local processor is ready to remove them. The resolver circuit receives a request signal (R_i) from each non-empty queue and produces the index of the next queue to be accessed under either the limited or the exhaustive

service disciplines. The local processor can force the next queue selection through the P_{in} input. A straightforward implementation of the resolver as a selection tree, using logic gates to select the next queue and multiplexers to forward the corresponding queue index, requires only several hundred gates organized in $\log_2(P)$ levels. The time required to select the next queue (polling walk time) is consequently very small and can be overlapped with the queue access time (Katsinis, 2004).

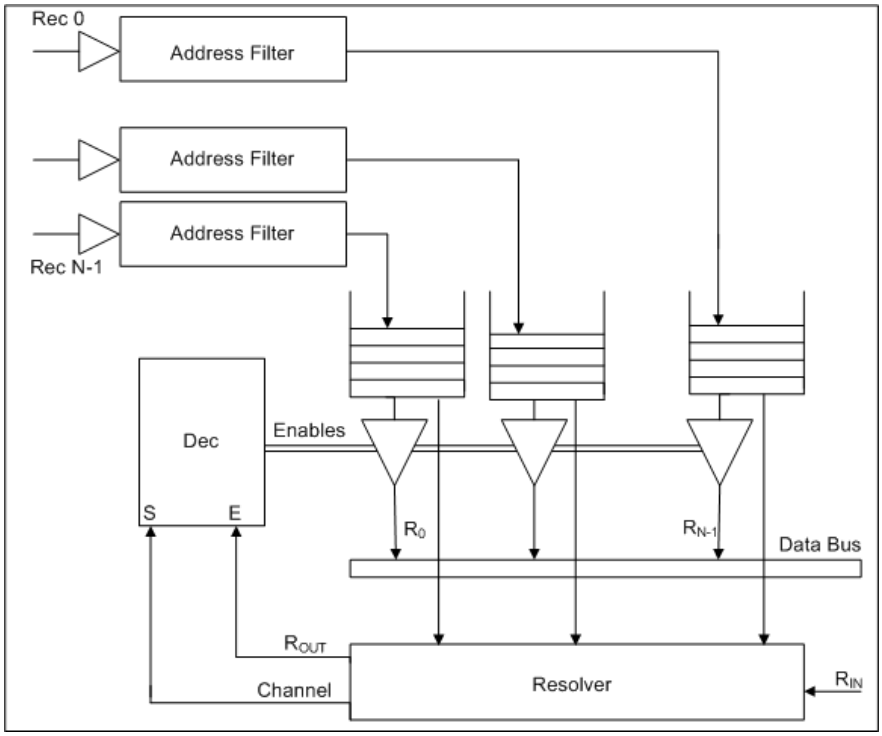

Figure 2.3. SOME-Bus processor interface (Katsinis, 2004)

The SOME-Bus has much more functionality than plain crossbar architecture. With N nodes, the diameter of the SOME-Bus is 1, the time needed for all-to-all communication with distinct messages is O(N) and the time needed for synchronization is O(1). Unlike a fully- connected point-to-point network, where the number of transmitters and channels increases $O(N^2)$, the number of transmitters and channels of the SOME-Bus is O(N), quite smaller than the number required in other popular architectures, such as the Hypercube or the Torus. The total number of receivers is N^2, which is larger than the number required in other architectures. They are arranged so that N receivers are fabricated as amorphous silicon structures

constructed as a thin film directly on the surface of a digital CMOS device, with no lithography required. Because of the low conductivity of the amorphous silicon layer, no subsequent patterning is required, and therefore the yield and cost of the receiver is determined by the yield and cost of the CMOS device itself. The full receiver array can be implemented on a single chip even for large values of N (N>128). Therefore, the total receiver cost is approximately O(N) instead of $O(N^2)$. The SOME-Bus with N nodes can be scaled to 2N nodes by using four SOME-Bus segments to create twice the number of channels where each channel is twice as long to accommodate the additional nodes (Hecht and Katsinis, 2003).

2.3. The 2-D SOME-Bus Architecture

The 2-D SOME-Bus consists of N horizontal and N vertical 1-D SOME-Bus networks and N^2 nodes. Each of N nodes connected to one 1-D SOME-Bus has a dedicated broadcast channel and an input channel interface based on an array of N receivers monitoring all N channels and allowing multiple simultaneous broadcasts. At each node, an electro-optical converter consisting of a dual receiver and transmitter pair allows messages broadcast on one bus to be forwarded and broadcast (in a cut-through manner) on the bus in the other dimension (Katsinis and Nabet, 2004).

If a node N_{ij} (connected to vertical bus V_i and horizontal bus H_j) sends a message to node N_{mn}, and either $i = m$ or $j = n$, then only one bus (vertical or horizontal, respectively) is used. The message header includes the identification of the destination node and the message is broadcast on the proper bus. If both $i = m$ and $j = n$, then the message is broadcast first on horizontal bus H_j to node N_{mj} with the header containing an indication that node N_{mj} broadcast the message on vertical bus V_m so it can be delivered to its final destination, node N_{mn}. By symmetry, the source node may choose to broadcast the message first on vertical bus V_i to intermediate node N_{in} for rebroadcast on horizontal bus H_n (Katsinis and Nabet, 2004).

The design of the header allows for the specification of multiple destinations so that the full capabilities of the architecture may be exploited. In the simplest configuration, the message header contains a list of final destinations. The message is broadcast first on the horizontal bus at the source node and is delivered to all required nodes on that horizontal bus. Each of these nodes examines the header to determine if it is specified as a destination of the message and/or if it must rebroadcast the message on its own vertical bus. If no rebroadcast from one dimension to the other is necessary, then the system operation is equivalent to the operation of the 1-D SOME-Bus (Katsinis and Nabet, 2004).

When a message must be rebroadcast from one dimension to the other, the architecture offers two ways to accomplish this operation. At the simplest, a message is enqueued locally at the intermediate node. At some later time, the message is broadcast on the other dimension in the same way as locally generated messages are transmitted. The disadvantage of this method is that latency is increased considerably due to queuing at the intermediate node and retransmission. There is large probability that a message that arrives at an intermediate node from one dimension finds the transmitter on the other dimension idle and therefore, cut-through transmission of the message is beneficial. As shown in Figure 2.4, each node has the electro-optic circuitry that allows the node to determine if an incoming message must be bypassed to the other bus and retransmits it, blocking the local transmitter for the duration of the transfer of the incoming message. Further details about the 2-D SOME-Bus architecture can be found in (Katsinis and Nabet, 2004).

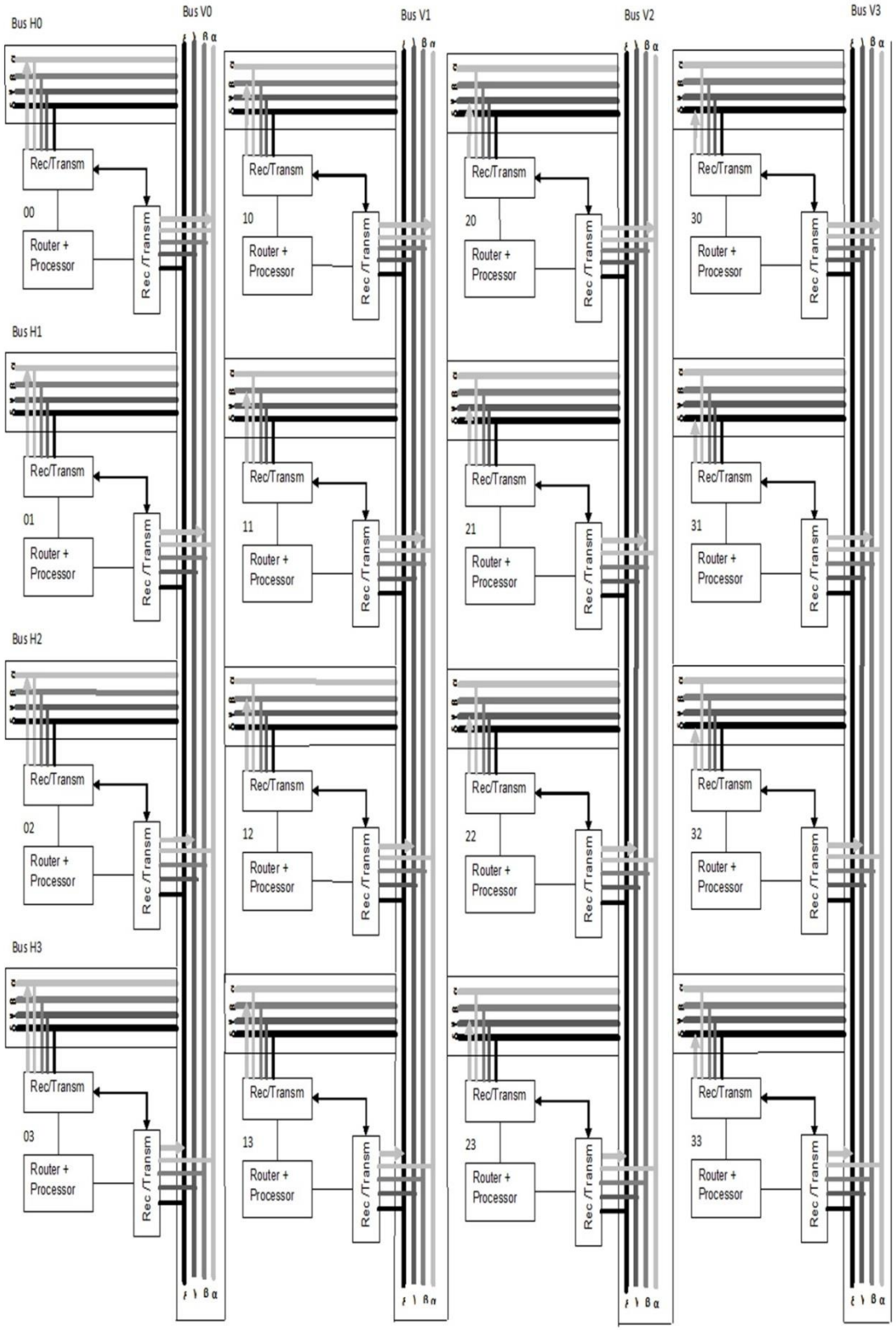

Figure 2.4. 2-D SOME-Bus

3. CONGESTION CONTROL ALGORITHM

3.1. Motivation

Aci and Akay (2010) presented an implementation of 1-D congestion control algorithm. It was shown that congestion does not only occur in the output buffers (output buffer is a node's output channel queue which is used to store packets when the channel is busy), also input buffers (input buffer is a node's processing input queue which is fed from the interconnection network) have great contribution to network latency.

In line with the results of 1-D congestion control algorithm, we develop a congestion control algorithm using architectural features of 2-D SOME-Bus network. 1-D and 2-D congestion control algorithms have some differences:

In terms of communication: The 2-D SOME-Bus has intermediate nodes *(inter_node)* (i.e. if destination node is not vertical or horizontal neighbor of source node, the message is routed to intermediate node which is either horizontal neighbor of source node and vertical neighbor of destination node or vertical neighbor of source node and horizontal neighbor of destination node) and *bypass operation* (i.e. if intermediate node's output buffer is not busy, incoming message must be bypassed to the destination node without waiting in the output buffers) which 1-D SOME-Bus does not have. To take advantage of *inter_node* and *bypass operation, intermediate node selection procedure* (i.e. choosing the less busy *inter_node* for transmitting the message) is added to the congestion control algorithm as well.

In terms of architecture: 1-D SOME-Bus with N nodes has one output queue which transfers messages through the associated channel and N input queues which store messages from the channels in each node. On the other hand, 2-D SOME-Bus with N^2 nodes, contains N 1-D SOME-Bus networks and there are N additional 1-D SOME-Bus networks interconnecting the nodes.

3.2. The Algorithm

The flow-chart representation of the congestion control algorithm is given in Figure 3.1. The algorithm effects both receive and send operations of processors. Therefore, an initialize operation is executed at the beginning of simulation. In this operation, type of the algorithm parameters (i.e. input, output or hybrid) is decided, parameter thresholds are defined, input queue pointers (it is used for determining which packet is going to be processed by the processor) for horizontal and vertical queues are set to their default values and local status tables (it indicates how many packets there are in each input queue of the neighbor nodes) are updated. Initially, the status table of each node is empty.

In the receive operation (*recv_op*), the packet is received by the receiver. Then, the destination address (*dest_addr*) of the packet pointed by the input queue pointer is extracted. If *dest_addr* is equal to the node address (*node_addr*), it is sent by whether a source node which is vertical or horizontal neighbor of the node or an intermediate node which has a mission to route the packet. The sent packet is inserted into suitable queue and waits for service. If *dest_addr* is not equal to the *node_addr*, the node becomes *inter_node*. If the output buffer is empty, the packet is routed to its destination. If not, the packet is inserted into suitable queue and waits for service.

In the send operation (*send_op*), firstly vertical or horizontal queue is chosen in order. The turn is changed in every *send_op*. According to the queue index that is supported by input queue pointer, the packet is selected to process. After that, *dest_addr* is decided and using local tables, parameter values are calculated. Parameter values of *dest_addr* are only considered if it is vertical or horizontal neighbor of the node. Otherwise, *inter_node* is specified and parameter values are calculated according to this temporary destination. If the parameter values surpass a threshold value, the processor does not send the packet; skips it and proceed to the next packet unless packet pass field (*pk_pass_fld*) is above a certain threshold. If not, the packet is removed from input queue and processed by the processor. If *dest_addr* is vertical or horizontal neighbor of the node, the packet is sent to the *dest_addr*

directly. If not, it is sent to the *inter_node* in order to route. The *inter_node* selection is a critical issue in the congestion control algorithm. Usually, the horizontal neighbors of the source node are chosen as *inter_node.* However, choosing vertical or horizontal neighbors constantly brings additional congestion problems for the input queues. In this algorithm, we have developed *intermediate node selection procedure* which is used for selecting either vertical or horizontal intermediate node for transmitting the packet. The procedure is operated in the same principle as the congestion control algorithm and details are described below.

The input parameters are specified as destination pointer position (*dest_ptr_pos*) (i.e. which packet is going to be processed by the processor in destination node), destination input queue number (*dest_input_q_no*) (i.e. the queue index which packet is going to place in the destination node), destination count (*dest_cnt*) (i.e. how far the destination processor can be from processing the packet assuming it was placed in the destination input queue. It is calculated by *dest_cnt* = *dest_input_q_no* − *dest_ptr_pos*, destination input queue packet count (*dest_input_q_pk_cnt*) (i.e. number of packets in destination input queue). The output parameter which is the most popular parameter in previous studies is chosen as source output queue packet count (*src_output_q_pk_cnt*) (i.e. number of packets in source output queue). The hybrid parameter is consisted by *dest_cnt* and *src_output_q_pk_cnt* parameters. The independent parameter, *pk_pass_fld* (i.e. how many times a packet can be skipped) is used for avoiding infinite waiting problems in the queues.

The algorithm is based on the fact that injection of a new packet will be allowed only if the destination node is get ready to process it. Moreover, if the channels that may forward the packet toward its destination are congested, input waiting time will increase dramatically as well as network response time. Therefore, thresholds are very important to keep the network below saturation and avoid the resulting performance degradation. The input and output queues in this study are considered congested when its occupancy goes above a settable threshold level (the thresholds are defined empirically). This is done by comparing the predefined threshold value to the number of packets in the queues. In addition, *dest_cnt* and

pk_pass_fld are compared with their thresholds. In order to use this mechanism, a complete analysis of its behavior for several network loads is done, leading to find optimal thresholds.

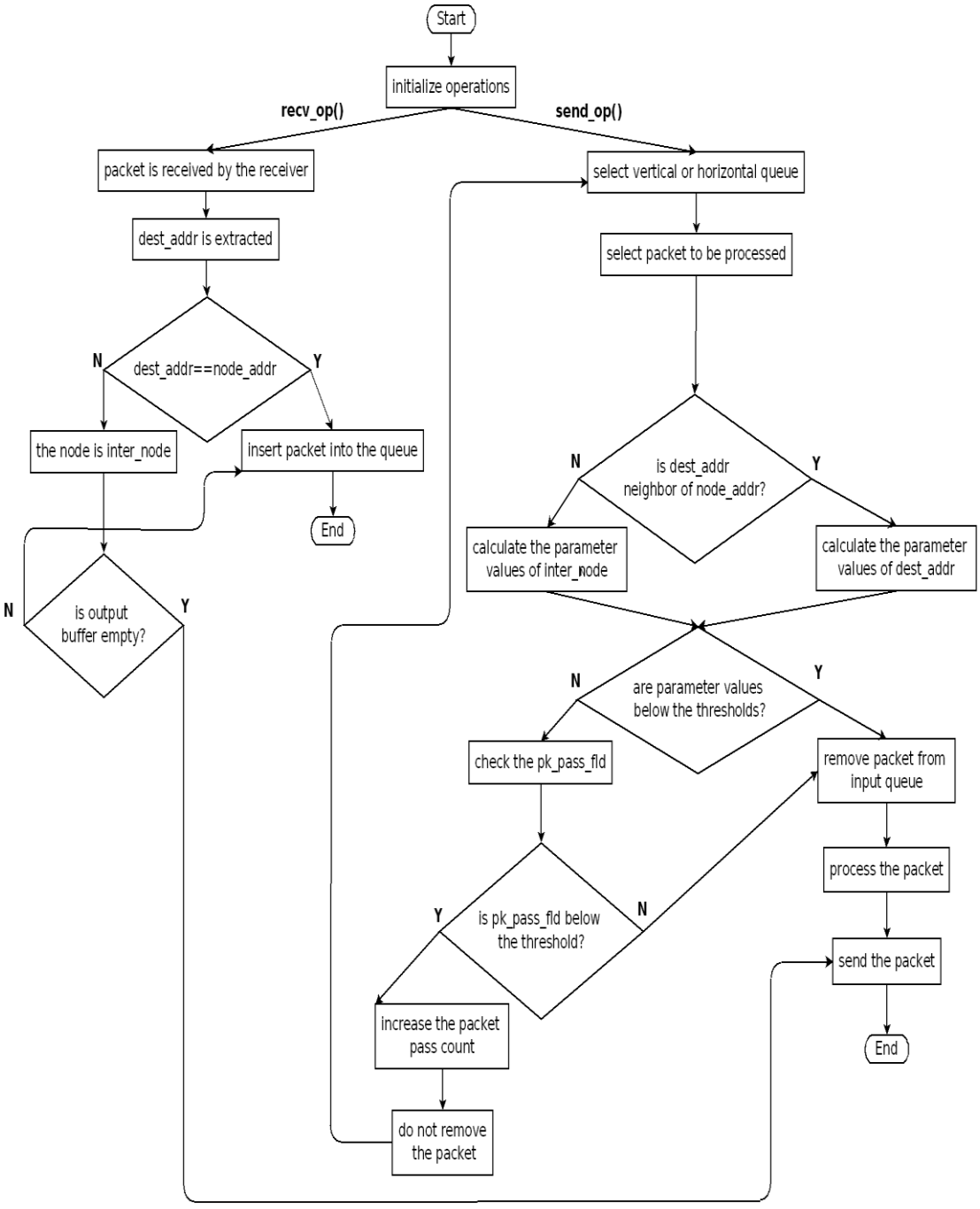

Figure 3.1. Flow-chart representation of congestion control algorithm

4. SIMULATION

In this section, we present simulation details to evaluate the performance of the 2-D SOME-Bus congestion control algorithm and synthetic traffic generation parameters.

4.1. Simulation Framework

Since the 1950s, various computer simulation techniques have become increasingly important research tools across a wide range of sciences. Software packages based on these techniques are also widely used in more applied fields such as engineering, finance, or environmental management, often in connection with computerized databases and electronic gathering devices (Foss et al., 2003). There are several advantages of simulation compared with using real data (Morrison, 2003):

1. Computer simulations are far cheaper and faster than physical experiments.
2. Computers can solve a much wider range of problems than specific laboratory equipment can.
3. Computational approaches are only limited by computer speed and memory capacity, while physical experiments have many practical constraints.

Simulation modeling is becoming an increasingly popular method for network performance analysis. Generally, there are two forms of network simulation: analytical modeling and computer simulation. The first is by mathematical analysis that characterizes a network as a set of equations. The main disadvantage is its over simplistic view of the network and inability to simulate the dynamic nature of a network. Thus, the study of a complex system always requires a discrete event simulation package, which can compute the time that would be associated with real

events in a real-life situation. Software simulator is a valuable tool especially for today's network with complex architectures and topologies. Designers can test their new ideas and carry out performance related studies, therefore freed from the burden of the "trial and error" hardware implementations (Chang, 1999).

A typical network simulator can provide the programmer with the abstraction of multiple threads of control and inter-thread communication. Functions and protocols are described either by finite-state machine, native programming code, or a combination of the two (Chang, 1999). OPNET Modeler is an environment for network modeling and simulation, which can also be used for designing and studying interconnection networks and protocols. It is based on a series of hierarchical editors, project editor, node editor and process editor, which directly parallel the structure of interconnection networks and protocols. The node editor captures the architecture of a system by depicting the flow of messages between functional elements, called modules. Each module can generate, send and receive packets from other modules to perform its function within the node. Modules typically represent physical resources such as buffers, ports, queues and buses. Modules are assigned process models, developed in the process editor, to achieve any required behavior. The process editor uses a finite state machine approach to support specification of protocols, algorithms and queuing policies. States (the condition of a module) and transitions (a change of state) graphically define the progression of a process in response to events. States have "enter executives" (code that is executed when the module moves into a state) and "exit executives" (code that is executed when the module leaves a state), and there is "transition executive" (code that is executed in response to a specific event). There are two kinds of states: An unforced state is the one that returns control of the simulation to the simulation kernel after executing its enter executives. A forced state is one that does not return control, but instead immediately executes the exit executives and transitions to another state (Akay and Abasıkeleş, 2010).

In this study, communication between nodes is performed by broadcasting messages. The processor at each node extracts a data message from an input queue processes it for a period of time and when that period expires, it generates an output data message. A processor becomes idle only when all its input queues are empty.

The underlying process model that controls queue modules' behavior is OPNET's built-in "acb_fifo" model, which is given in Figure 4.1. The "init" state is used to initialize the process and setting the appropriate variables. If a packet arrives when the process is in "init" state, the process transitions to the "arrival" state, else it transitions to the "idle" state where it waits for packet arrival. The "arrival" state is used for receiving packets and starting service. In the "arrival" state, if the server is not busy then the process moves into the "svc_start" state, which in turn transitions to the "idle" state, where it waits either for packet arrival or service completion. While in the "idle" state, if the processing of a packet is completed, the process moves into the "svc_comp" state. While in the "svc_comp" state, if the queue is not empty, the process moves into the "svc_strt" state.

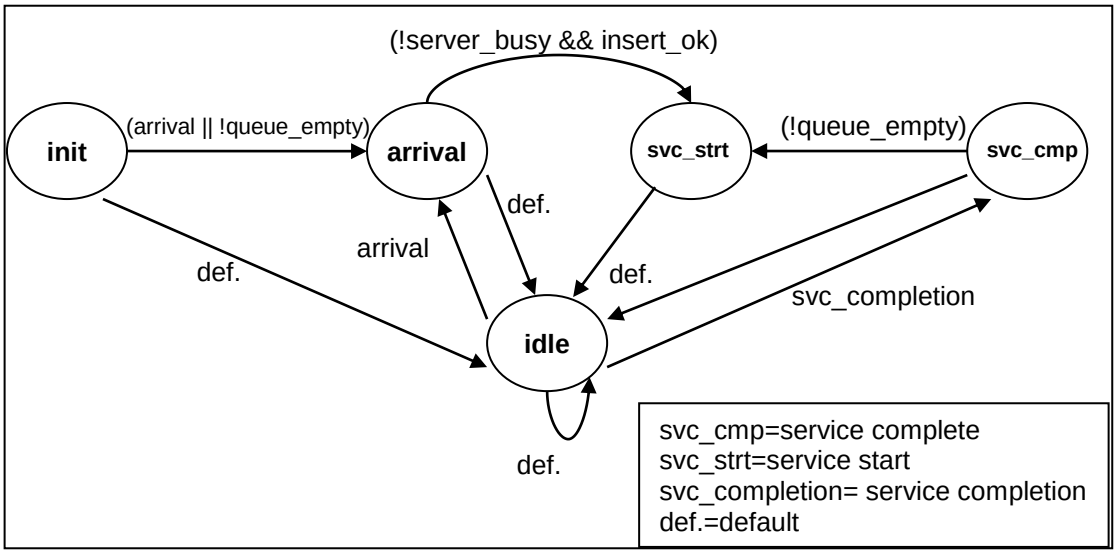

Figure 4.1. OPNET's acb_fifo model

4.2. Traffic Generation Parameters

Figure 4.2 shows the queuing network model of the 2-D SOME-Bus architecture used in the simulation. Each node has a processor service station and a channel service station. The processor service station contains one server and 2N input queues into which all channels may transfer messages. The input queues constitute of vertical and horizontal queues. For 64-node 2-D SOME-Bus has 16

input queues which 8 of it regarded as vertical, and 8 of it regarded as horizontal queue. The channel server station contains one server and a single queue which receives messages from the processor in the same node.

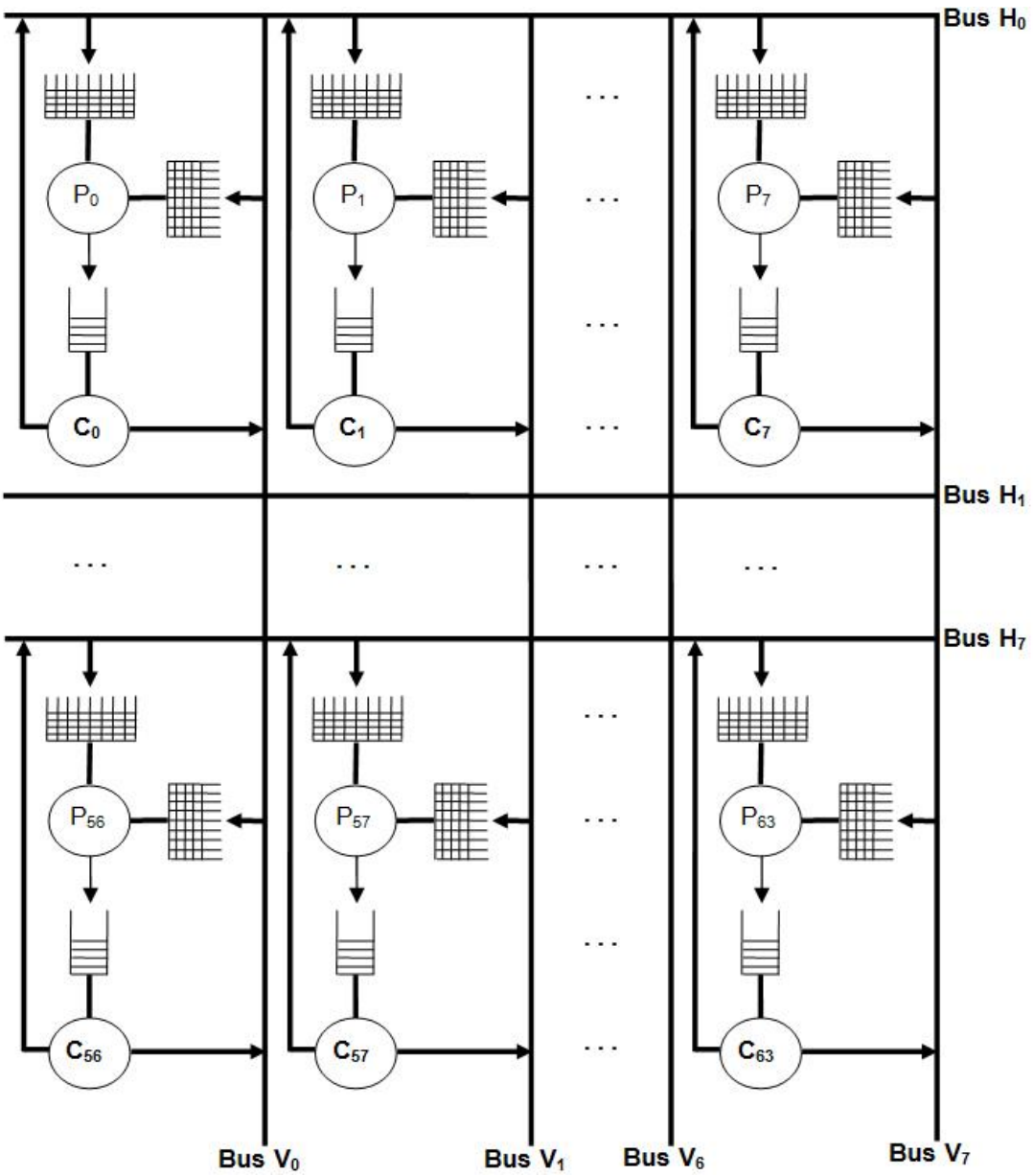

Figure 4.2. Queuing network representation of 2-D SOME-Bus (P=Processor, C=Channel, V=Vertical Bus, H=Horizontal Bus)

Using synthetic traffic workloads and running a simulator for a large number of cycles to get performance results with the network in steady state has been widely

used in past studies (Miguel-Alonso et al., 2008). Although not a completely realistic assumption, the results obtained with synthetic traffic are expected to indicate the minimum level of performance the network could provide under actual traffic. This has been shown to be true for some applications such as Radix or LU (Singh et al., 1992) which are part of the SPLASH benchmark suite. A synthetic traffic workload is defined by three important parameters: *spatial distribution* describes the destination node distribution for each source node, *temporal distribution* specifies packet generation times and *message length distribution* gives the size of each message. Regarding spatial distributions, we have used a collection of well-known permutation: BR. We have also included UN and HR traffic models.

UN traffic pattern can be represented by a traffic matrix, where each matrix element $\lambda_{s,d}$ gives the fraction of traffic sent from node s to the destination node d. In the UN traffic, the destination node is selected using uniform distribution with mean in range from 1 to N. Bit permutations such as BR are those in which each bit d_i of the b-bit destination address is a function of the one bit of the source address (Dally and Towles, 2004). In the HR pattern, the destinations of the 25% of the packets are chosen randomly within a small hot-region consisting of 12.5% of the nodes (Blumrich et al., 2003). Table 4.1 lists the destination node selection for these traffic patterns.

Table 4.1. Synthetic traffic patterns (Dally and Towles, 2004)

Name	**Pattern**
UN (Random)	$\lambda_{s,d} = 1/N$
BR	$d_i=b_{i+1}$
HR	The 25% of the packets are sent to 12.5% of the node group

Temporal distribution of packet generation can be implemented by independent or non-independent traffic sources (Miguel-Alonso et al., 2008). As its name implies, independent traffic sources such as Asynchronous traffic progress independently of the others and may use a Poisson distribution or on-off models. Most simulation-based studies of interconnection networks use independent traffic

sources (Shin, 2003). The main drawback of using just independent sources is that the obtained results may not be realistic representative of network performance under heavy loads (Izu et al., 2005). Also, independent sources can not capture reactive data exchange patterns, which are common in real applications. Non-independent traffic sources such as Client-Server traffic can simulate reactive data exchange patterns. In our simulations, we utilized both Asynchronous Message Passing (i.e. initially, all nodes generate traffic independently of the others, as time progresses traffic generation at the source / destination nodes depend on the receipt of messages from destination / source nodes) and Client-Server traffic (i.e. a server node sends packets to respond to the reception of packets from clients).

Regarding message length distributions different message sizes have been evaluated for Message-Passing applications in past studies (Abbas and Bayoumi, 2002; Kumar et al., 2008). The message transfer time (T) is assumed to be uniformly distributed with mean in range from 20 to 200 clock cycles. Since T is closely related to the packet length, using different values for T allows us to evaluate the performance of the congestion control algorithm for varying packet sizes. Specifically, let m be the miss rate and F the number of instructions per second performed by the processor at each node. Also, let S be the mean packet size in bytes and C the channel bandwidth (in bytes per second). Then, the ratio of the mean thread run time to the mean packet transfer time (T/R) is equal to mSF/C. The ratio T / R varies between 0.2 and 2. Congestion control algorithm parameters increase the message header by 18 bits and we took it into account after applying the congestion control algorithm in all simulations.

The parameters of the simulated SOME-Bus system are given in Table 4.2. In all simulations, simulation duration is 10 hour for each run and it is assumed that the processor at each node extracts a packet from an input queue, processes it for a period of time and when that period expires; it generates an output data message. A processor becomes idle only when all its input queues are empty. Each packet carries the *node_addr*, *dest_addr*, *inter_node*, type of message, position of the input queue pointer and packet array in its header portion. Packet array is a local status table in a node indicates how many packets there are in each input queue of the remote nodes.

The major parameters of the simulation are the number of nodes (N^2), the number of threads (K) in the system and the destination node selection distribution. Under steady-state conditions, K can be viewed as a measure of the system workload. Here, it is assumed to be constant (but dependent on system size N). The system workload is examined by selecting either four (K=256) or eight (K=512) threads per node with a fixed number of node (N^2=64) in the architecture.

Other parameters of the simulation are the threshold values for parameters. For K=256, threshold values of *dest_cnt* (selected as 3), *dest_input_q_pk_cnt* (selected as 5) and *src_output_q_pk_cnt* (selected as 3) are chosen. The threshold value for *pk_pass_fld* is selected as 3. For K=512, threshold values of *dest_cnt* (selected as 2), *dest_input_q_pk_cnt* (selected as 6) and *src_output_q_pk_cnt* (selected as 6) are chosen. The threshold value for *pk_pass_fld* is selected as 3 for all synthetic traffic patterns. Several runs of the simulation reveal that there is not a simple mathematical relation between these parameters. As a result of this, all of the thresholds are decided empirically.

Table 4.2. Simulation parameters

Parameter	**Value**
Simulation Duration	10 h
Number of threads in the system (K)	256 and 512
Number of nodes in the system (N^2)	64
Message Transfer Time (T)	Uniformly distributed with mean in range from 20 to 200 clock cycles
Message run time (R)	Exponentially distributed with mean of 100 clock cycles
Number of threads in each node	4 for K= 256, 8 for K= 512
dest_cnt	3 for K= 256, 2 for K= 512
dest_input_q_pk_cnt	5 for K= 256, 6 for K= 512
src_output_q_pk_cnt	3 for K= 256, 6 for K= 512
pk_pass_fld	3

5. RESULTS AND DISCUSSION

Simulation has been run for each synthetic traffic pattern using different T/R values and several quantities are used to measure and compare the performance of the proposed congestion control algorithm. The primary ones are the average processor utilization (i.e. percent of time that threads are running), average channel waiting time (i.e. average waiting time of a packet until it is serviced by the channel), average input waiting time (i.e. average waiting time of a packet until it is serviced by the processor) and average network response time (i.e. the time between a request message is enqueued at the output channel and the corresponding data message is received in the input queue).

5.1. Results for Client-Server Traffic Model

Figure 5.1 through Figure 5.24 show the results for the average processor utilization, average channel waiting time, average input waiting time and average network response time before and after applying the 2-D congestion control algorithm using input, output and hybrid parameters for Client-Server traffic model under UN, HR and BR synthetic traffic patterns with 4 and 8 threads. Bypass and no-bypass operation case studies have been performed separately to all synthetic traffic patterns and performance measures are collected before and after applying the algorithm. The average rate of decrement / increment rates of the performance metrics for all synthetic traffic patterns using bypass and no-bypass operation case studies are shown in Figure 5.49 through Figure 5.52.

5.1.1. Average Processor Utilization Results for Client-Server Traffic Model

The average processor utilization results for Client-Server traffic model with bypass operation case are depicted in Figure 5.1, Figure 5.2 and Figure 5.3; with no-bypass operation case are depicted in Figure 5.13, Figure 5.14 and Figure 5.15. The average rate of increment rates of the average processor utilization results are

summarized in Figure 5.49.a and Figure 5.50.a for bypass operation case and Figure 5.51.a and Figure 5.52.a for no-bypass operation case.

The average processor utilization results of 4 threads and bypass operation case show the best performance improvement at HR traffic pattern that uses hybrid parameter (12.57%) and the least performance improvement is gained by BR traffic pattern that uses output parameter (2.99%). In UN traffic pattern (Figure 5.1.a); the hybrid parameter has higher processor utilization compared with input and output parameter. However at the T/R values less than 0.8, input parameter, output parameter and no algorithm measurements give results that are close to each other. At the T/R values of more than 1 in HR traffic pattern (Figure 5.2.a) and more than 1.4 in BR traffic pattern (Figure 5.3.a) hybrid and input parameter results show nearly the same performance improvement. Output parameter gives better results than no algorithm case although it has the worst performance among all parameters.

The average processor utilization results of 8 threads and bypass operation case present the best performance improvement at HR traffic pattern that uses hybrid parameter (13.91%) and the least performance improvement is gained by BR traffic pattern that uses output parameter (3.47%). In UN traffic pattern (Figure 5.1.b) and BR traffic pattern (Figure 5.3.b); the better performance results are obtained using input parameter at the low values of T/R. In HR traffic pattern (Figure 5.2.b); the average processor utilization yields lower values considering to UN and BR traffic patterns. At the low values of T/R, input parameter is better than hybrid and output parameters similar to UN and BR traffic patterns. All remaining values of T/R, hybrid parameter has the best performance improvement.

In Figure 5.51.a, the average processor utilization results of 4 threads and no-bypass operation case are summarized. The average rate of increment for processor utilization is the highest (3.94%) for UN traffic pattern that uses hybrid parameter and the lowest for HR traffic pattern that uses output parameter (1.59%). All of the three communication patterns (Figure 5.13.a, Figure 5.14.a, and Figure 5.15.a) yield lower performance improvements.

The average processor utilization results of 8 threads and no-bypass operation give the best performance improvement at HR traffic pattern that uses hybrid

parameter (4.62%) and the least performance improvement is gained by UN traffic pattern that uses output parameter (1.21%). As in the case of 4 threads with no-bypass operation, the increment rates of the average processor utilization do not achieve better performance than 4 and 8 threads with bypass operation cases for all synthetic traffic patterns (Figure 5.13.b, Figure 5.14.b, Figure 5.15.b).

For overall performance of the average processor utilization results for Client-Server Traffic Model is considerably influenced by the bypass operation and hybrid parameter. As a consequence, better results are obtained. This stems from the fact that checking the channel output queue for packet transmission has positive effect on performance improvement. There is not an exact performance difference between parameters but in terms of increment rates, hybrid parameter always has the best performance improvement.

5.1.2. Average Channel Waiting Time Results for Client-Server Traffic Model

The results for the average channel waiting time are indicated in Figure 5.4 through Figure 5.6 and Figure 5.16 through Figure 5.18. The most decrement rate of the average channel waiting times in 4 threads and bypass operation case (Figure 5.49.b), 8 threads and bypass operation case (Figure 5.50.b), 4 threads and no-bypass operation case (Figure 5.51.b) and 8 threads and no-bypass operation case (Figure 5.52.b) are obtained from UN (16.17%), HR (25.02%), UN (8.34%) and HR (14.06%) traffic patterns respectively. While all of the cases that give the best result use hybrid parameter, the performance of the cases that use input parameter follow subsequently.

As is seen from the figures, both bypass and no-bypass cases decrease the average channel waiting time. The performance of bypass cases that use hybrid parameter is better or approximately equal to the performance of other parameters. In no-bypass cases, the performance of the hybrid parameter is more noticeable. In all cases, input, output and hybrid parameter results are very close to results that collected before applying the algorithm at the T/R values of 0.2 to 0.8. The gap between the performances of no-algorithm and other parameters increases as T/R

increases after value of 0.8. This is due to the fact that level of the congestion problem in channel queues is very low in that interval. Therefore, the effect of the congestion control algorithm is not reflected on the results at the low values of T/R.

5.1.3. Average Input Waiting Time Results for Client-Server Traffic Model

Figure 5.7 through Figure 5.9 and Figure 5.19 through Figure 5.21 show the results for the average input waiting time before and after applying the congestion control algorithm with input, output and hybrid parameters. The maximum reduction of the average input waiting time after applying the algorithm is as much as 19.29% (HR) for 4 threads and bypass operation case (Figure 5.49.c), 27.85% (UN) for 8 threads and bypass operation case (Figure 5.50.c), 11.48% (UN) for 4 threads and no-bypass operation case (Figure 5.51.c) and 16.68% (UN) for 8 threads and no-bypass operation case (Figure 5.52.c). The results reveal that for all cases, except for 4 threads and bypass operation case, using hybrid parameters and UN traffic pattern together give the best performance.

From these figures it is noticed that the performance of all cases with input, output and hybrid parameters decreases the average input waiting time. However, the performance of no-bypass cases is the least affected due to lack of the positive effect of bypass operation. For bypass operation cases (Figure 5.7 through Figure 5.9), input and hybrid parameters perform similar results with respect to the output parameter because using only output parameter is insufficient to decrease the average input waiting time.

The average waiting times in the input queues are very long whereas the average waiting times in the output queues are very short at the low values of T/R. Therefore, the average input waiting times decrease as T/R increases. The effect of congestion control algorithm on the average input waiting time can be seen intensely at the low values of T/R.

5.1.4. Average Network Response Time Results for Client-Server Traffic Model

This section presents the results for the average network response time before and after applying the congestion control algorithm with input, output and hybrid parameters. The charts are depicted in Figure 5.10 through Figure 5.12 and Figure 5.22 through Figure 5.24. The average rate of decrement rates of the average network response time results are given in Figure 5.49.d and Figure 5.50.d for bypass operation cases and Figure 5.51.d and Figure 5.52.d for no-bypass operation cases. The proposed congestion control algorithm is able to decrease the average network response time using input, output and hybrid parameters on bypass and no-bypass cases with 4 and 8 threads. The case studies on the average network response time present the best performance improvement of 21.07% (HR) in 4 threads and bypass operation case, 30.27% (UN) in 8 threads and bypass operation case, 14.01% (UN) in 4 threads and no-bypass operation case and 19.18% (HR) in 8 threads and no-bypass operation case.

It should be noted that the main goal of the proposed congestion control algorithm is to decrease the average network latency by preventing congestion in input and output queues. In 4 threads and bypass operation case, hybrid parameter performs better than the other parameters at most of the values of T/R (Figure 5.10.a, Figure 5.11.a and Figure 5.12.a). In 8 threads and bypass operation case, the performance of hybrid and input parameter is comparable for several values of T/R: In UN (Figure 5.10.b) and BR (Figure 5.12.b) traffic patterns, input parameter is better than hybrid parameter at the T/R values of less than 1. The gap between the performances of hybrid and input parameter increases as T/R increases after value of 1. In 4 threads and no-bypass operation case, input parameter is better than hybrid parameter at the high values of T/R (Figure 5.22.a and Figure 5.24.a) to the contrary of bypass case. The same scenario is repeated in 8 threads and no-bypass operation case: hybrid parameter perform better than the other parameters only at the low values of T/R. Since, the effect of bypass operation on network latency is more noticeable at high values of T/R.

In general, the average rate of decrement in the input waiting and channel waiting times are reflected properly on the average decrement rate of network response time in all case studies.

5.2. Results for Asynchronous Message Passing Traffic Model

A send/receive pair in the message passing model is conceptually a one-way transfer from a source area specified by the source user process to a destination address specified by the destination user process. The asynchronous send and receive allow the greatest overlap between computation and message passing by returning control most quickly to the calling process. A blocking asynchronous receive is similar in that it returns control to the calling process only when the data it is receiving have been successfully removed from the system buffer and placed at the designated application address. Once it returns, the application can immediately use the data in the specified application buffer. Unlike a synchronous receive, a blocking receive does not send an acknowledgment to the sender. In both the asynchronous send and receive, however, the return of control does not imply anything about the state of the message or the application data structures it uses, so it is the user's responsibility to determine that state when necessary (Culler et al., 1998).

In this section, results are presented for Asynchronous Message Passing Traffic Model using UN, HR and BR synthetic traffic patterns and different number of threads (i.e. 4 and 8). Four case studies (i.e. 4 threads and bypass operation, 8 threads and bypass operation, 4 threads and no-bypass operation and 8 threads and no-bypass operation) have been performed to show the effect of the proposed congestion control algorithm. The results of the average processor utilization, average channel waiting time, average input waiting time and average network response time before and after applying the congestion control algorithm for Asynchronous Message Passing Traffic Model with 4 and 8 threads under four case studies are given in Figure 5.25 through Figure 5.48. Figure 5.53, Figure 5.54, Figure 5.55 and Figure 5.56 show the average rate of decrement / increment rates of performance measures for UN, HR and BR synthetic traffic patterns.

5.2.1. Average Processor Utilization Results for Asynchronous Message Passing Traffic Model

The average processor utilization results for Asynchronous Message Passing traffic model are indicated in Figure 5.25 through Figure 5.27 and Figure 5.37 through Figure 5.39. The highest increment rate of the average processor utilization results in 4 threads and bypass operation case (Figure 5.53.a), 8 threads and bypass operation case (Figure 5.54.a), 4 threads and no-bypass operation case (Figure 5.55.a) and 8 threads and no-bypass operation case (Figure 5.56.a) are obtained from HR (13.75%), HR (13.91%), HR (5.25%) and HR (6.95%) traffic patterns respectively. All cases succeed in increasing the average processor utilization. However, the performance of HR synthetic traffic pattern and hybrid parameter is better than other traffic patterns and parameters.

In 4 threads and bypass operation case, the performance of hybrid parameter is approximately equal to the performance of input and output parameter at very low and very high values of T/R (Figure 5.25.a, Figure 5.26.a and Figure 5.27.a). In 8 threads and bypass operation case, hybrid and input parameters perform close results at the low values of T/R (Figure 5.25.b, Figure 5.26.b and Figure 5.27.b). 4 threads and no-bypass operation case gives the worst performance among other cases (Figure 5.37.a, Figure 5.38.a and Figure 5.39.a). In 8 threads and no-bypass operation case, performance of input and output parameter is very close to each other (Figure 5.37.b and Figure 5.38.b). However, hybrid parameter gives better average processor utilization measurement with respect to input and output parameter in this case (Figure 5.39.b).

In general, most of average processor utilization results that are obtained from Asynchronous Message Passing traffic model are better than the average processor utilization results of Client-Server traffic model. The reason is that the performance of average processor utilization results relies on system's overall performance which is higher in Asynchronous Message Passing traffic model.

5.2.2. Average Channel Waiting Time Results for Asynchronous Message Passing Traffic Model

Figure 5.28 through Figure 5.30 and Figure 5.40 through Figure 5.42 show the results for the average channel waiting time using input, output and hybrid parameters under UN, HR and BR synthetic traffic patterns. The average rate of decrement rates of the average channel waiting time results are given in Figure 5.53.b and Figure 5.54.b for bypass operation cases and Figure 5.55.b and Figure 5.56.b for no-bypass operation cases. The case studies on the average channel waiting time present the best performance improvement of 18.07% (HR) in 4 threads and bypass operation case, 29.53% (HR) in 8 threads and bypass operation case, 12.55% (HR) in 4 threads and no-bypass operation case and 17.85% (HR) in 8 threads and no-bypass operation case. The hybrid parameter and HR synthetic traffic pattern perform the best performance improvement as it is seen in average processor utilization results.

In 4 and 8 threads bypass operation cases, input, output and hybrid parameters show very close performance improvement under all synthetic traffic patterns (Figure 5.28 through Figure 5.30). However, the performance of 8 threads and bypass operation case is better than 4 threads and bypass operation case. That is because; the effect of congestion control algorithm can be seen more at intensive traffics just as HR and 8 thread cases. The performance of 4 threads and no-bypass operation case is the least effected among other cases due to the negative impact of the no-bypass operation (Figure 5.40.a, Figure 5.41.a and Figure 5.42.a). The 8 threads and no-bypass operation case (Figure 5.40.b, Figure 5.41.b and Figure 5.42.b) is better than 4 threads and no-bypass operation case, it is not as effective as 4 and 8 threads bypass operation cases when it comes to reducing average channel waiting time.

The results that are obtained from input, output and hybrid parameters are very similar until T/R value of 1 for all cases. This is because; the waiting times in the output queues are too short between 0.2 and 1 interval of T/R. Therefore, the effect of the congestion control algorithm can be observed after T/R value of 1.

5.2.3. Average Input Waiting Time Results for Asynchronous Message Passing Traffic Model

The results in this section show the performance of the average input waiting time after applying the congestion control algorithm for Asynchronous Message Passing Traffic Model with 4 and 8 threads under four case studies. Figure 5.31 through Figure 5.33 and Figure 5.43 through Figure 5.45 demonstrate the average input waiting time results using input, output and hybrid parameters under UN, HR and BR synthetic traffic patterns. Figure 5.53.c and Figure 5.54.c summarize the average rate of decrement rates of the average input waiting time for bypass operation cases; Figure 5.55.c and Figure 5.56.c indicate the average rate of decrement rates of the average input waiting time for no-bypass operation cases.

The case studies on the average input waiting time present the best performance improvement of 22.06% (HR) in 4 threads and bypass operation case, 30.22% (UN) in 8 threads and bypass operation case, 14.65% (UN) in 4 threads and no-bypass operation case and 19.84% (HR) in 8 threads and no-bypass operation case.

As it is clearly seen from the Figures, the proposed congestion control algorithm has yielded lower average input waiting time for all traffic patterns and T/R values. The no-bypass operation cases are less affected than bypass operation cases by the proposed congestion control algorithm. The performance curves in 4 threads and bypass case (Figure 5.31.a, Figure 5.32.a and Figure 5.33.a) are more distinguishable from each other than the curves that are obtained from 4 threads and no-bypass operation case (Figure 5.43.a, Figure 5.44.a and Figure 5.45.a). This means, the performance of hybrid parameter is more noticeable in 4 threads and bypass operation case while it is approximately equal to the input parameter in 4 threads and no-bypass operation case. The performance of bypass and no-bypass operation cases with 8 threads is better than 4 threads cases. This stems from the fact that the congestion control algorithm is more responsive to the intensive traffic in input and output queues.

In all cases, the hybrid parameter has the best performance improvement while the output parameter has the worst performance. At the low values of T/R, the average waiting times in the input queues are very long whereas the average waiting times in the output queues are very short. For this reason, the performance curves are approximately equal to each other at high values of T/R.

5.2.4. Average Network Response Time Results for Asynchronous Message Passing Traffic Model

The average network response time for Asynchronous Message Passing traffic model is calculated by the sum of the average channel waiting time and the average channel service time. In every sending operation, nodes generate new packets without waiting for acknowledgment from destination nodes. Therefore, under Asynchronous Message Passing traffic model, average channel waiting time and T/R values are critical issues because of the definition of the average network response time.

The results for the average network response time are indicated in Figure 5.34 through Figure 5.36 and Figure 5.46 through Figure 5.48. The maximum reduction of the average network response time after applying the congestion control algorithm is as much as 44.71% (HR) for 4 threads and bypass operation case (Figure 5.53.d), 52.09% (HR) for 8 threads and bypass operation case (Figure 5.54.d), 30.49% (UN) for 4 threads and no-bypass operation case (Figure 5.55.d) and 41.69% (UN) for 8 threads and no-bypass operation case (Figure 5. 56.d).

The proposed congestion control algorithm gives a significantly reduced average network response time for input, output and hybrid parameters under various synthetic traffic patterns and different number of threads (i.e. 4 and 8). The results reveal that for all cases, the hybrid parameter gives the best performance improvement among other parameters and the performance of bypass operation cases is better than no-bypass operation cases. The reason is that the average network response time relies on average channel waiting time as it is mentioned before. Therefore, the performance of no-bypass operation cases for average channel waiting

time affect the average network response time results in this traffic model. As it is stated previously, the effect of the algorithm reflected on the results is more than the Client-Server traffic model due to the definition of the average network response time.

6. CONCLUSIONS

In this thesis, a congestion control algorithm that prevents congestion in 2-D broadcast-based multiprocessor architectures with multiple input queues was proposed. The performance of the algorithm is tested via simulation using synthetic traffic patterns (UN, HR and BR) on a 64-node 2-D SOME-Bus multiprocessor architecture for Client-Server and Asynchronous Message Passing traffic models. We proved the effectiveness of our approach using four evaluation cases: 4 threads and bypass operation, 8 threads and bypass operation, 4 threads and no-bypass operation and 8 threads and no-bypass operation. Performance measures such as average processor utilization, average channel waiting time, average input waiting time and average network response time have been collected before and after applying the algorithm.

Several of the issues outlined in this thesis offer opportunities for further investigations. These are summarized below:

- In all cases and synthetic traffic patterns, the hybrid parameter gives the best performance improvement. The input and output parameters follow the hybrid parameter subsequently. In some cases, the performance of input parameter is approximately equal to the performance of hybrid parameter. This means that using input queue parameters as only input or hybrid form is very important in regards to improving performance. This conclusion supports the results that are obtained from 1-D congestion control algorithm (Aci and Akay, 2010).
- In all performance measures except average processor utilization, the performance of 8 threads cases is better than 4 threads cases. The reason is that the 2-D congestion control algorithm is more responsive to the intensive traffic in input and output queues.
- For overall performance of the all performance measures except average processor utilization, the bypass cases (i.e. 4 and 8 threads) achieve better performance than the no-bypass cases (i.e. 4 and 8 threads). This stems from

the fact that checking the channel output queue for packet transmission has positive effect on performance improvement.

- Under Asynchronous Message Passing traffic model; all of the performance measures except average processor utilization give better results than Client-Server traffic model. The decrement rate of the average network response time is better or approximately equal to the decrement rate of the average input waiting time under Client-Server traffic model. The decrement rate of the average network response time is nearly 2 times more than the decrement rate of the average input waiting time under Asynchronous Message Passing traffic model because of the modified definition of the average network response time.
- There is not an exact superiority of Asynchronous Message Passing traffic model on Client-Server traffic model or bypass operation on no-bypass operation or 8 threads cases on 4 threads cases with regard to the performance of average processor utilization. The reason is that the most of average processor utilization values are in interval of 0.9 and 1. Therefore, the average rate of the increment rates is very close to each other.
- In all cases and synthetic traffic patterns, the average input waiting times decrease as T/R increases whereas the average channel waiting times increase as T/R increases under both Asynchronous Message Passing and Client-Server traffic models. This is due to the fact that waiting times are very long in the input queues while the waiting times are very short in the output queues at the low values of T/R.

These results show that congestion occurring in input and output queues causes performance degradation in multiprocessor architectures using message passing paradigm and the suggested congestion control algorithm is able to increase the architecture's performance.

REFERENCES

ABBAS, H.M. AND BAYOUMI, M.M., 2002. Parallel codebook design for vector quantization on a message passing MIMD architecture. Parallel Computing, 28(7–8):1079–1093.

ACI, C.I. AND AKAY, M.F., 2010. A new congestion control algorithm for improving the performance of a broadcast-based multiprocessor architecture. Journal of Parallel and Distributed Computing, 70(9): 930–940.

AKAY, M.F., AND ABASIKELES, I., 2010. Predicting the performance measures of an optical distributed shared memory multiprocessor by using support vector regression. Expert Systems with Applications, 37(9):6293–6301.

ANDERSON, E., BROOKS, J., GRASSL, C. AND SCOTT, S., 1997. Performance of the CRAY T3E multiprocessor. Proceedings of the 1997 ACM/IEEE conference on Supercomputing, 1–17.

BANI-MOHAMMAD, S., ABABNEH, I. AND HAMDAN, M., 2011. Performance evaluation of noncontiguous allocation algorithms for 2D mesh interconnection networks. Journal of Systems and Software, 84(12):2156–2170.

BAYDAL, E., LOPEZ, P. AND DUATO, J., 2005. A family of mechanisms for congestion control in wormhole networks. IEEE Transactions on Parallel and Distributed Systems, 16(9):772–784.

BAYDAL, E., LOPEZ, P. AND DUATO, J., 2000. A simple and efficient mechanism to prevent saturation in wormhole networks. Parallel and Distributed Processing Symposium, 617–622.

BHUYAN, L.N. AND AGRAWAL, D.P., 1984. Generalized Hypercube and Hyperbus Structures for a Computer Network. IEEE Transactions on Computers, 33(4):323–333.

BLUMRICH, M., CHEN, D., COTEUS, P., GARA, A. AND GIAMPAPA, M., 2003. Design and Analysis of the BlueGene/L Torus Interconnection Network. IBM Research Report, RC23025.

BOMANS, L. AND ROOSE, D., 1989. Benchmarking the iPSC/2 hypercube multiprocessor. Concurrency: Practice and Experience, 1(1):3–18.

BOUZID, A. AND ABUSHAGUR, M.A.G., 1996. Thin-film approximate modeling of in-core fiber gratings. Optical Engineering, 35(10):2793–2797.

CHAN, F., CAO, J. AND SUN, Y., 2003. High-level abstractions for message-passing parallel programming. Parallel Computing, 29(11–12):1589–1621.

CHANG, H.K., 2004. Modeling the effect of hot spot contention on crossbar multiprocessors. Computer Standards & Interfaces, 26(5):483–488.

CHANG, X., 1999. Network simulations with OPNET. Simulation Conference Proceedings, 307–314.

CHEN, Z., FAGG, G.E., GABRIEL, E., LANGOU, J., ANGSKUN, T., BOSILCA, G. AND DONGARRA, J., 2005. Fault tolerant high performance computing by a coding approach. Proceedings of the tenth ACM SIGPLAN symposium on principles and practice of parallel programming. 213–223.

COHEN, W.E., HYDE, D.W. AND GAEDE, R.K., 2000. An optical bus-based distributed dynamic barrier mechanism. IEEE Transactions on Computers, 49(12):1354–1365.

CULLER, D., SINGH, J. P. AND GUPTA, A., 1998. Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann, 1056 p.

DALLY, W.J. AND AOKI, H., 1993. Deadlock-free adaptive routing in multicomputer networks using virtual channels. IEEE Transactions on Parallel and Distributed Systems, 4(4):466–475.

DALLY, W.J., TOWLES, B.P., 2004. Principles and Practices of Interconnection Networks, Morgan Kaufmann, 550 p.

DANESHTALAB, M., EBRAHIMI, M., LILJEBERG, P., PLOSILA, J. AND TENHUNEN, H., 2010. Input-Output Selection Based Router for Networks-on-Chip. IEEE Computer Society Annual Symposium on VLSI, 92–97.

DANESHTALAB, M., EBRAHIMI, M., LILJEBERG, P., PLOSILA, J. AND TENHUNEN, H., 2012a. A systematic reordering mechanism for on-chip networks using efficient congestion-aware method. Journal of Systems Architecture, In press.

DANESHTALAB, M., KAMALI, M., EBRAHIMI, M., MOHAMMADI, S., AFZALI-KUSHA, A., PLOSILA, J., 2012b. Adaptive Input-Output Selection Based On-Chip Router Architecture. Journal Low Power Electronics, 8(1):11–29.

DARWISH, M.G., RADWAN, A.A., ABD EL-BAKY, M.A. AND HAMED, K., 2008. TTPM – An efficient deadlock-free algorithm for multicast communication in 2D torus networks. Journal of Systems Architecture, 54(10):919–928.

DUATO, J., YALAMANCHILI, S., NI, L.M., 1997. Interconnection Networks: An Engineering Approach, IEEE Computer Society, 515 p.

EL-REWINI, H., ABD-EL-BARR, M., 2005. Advanced Computer Architecture and Parallel Processing, Wiley-Interscience, 288 p.

FAGG, G.E., GABRIEL, E., BOSILCA, G., ANGSKUN, T., CHEN, ZHIZHONG, PJESIVAC-GRBOVIC, J., LONDON, K. AND DONGARRA, J.J., 2004. Extending the MPI Specification for Process Fault Tolerance on High Performance Computing Systems. Innovatives Supercomputing in Deutschland, 2(1):1–24.

FOSS, T., STENSRUD, E., KITCHENHAM, B. AND MYRTVEIT, I., 2003. A simulation study of the model evaluation criterion MMRE. IEEE Transactions on Software Engineering, 29(11):985–995.

GALLES, M., 1997. Spider: a high-speed network interconnect. IEEE Micro, 17(1):34–39.

GEIST, A., BEGUELIN, A., DONGARRA, J., JIANG, W., MANCHEK, R. AND SUNDERAM, et.al., 1994. PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing, The MIT Press, 299 p.

GOH, L.K. AND VEERAVALLI, B., 2008. Design and performance evaluation of combined first-fit task allocation and migration strategies in mesh multiprocessor systems. Parallel Computing, 34(9): 508–520.

GORLATCH, S., 2002. Message passing without send–receive. Future Generation Computer Systems, 18(6):797–805.

GRAMA, A., KARYPIS, G., KUMAR, V. AND GUPTA, A., 2003. Introduction to Parallel Computing, Addison-Wesley, 656 p.

GROPP, W., HUSS-LEDERMAN, S., LUMSDAINE, A., LUSK, E.L., NITZBERG, B., SAPHIR, W., SNIR, M., 1998. MPI: The Complete Reference, The MIT Press, 362 p.

GU, H., WANG, K., KE, C., WANG, CHANGSHAN AND KANG, G., 2007. A New Inner Congestion Control Mechanism in Terabit Routers. Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 178–181.

GURSOY, A. AND KALE, L.V., 2004. Performance and modularity benefits of message-driven execution. Journal of Parallel and Distributed Computing, 64(4):461–480.

HAWKINS, C., SMALL, B.A., WILLS, D.S. AND BERGMAN, K., 2007. The Data Vortex, an All Optical Path Multicomputer Interconnection Network. IEEE Transactions on Parallel and Distributed Systems, 18(3):409–420.

HECHT, D. AND KATSINIS, C., 2003. Performance analysis of a fault-tolerant distributed-shared memory protocol on the SOME-bus multiprocessor architecture. Parallel and Distributed Processing Symposium, 1–8.

HEMENWAY, R., GRZYBOWSKI, R., MINKENBERG, C. AND LUIJTEN, R., 2004. Optical-packet-switched interconnect for supercomputer applications. Journal of Optical Networking, 3(12):900–913.

HENNESSY, J.L., PATTERSON, D.A., 2006. Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 704 p.

HUANG, W., KOOP, M.J., GAO, Q. AND PANDA, D.K., 2007. Virtual machine aware communication libraries for high performance computing.

Proceedings of the 2007 ACM/IEEE conference on Supercomputing, 9:1–12.

HWANG, K., XU, Z. AND ARAKAWA, M., 1996. Benchmark evaluation of the IBM SP2 for parallel signal processing. IEEE Transactions on Parallel and Distributed Systems, 7(5): 522–536.

IZU, C., MIGUEL-ALONSO, J. AND GREGORIO, J.A., 2005. Evaluation of Interconnection Network Performance Under Heavy Non-uniform Loads, Distributed and Parallel Computing, 3719:396–405.

JAIN, R., 1998. MYTHS ABOUT CONGESTION MANAGEMENT IN HIGH SPEED NETWORKS. Internetworking: Research and Experience, 3:101–113.

JIN, H., JESPERSEN, D., MEHROTRA, P., BISWAS, R., HUANG, L. AND CHAPMAN, B., 2011. High performance computing using MPI and OpenMP on multi-core parallel systems. Parallel Computing, 37(9):562–575.

JU, X. AND YANG, L., 2012. Performance analysis and comparison of 2 × 4 network on chip topology. Microprocessors and Microsystems, 36(6):505–509.

KATSINIS, C., 2004. Merging, sorting and matrix operations on the SOME-bus multiprocessor architecture. Future Generation Computing Systems, 20(4):643–661.

KATSINIS, C., 2001. Performance analysis of the simultaneous optical multi-processor exchange bus. Parallel Computing, 27(8):1079–1115.

KATSINIS, C. AND HECHT, D., 2004. Fault-tolerant DSM on the SOME-Bus multiprocessor architecture with message combining. Parallel and Distributed Processing Symposium, 210–213.

KATSINIS, C. AND NABET, B., 2004. A Scalable Interconnection Network Architecture for Petaflops Computing. The Journal of Supercomputing, 27(2):103–128.

KESSLER, R.E. AND SCHWARZMEIER, J.L., 1993. Cray T3D: a new dimension for Cray Research. Compcon Spring ’93, 176–182.

KIM, J.H., LIU, Z. AND CHIEN, A.A., 1994. Compressionless routing: a framework for adaptive and fault-tolerant routing. SIGARCH Computer Architecture News, 22(2):289–300.

KONSTANTINIDOU, S. AND SNYDER, L., 1994. The Chaos router. IEEE Transactions on Computers, 43(12): 1386–1397.

KORNAROS, G. AND ORPHANOUDAKIS, T., 2010. Design and implementation of high-speed buffered crossbars with efficient load balancing for multi-core SoCs. Microprocessors and Microsystems, 34(7–8):301–315.

KUMAR, V.S., NANJUNDIAH, R., THAZHUTHAVEETIL, M.J. AND GOVINDARAJAN, R., 2008. Impact of message compression on the scalability of an atmospheric modeling application on clusters. Parallel Computing, 34(1):1–16.

LI, M., ZENG, Q.A. AND JONE, W.B., 2006. DyXY: a proximity congestion-aware deadlock-free dynamic routing method for network on chip. Proceedings of the 43rd annual Design Automation Conference, 849–852.

LOPEZ, P., MARTINEZ, J.M. AND DUATO, J., 1998. DRIL: dynamically reduced message injection limitation mechanism for wormhole networks. International Conference on Parallel Processing, 535–542.

LOPEZ, P., MARTINEZ, J.M., DUATO, J. AND PETRINI, F., 1997. On the Reduction of Deadlock Frequency by Limiting Message Injection in Wormhole Networks. Parallel Computer Routing and Communication, 1417:295–307.

LOTFI-KAMRAN, P., RAHMANI, A.M., DANESHTALAB, M., AFZALI-KUSHA, A. AND NAVABI, Z., 2010. EDXY - A low cost congestion-aware routing algorithm for network-on-chips. Journal of System Architecture, 56(7):256–264.

MIGUEL-ALONSO, J., IZU, C. AND GREGORIO, J.A., 2008. Improving the performance of large interconnection networks using congestion-control mechanisms. Performance Evaluation, 65(3-4):203–211.

MORRISON, R.S., 2003. Cluster Computing-Architectures, Operating Systems, Parallel Processing & Programming Languages, GNU General Public Licence, 149p.

PAKIN, S., LAURIA, M. AND CHIEN, A., 1995. High performance messaging on workstations: Illinois fast messages (FM) for Myrinet. Proceedings of the 1995 ACM/IEEE conference on Supercomputing, 55a.

PARK, S.J., HENZ, B. AND SHIRES, D., 2007. Reconfigurable Computing for High Performance Computing Computational Science. High Performance Computing Modernization Program Users Group Conference, 350–358.

PETRINI, F. AND VANNESCHI, M., 1996. Minimal adaptive routing with limited injection on Toroidal k-ary n-cubes. Proceedings of the 1996 ACM/IEEE conference on Supercomputing, 23.

ROIG, C., RIPOLL, A. AND GUIRADO, F., 2007. A New Task Graph Model for Mapping Message Passing Applications. IEEE Transactions on Parallel and Distributed Systems, 18(12):1740–1753.

SHIN, J., AND PINKSTON, T.M., 2003. The Performance of Routing Algorithms Under Bursty Traffic Loads. SMART Interconnects Group, 1–14.

SINGH, J.P, WEBER, W.D. AND GUPTA, A., 1992. SPLASH: Stanford parallel applications for shared-memory. SIGARCH Computer Architecture News, 20(1): 5–44.

SMAI, A.H. AND THORELLI, L.E., 1998. Global reactive congestion control in multicomputer networks. 5th International Conference On High Performance Computing, 179–186.

TAN, M., ROSENBERG, P., YEO, J.S., MCLAREN, M., MATHAI, S., MORRIS, T., STRAZNICKY, J., 2008. A high-speed optical multi-drop bus for computer interconnections. Applied Physics, 95(4):945–953.

THOTTETHODI, M., LEBECK, A.R. AND MUKHERJEE, S.S., 2004. Exploiting global knowledge to achieve self-tuned congestion control for k-ary n-cube networks. IEEE Transactions on Parallel and Distributed Systems, 15(3):257–272.

THOTTETHODI, M., LEBECK, A.R. AND MUKHERJEE, S.S., 2001. Self-tuned congestion control for multiprocessor networks. The Seventh International Symposium on High-Performance Computer Architecture, 107–118.

TRUMLER, W., SCHLINGMANN, S., UNGERER, T., BAHN, J.H., BAGHERZADEH, N., 2008. Self-optimized Routing in a Network on-a-Chip. Biologically-Inspired Collaborative Computing, 268:199–212.

VAN DEN BRAND, J.W., CIORDAS, C., GOOSSENS, K. AND BASTEN, T., 2007. Congestion-Controlled Best-Effort Communication for Networks-on-Chip. Design, Automation Test in Europe Conference Exhibition, 1–6.

WANG, C., HU, W.H. AND BAGHERZADEH, N., 2012. A load-balanced congestion-aware wireless network-on-chip design for multi-core platforms. Microprocessors and Microsystems, 36(7):555–570.

ZHU, M., NARRAVULA, H., KATSINIS, C. AND HECHT, D., 2004. A channel caching scheme on an optical bus-based distributed architecture. Parallel and Distributed Processing Symposium, 233.

BIOGRAPHY

Çiğdem (İnan) ACI was born in Adana in 1984. She received her BSc degree in Computer Engineering, Fırat University in 2007. She completed MSc program of the Computer Engineering, Çukurova University in 2009. She started to PhD education in 2009 and completed in 2013, in Electrical and Electronics Engineering, Çukurova University. Her research interests are parallel computing and multiprocessor architectures.

APPENDICES

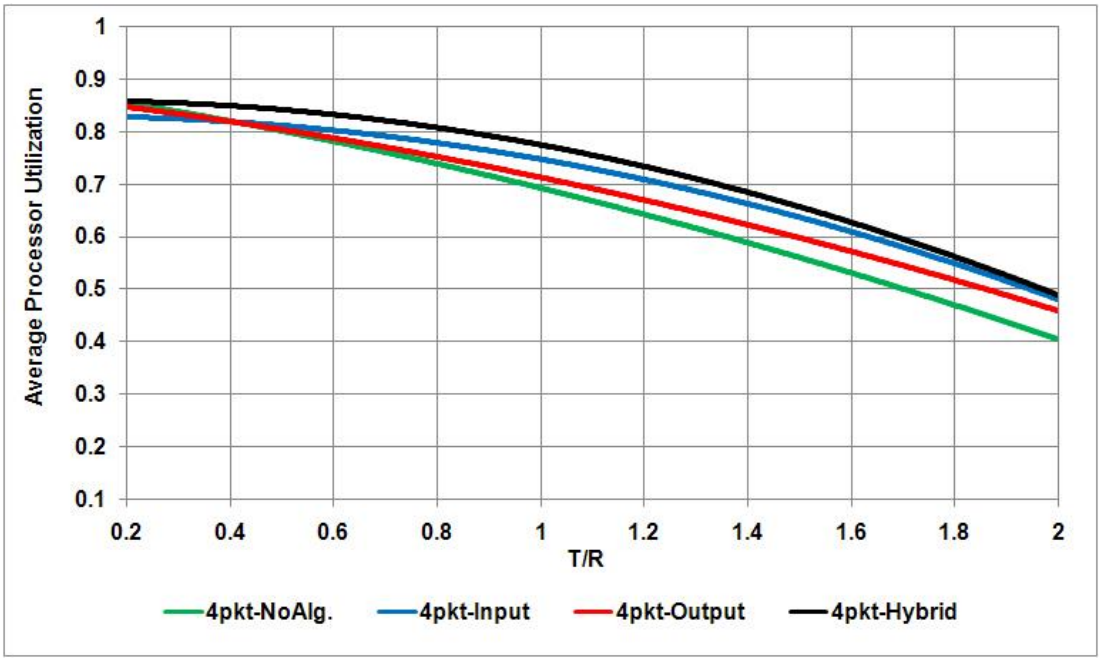

Figure 5.1.a. Average processor utilization for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

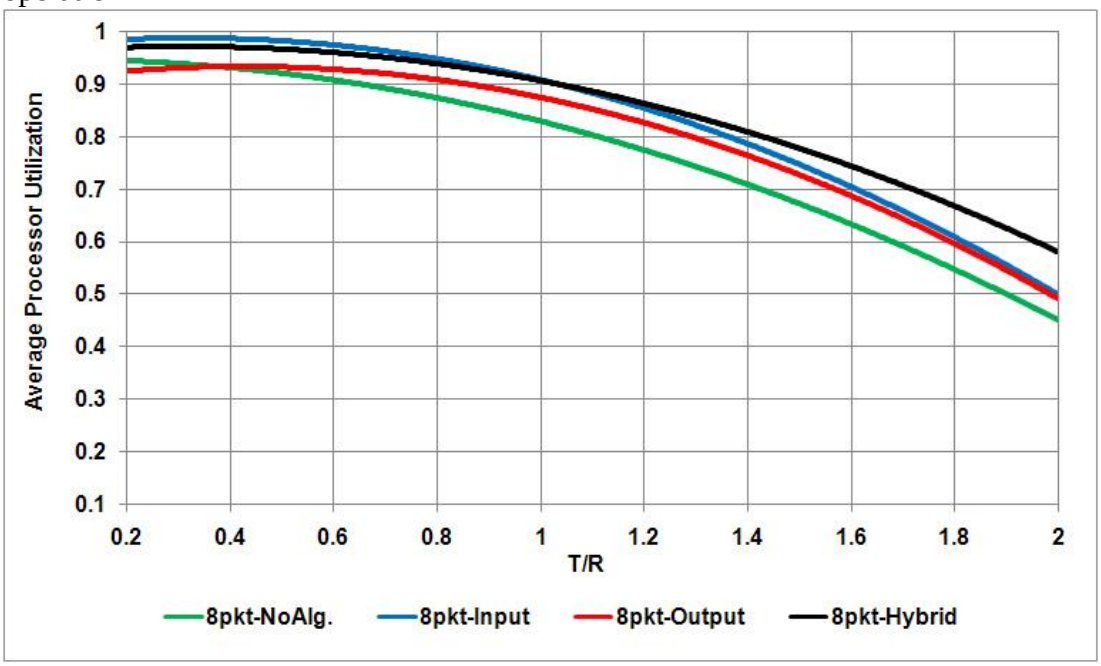

Figure 5.1.b. Average processor utilization for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

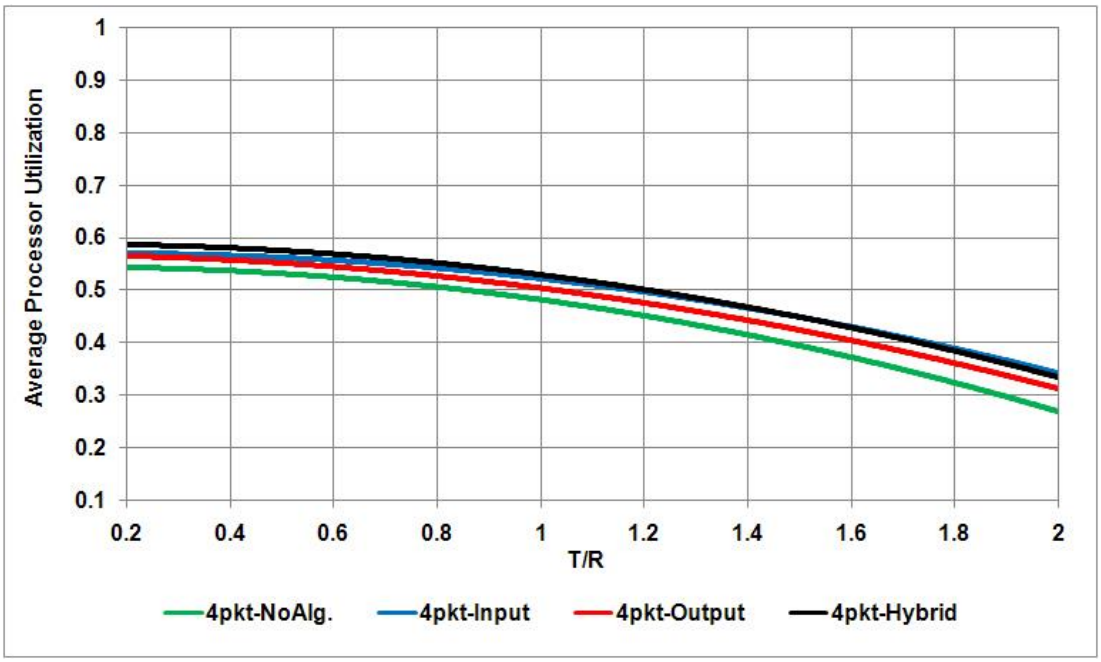

Figure 5.2.a. Average processor utilization for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

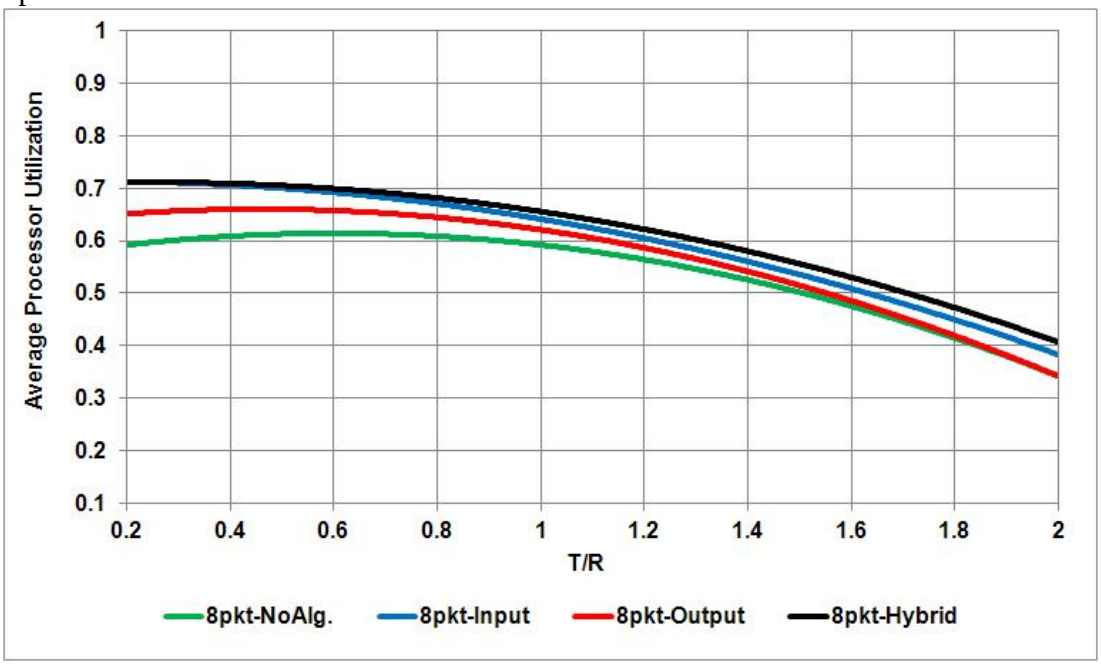

Figure 5.2.b. Average processor utilization for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

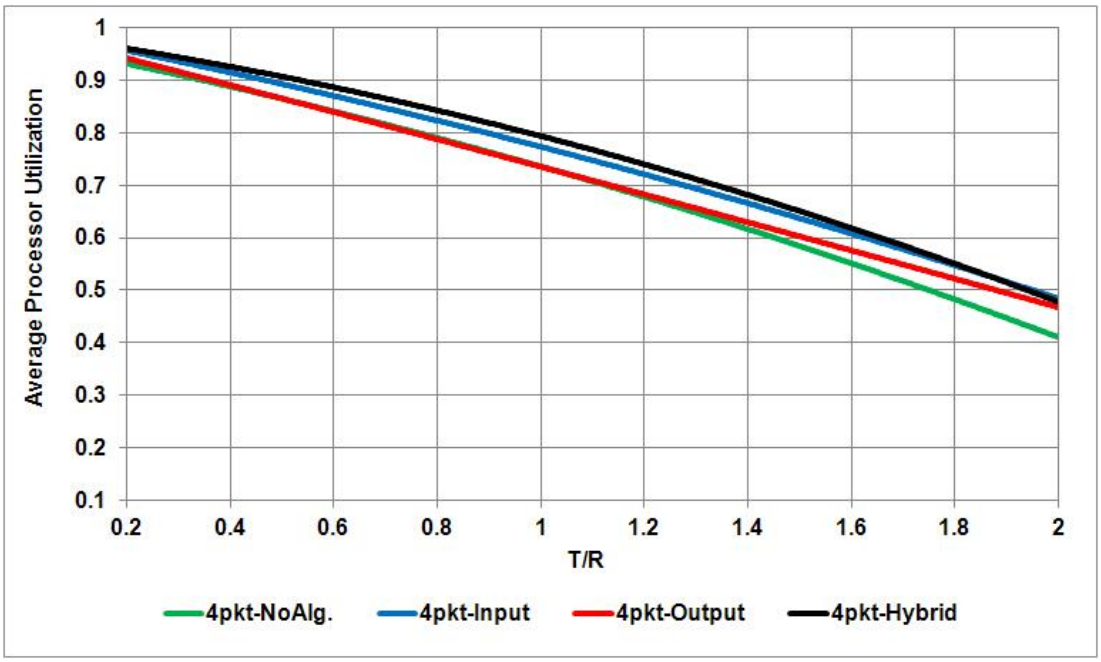

Figure 5.3.a. Average processor utilization for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

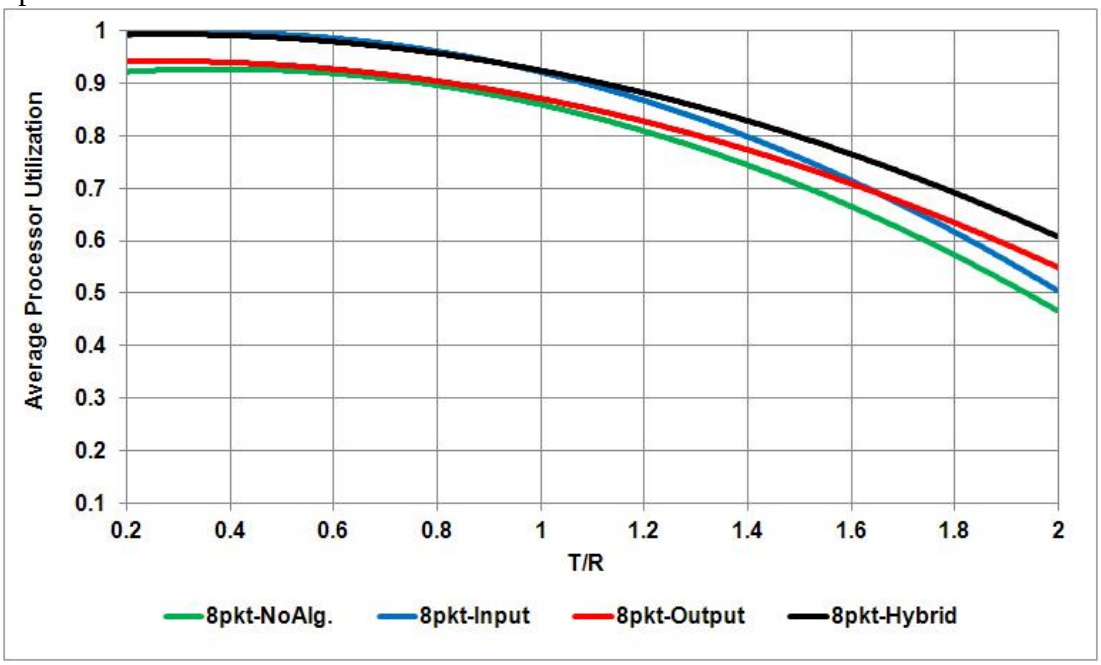

Figure 5.3.b. Average processor utilization for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

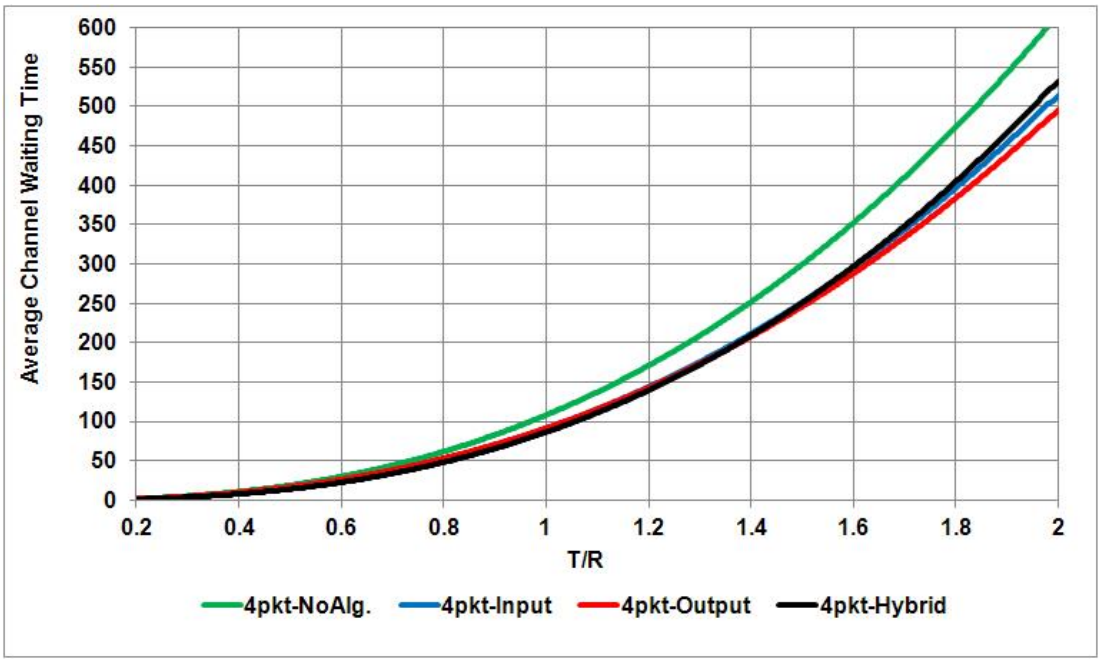

Figure 5.4.a. Average channel waiting time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

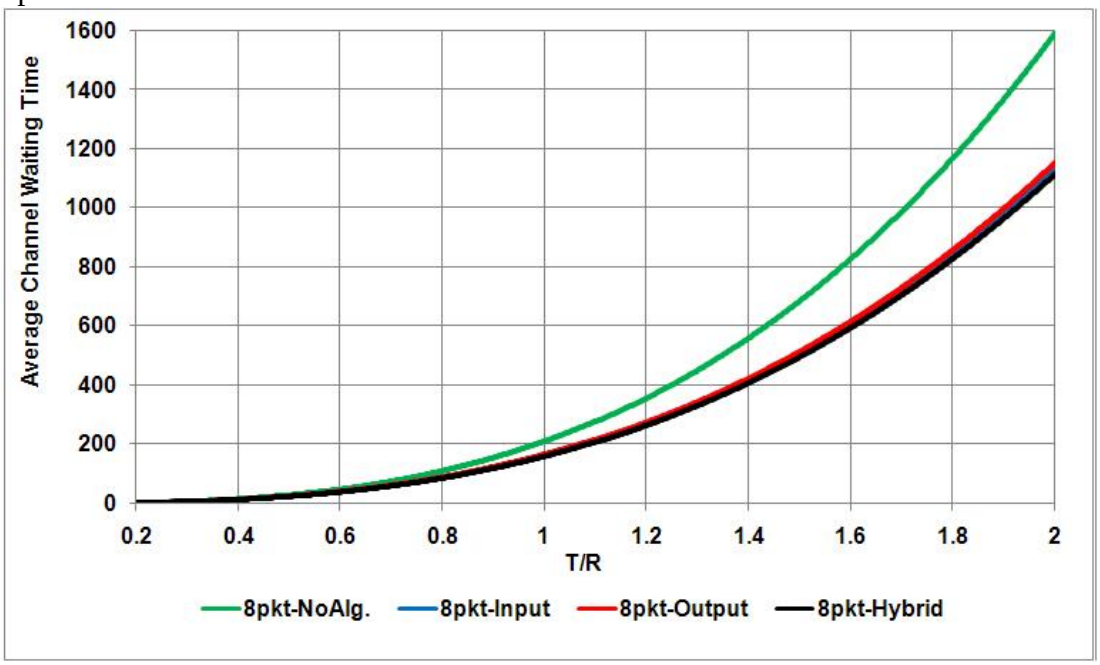

Figure 5.4.b. Average channel waiting time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

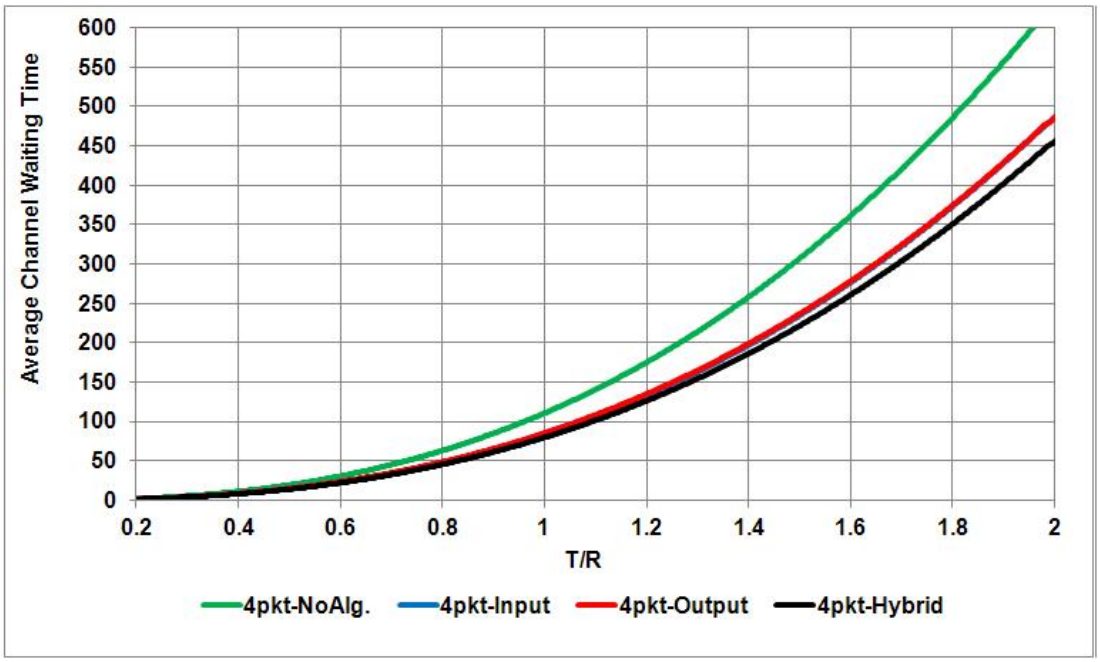

Figure 5.5.a. Average channel waiting time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

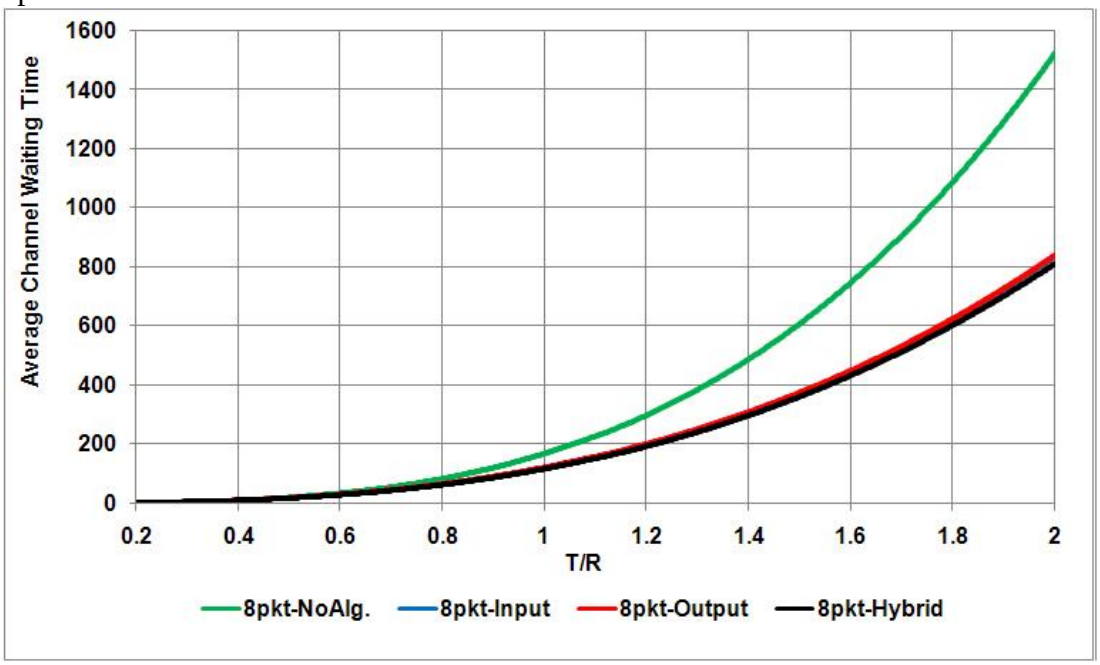

Figure 5.5.b. Average channel waiting time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

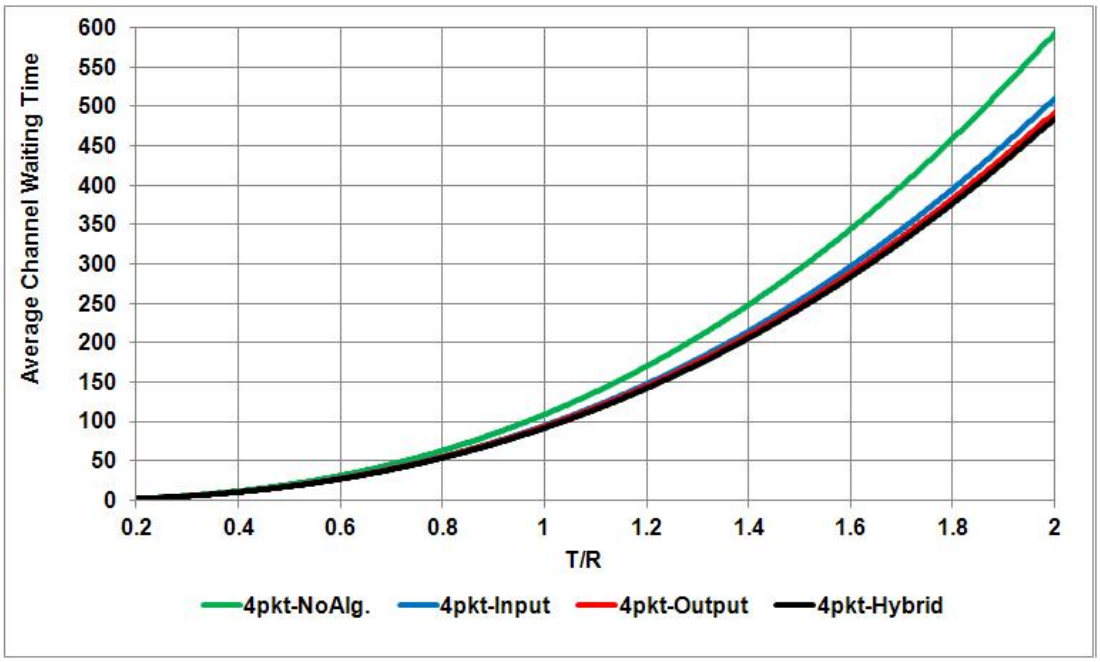

Figure 5.6.a. Average channel waiting time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

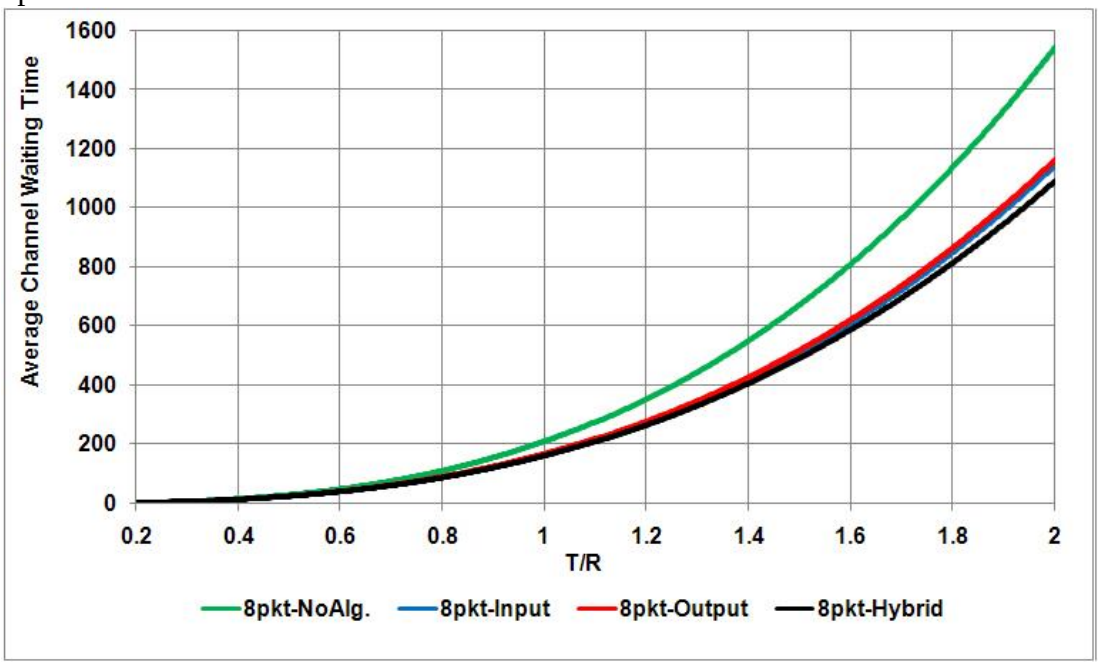

Figure 5.6.b. Average channel waiting time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

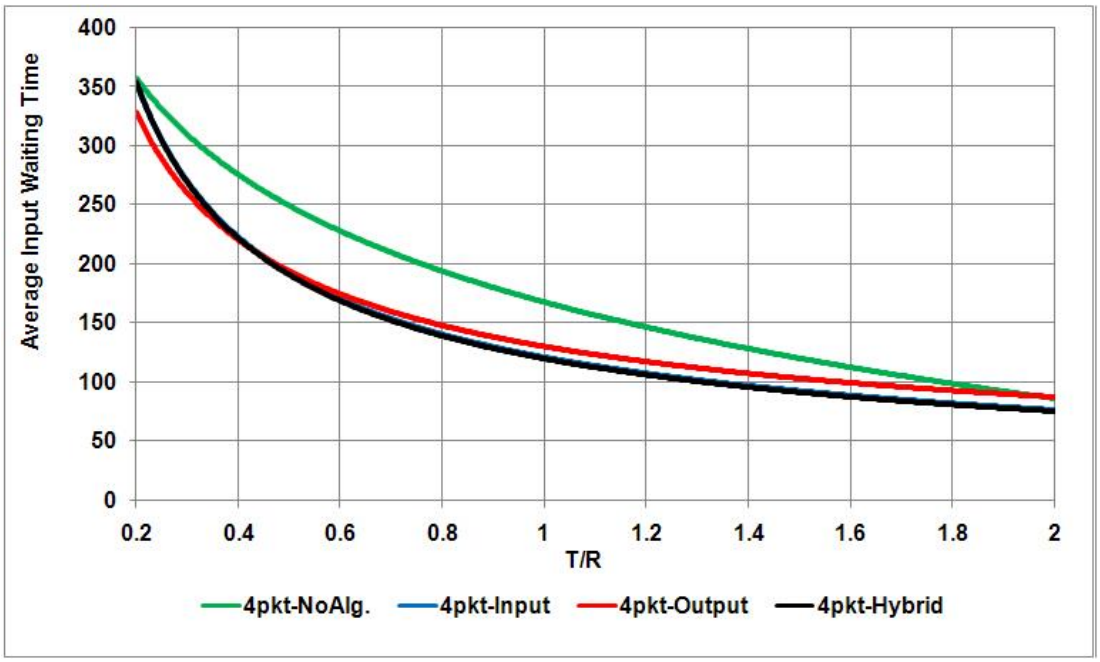

Figure 5.7.a. Average input waiting time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

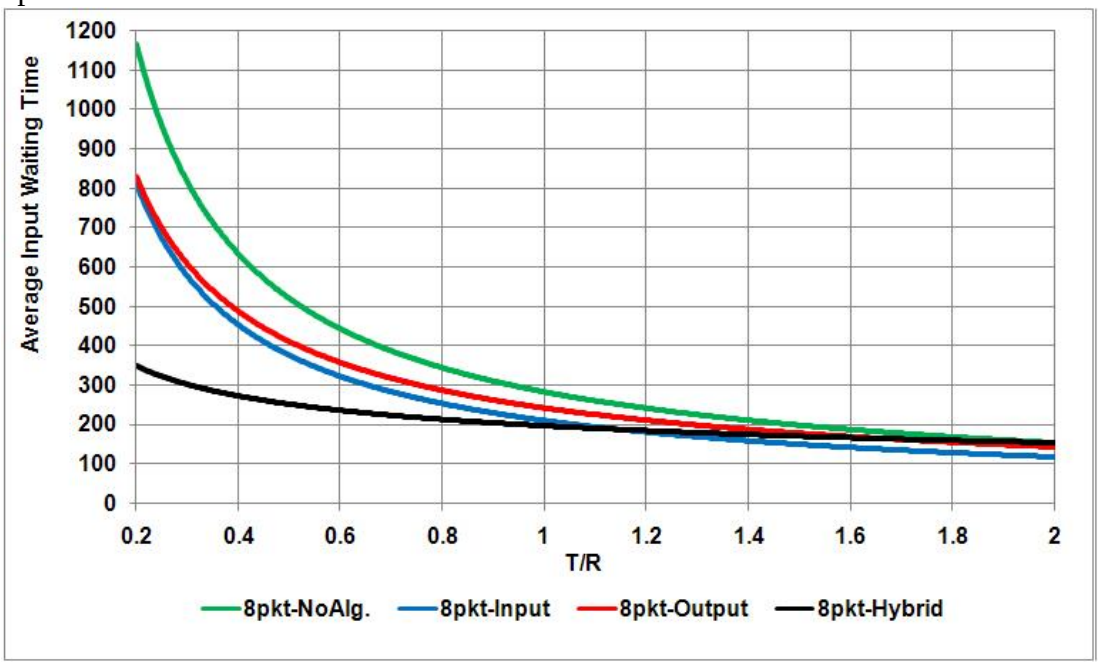

Figure 5.7.b. Average input waiting time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

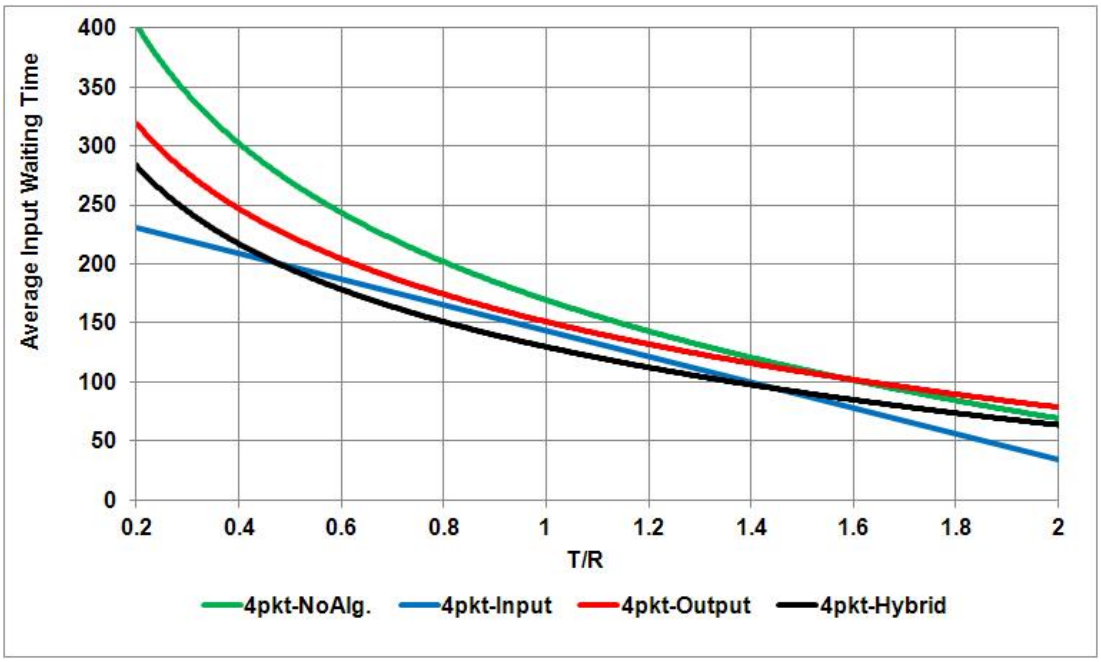

Figure 5.8.a. Average input waiting time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

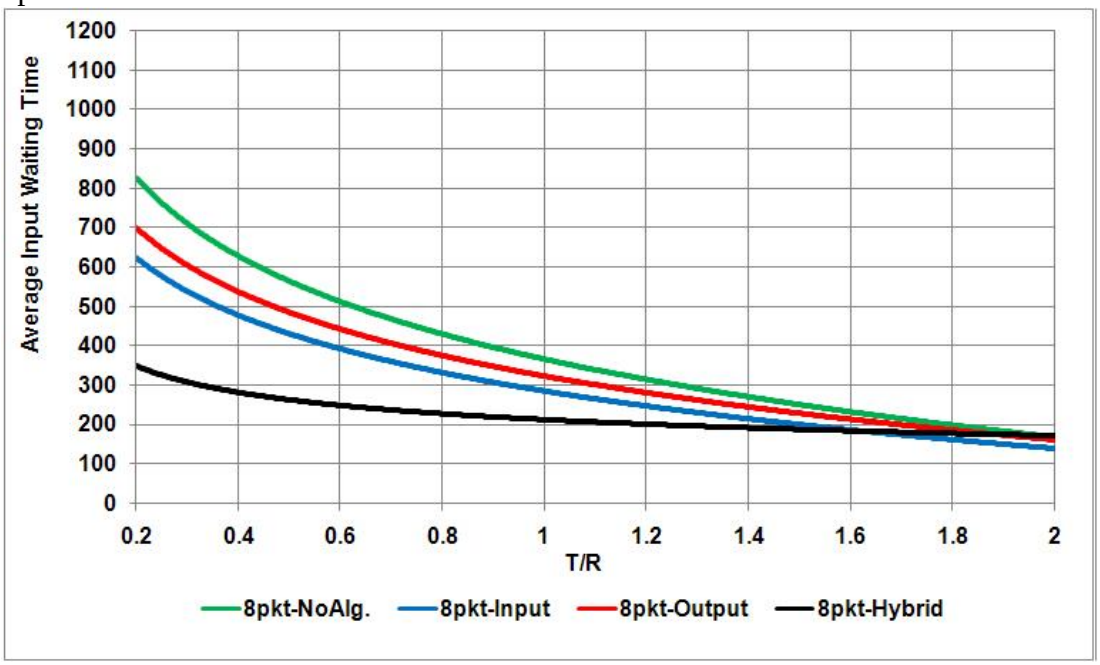

Figure 5.8.b. Average input waiting time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

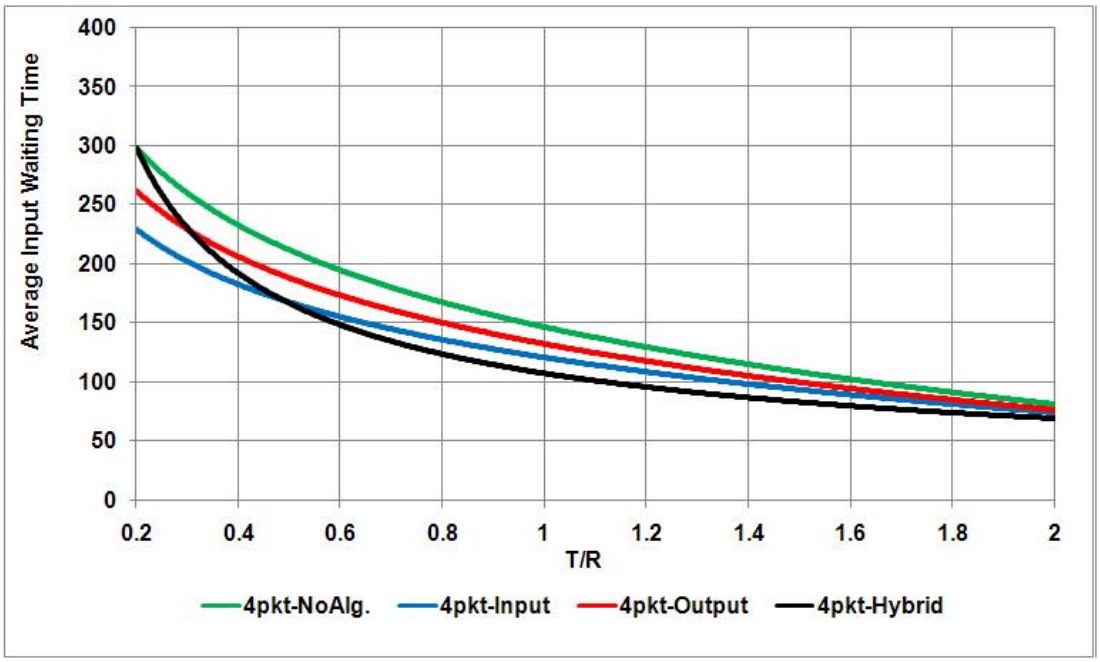

Figure 5.9.a. Average input waiting time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

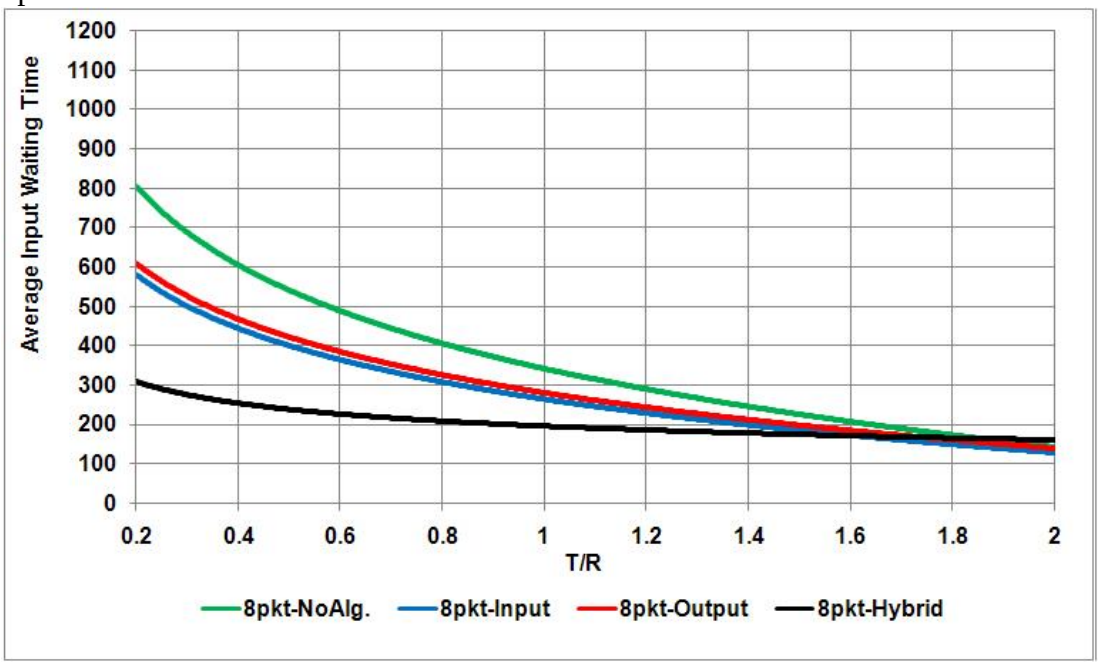

Figure 5.9.b. Average input waiting time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

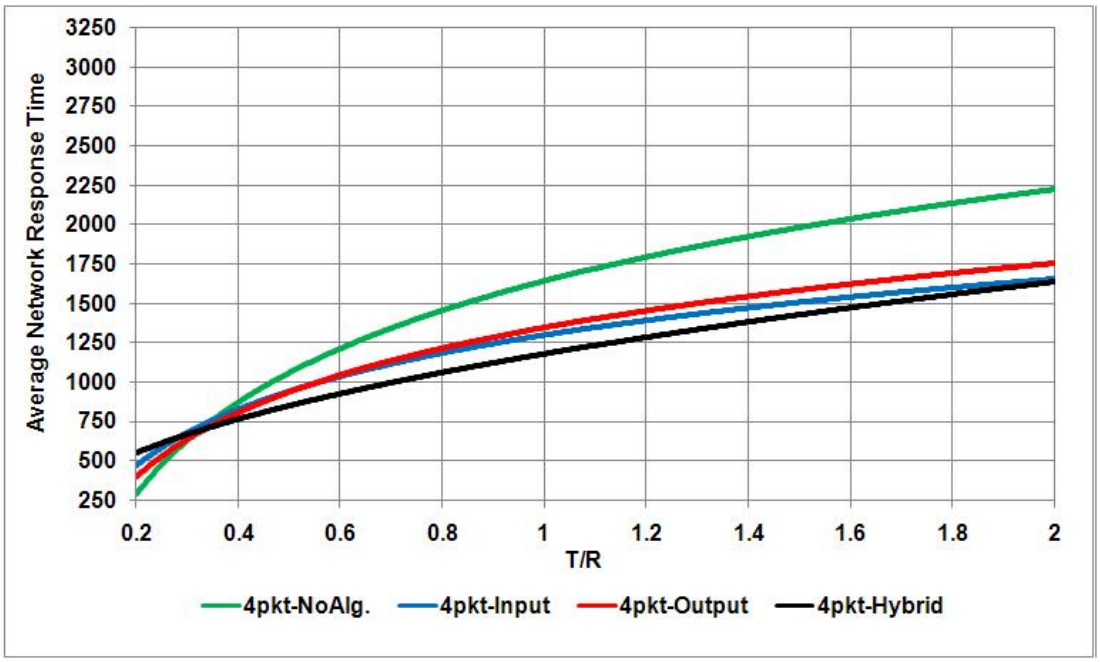

Figure 5.10.a. Average network response time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

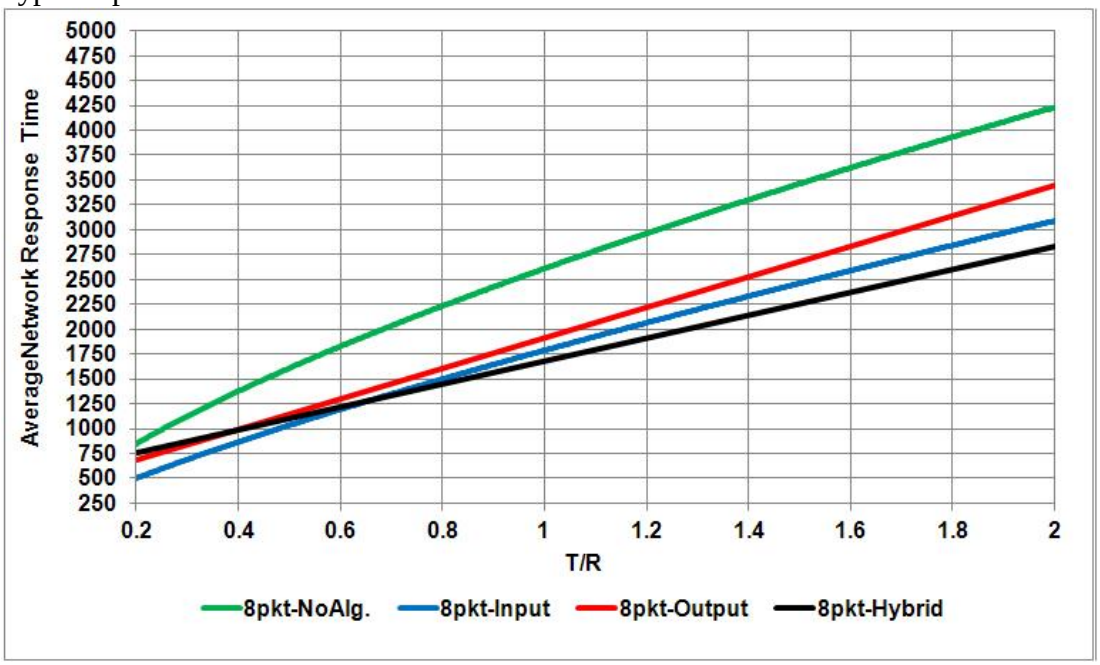

Figure 5.10.b. Average network response time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

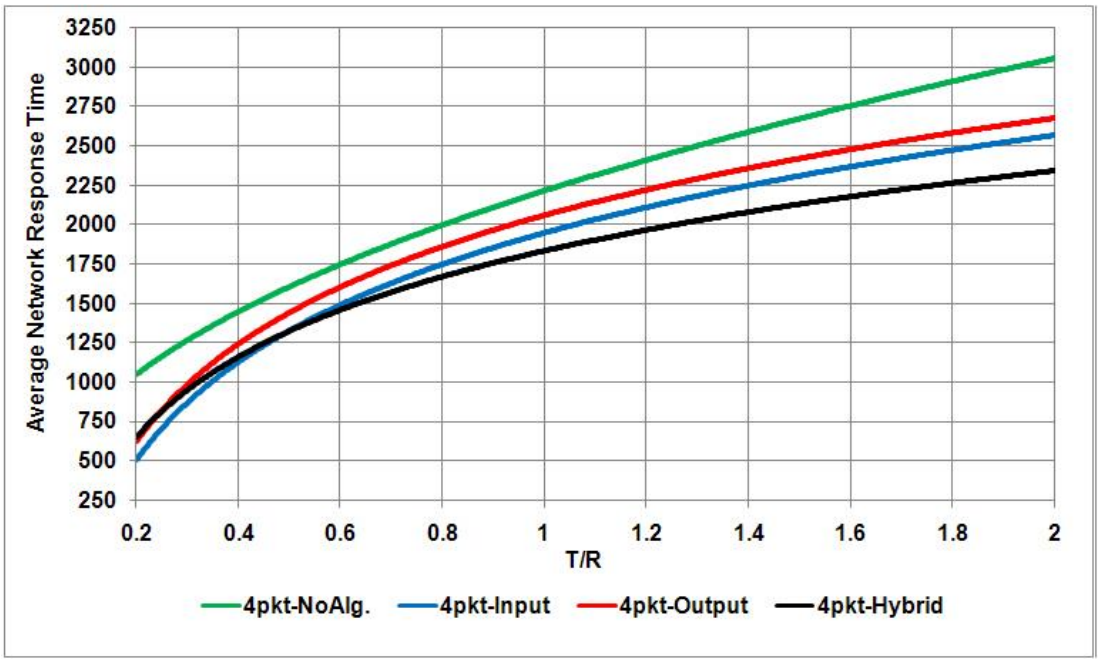

Figure 5.11.a. Average network response time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

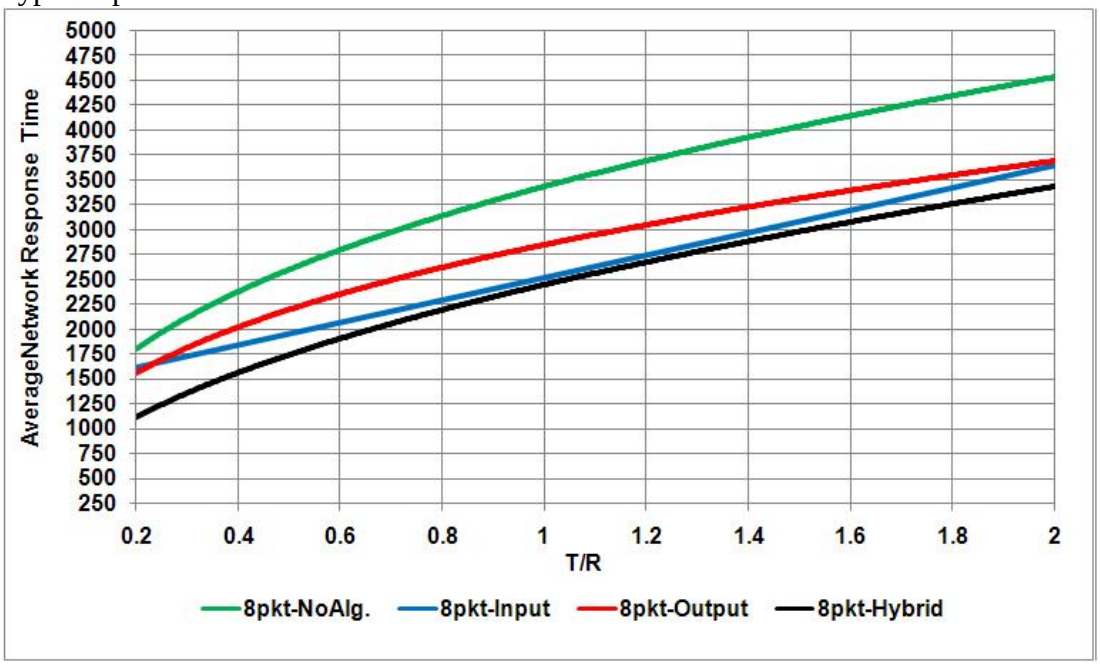

Figure 5.11.b. Average network response time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

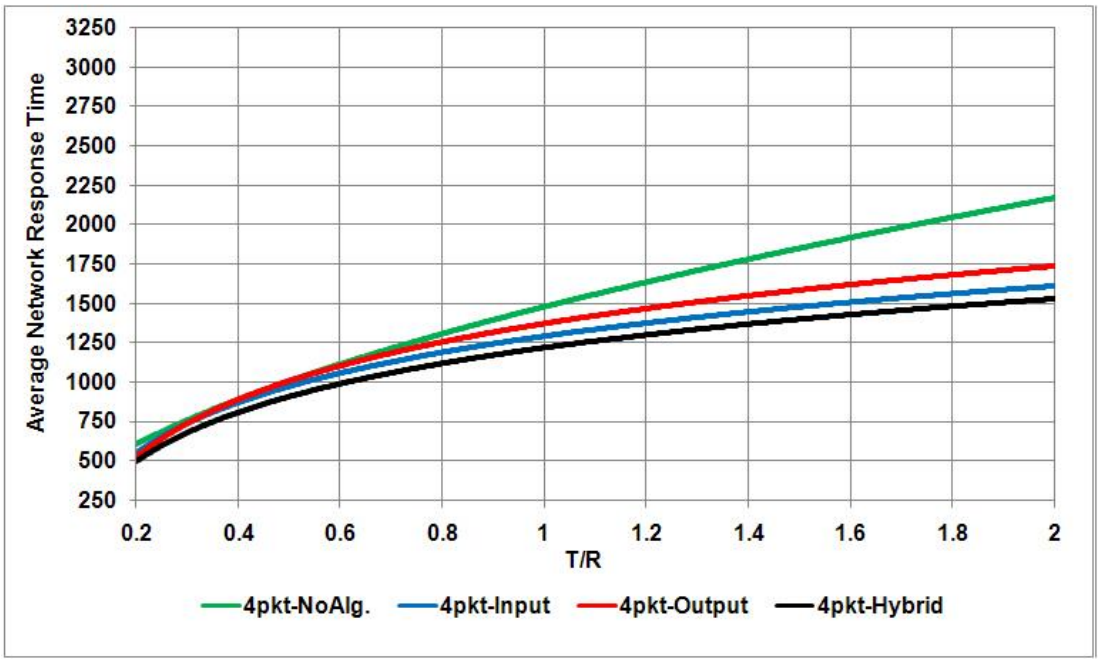

Figure 5.12.a. Average network response time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

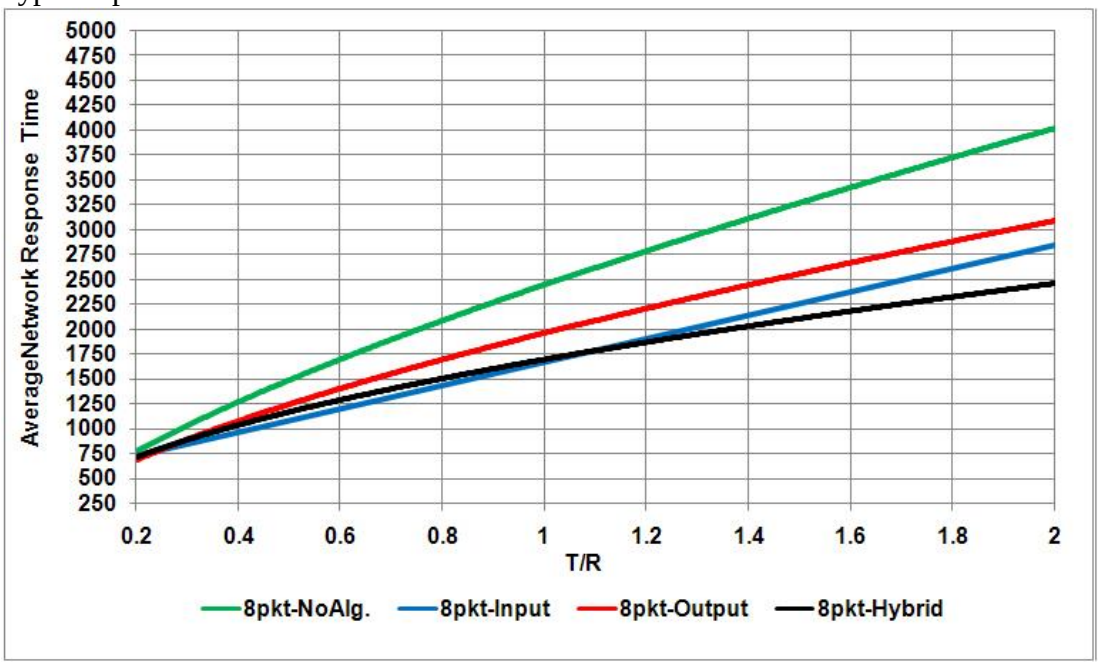

Figure 5.12.b. Average network response time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

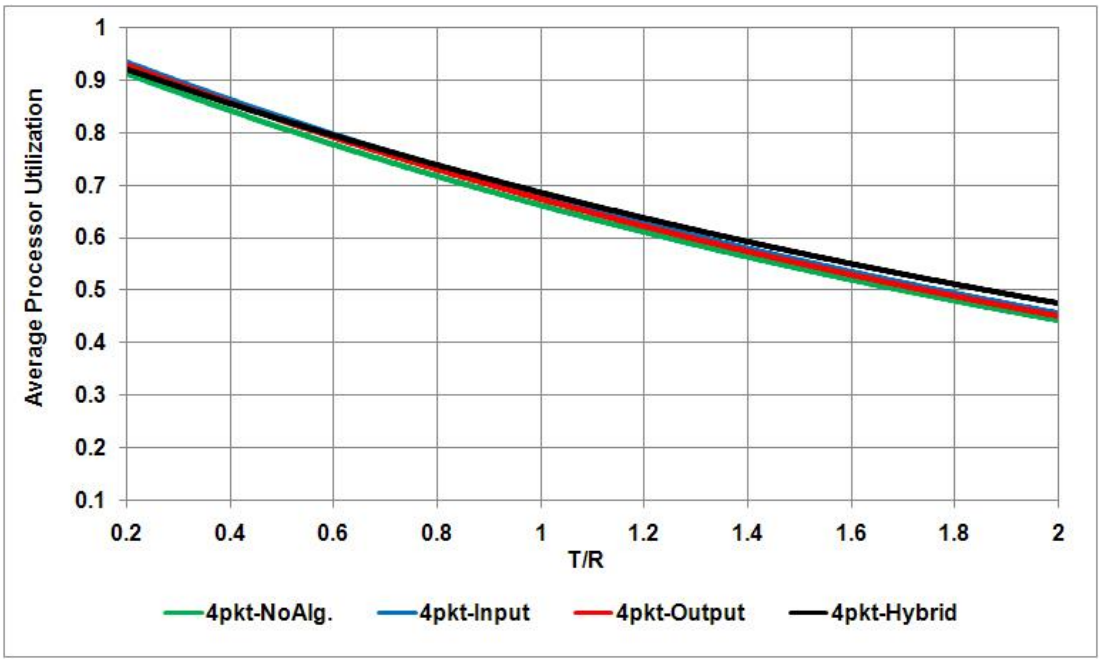

Figure 5.13.a. Average processor utilization for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

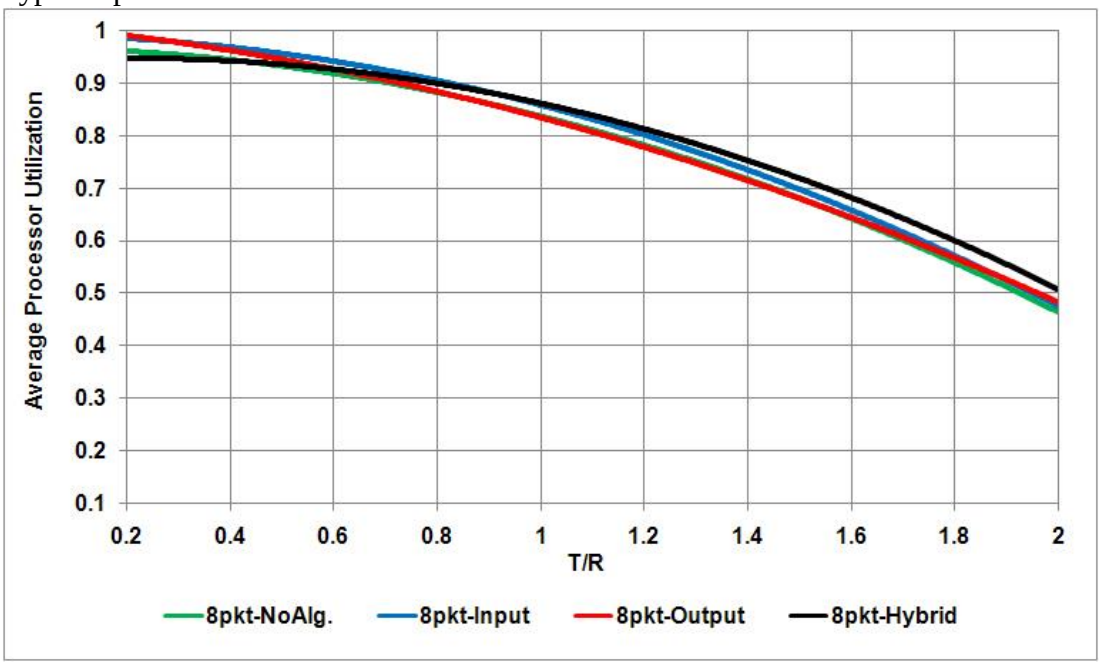

Figure 5.13.b. Average processor utilization for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

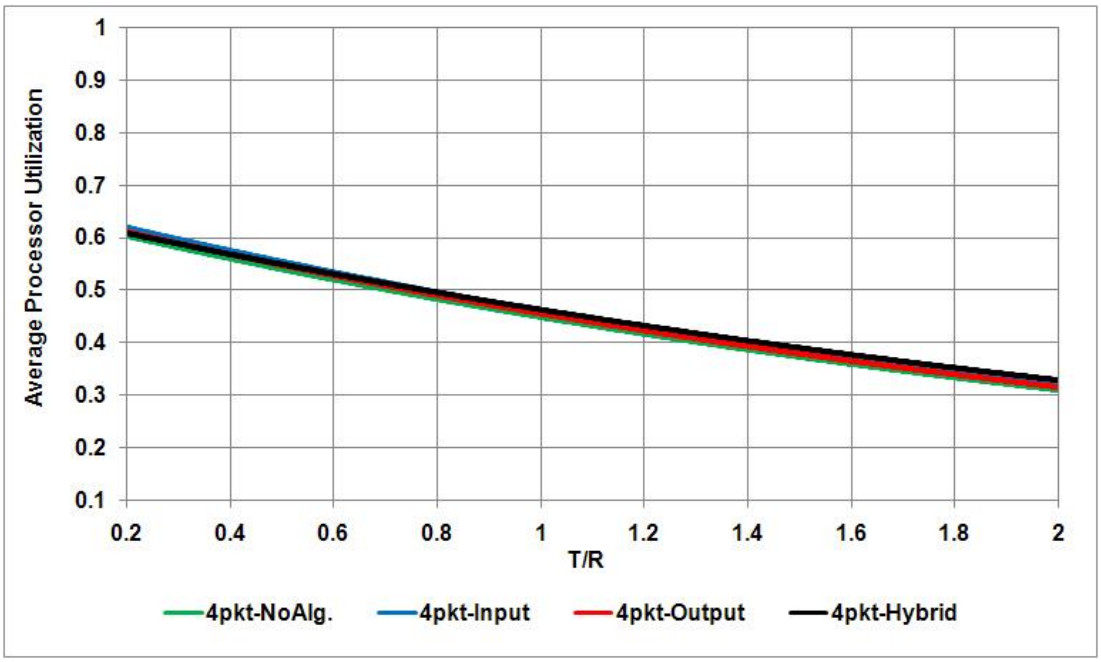

Figure 5.14.a. Average processor utilization for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

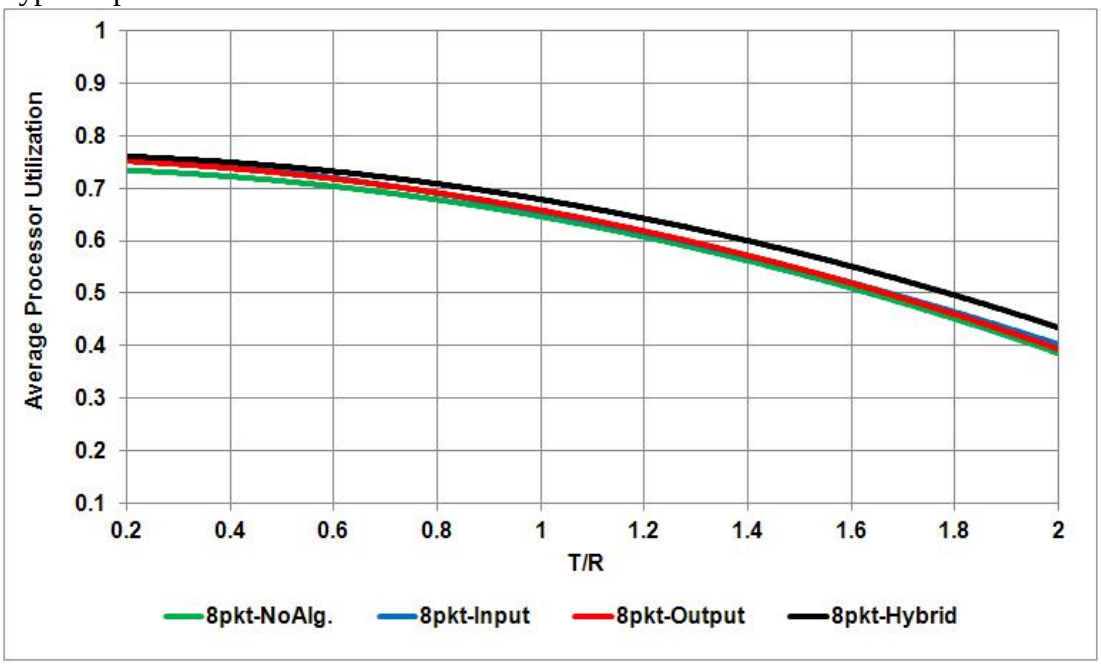

Figure 5.14.b. Average processor utilization for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

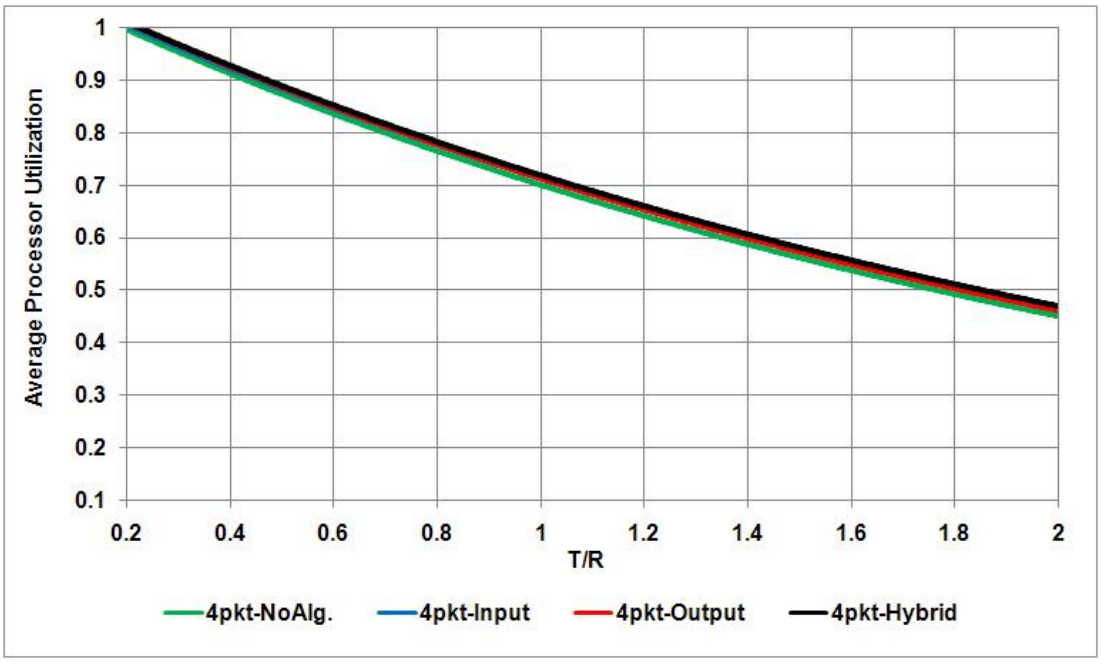

Figure 5.15.a. Average processor utilization for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

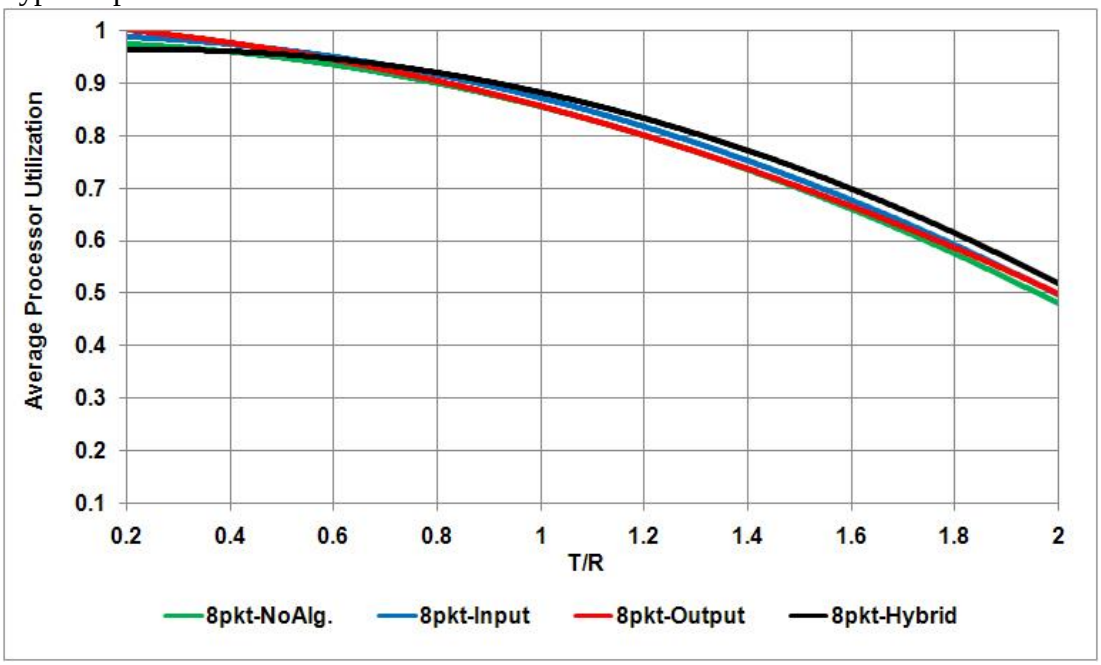

Figure 5.15.b. Average processor utilization for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

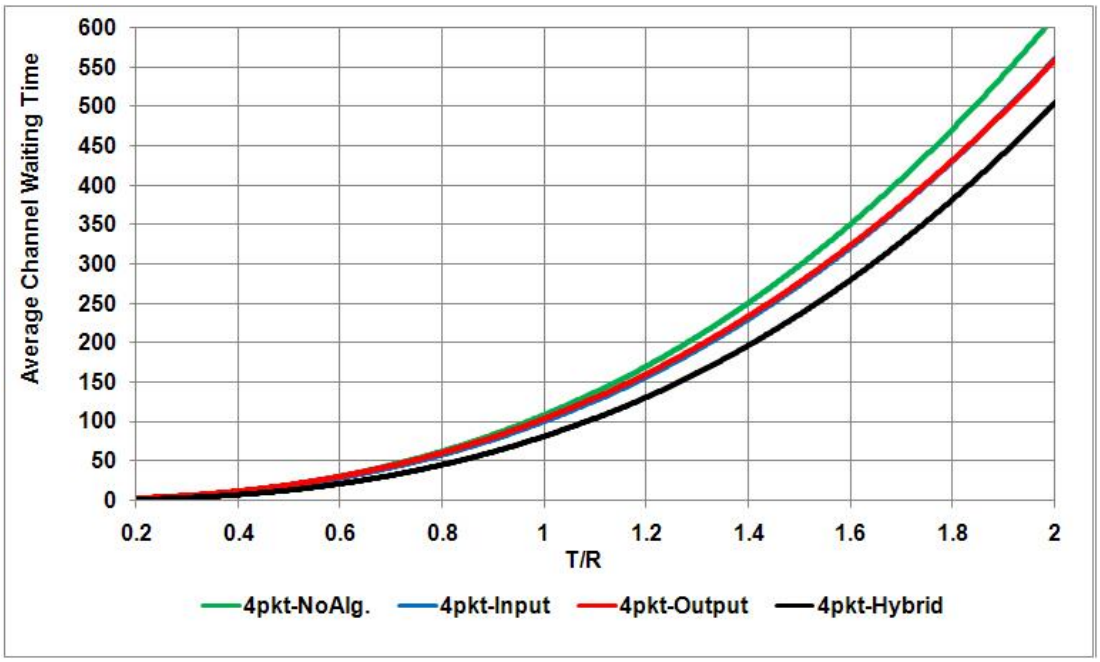

Figure 5.16.a. Average channel waiting time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

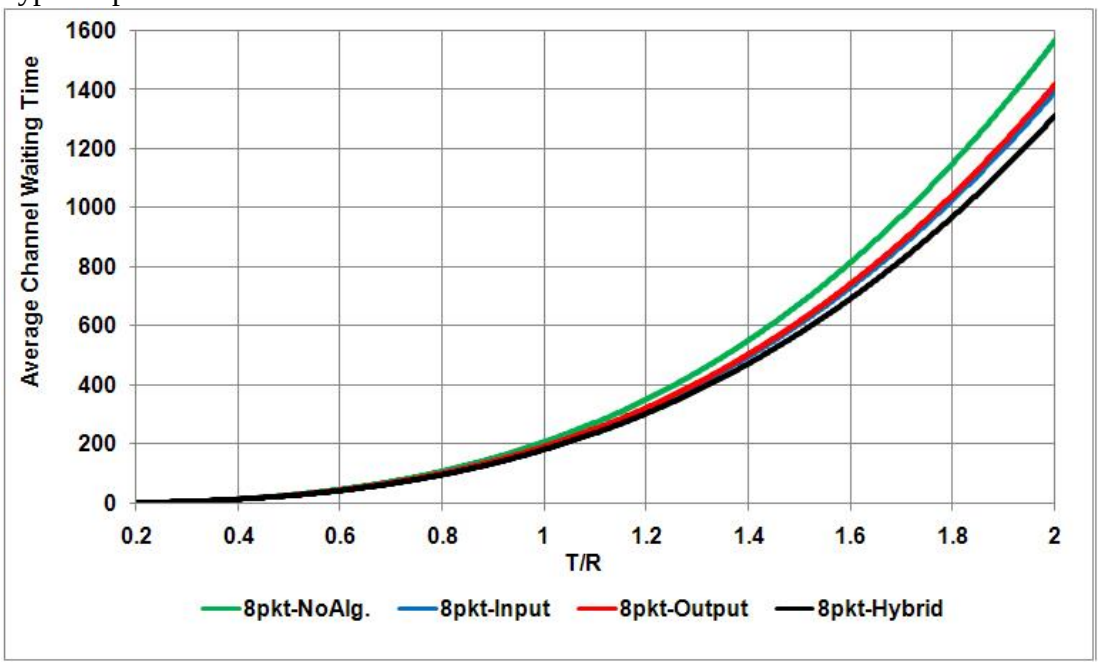

Figure 5.16.b. Average channel waiting time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

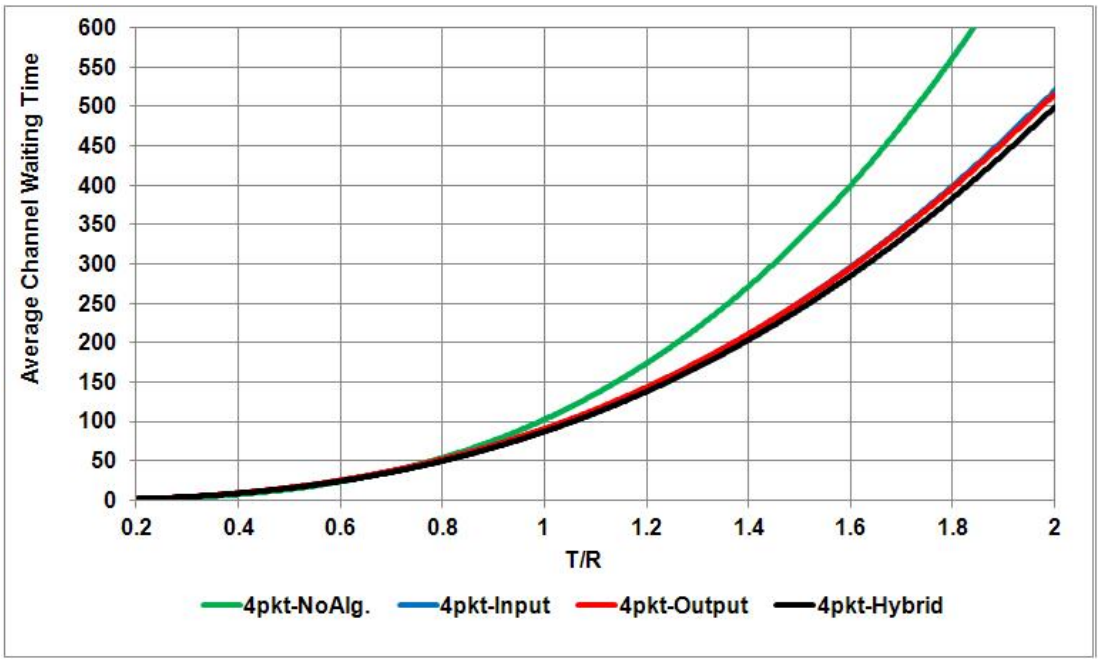

Figure 5.17.a. Average channel waiting time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

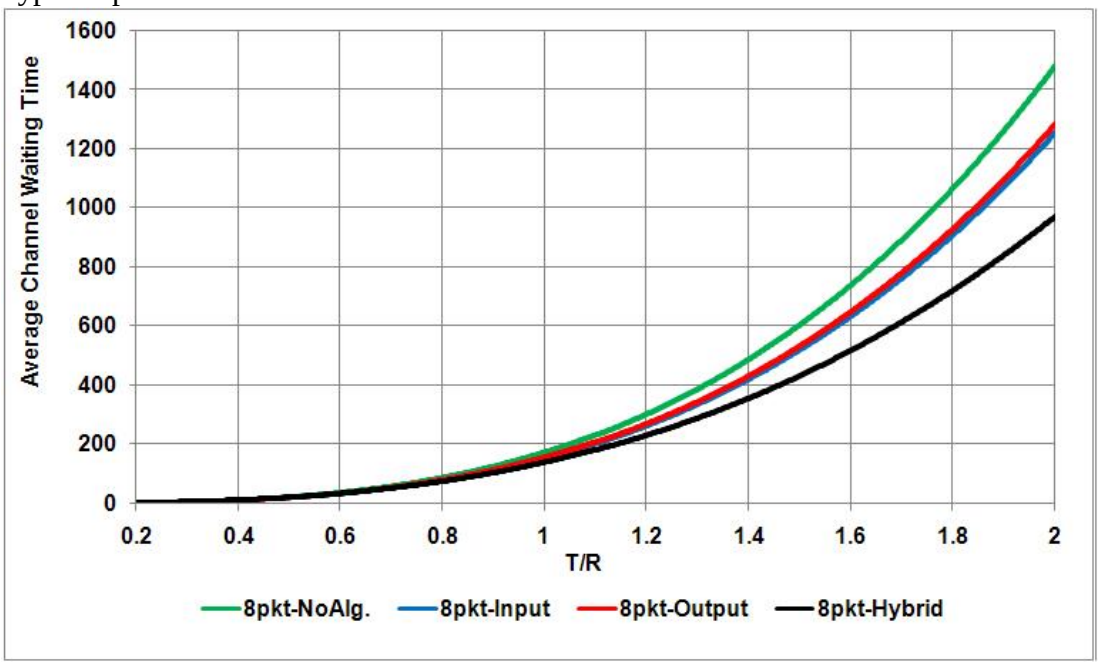

Figure 5.17.b. Average channel waiting time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

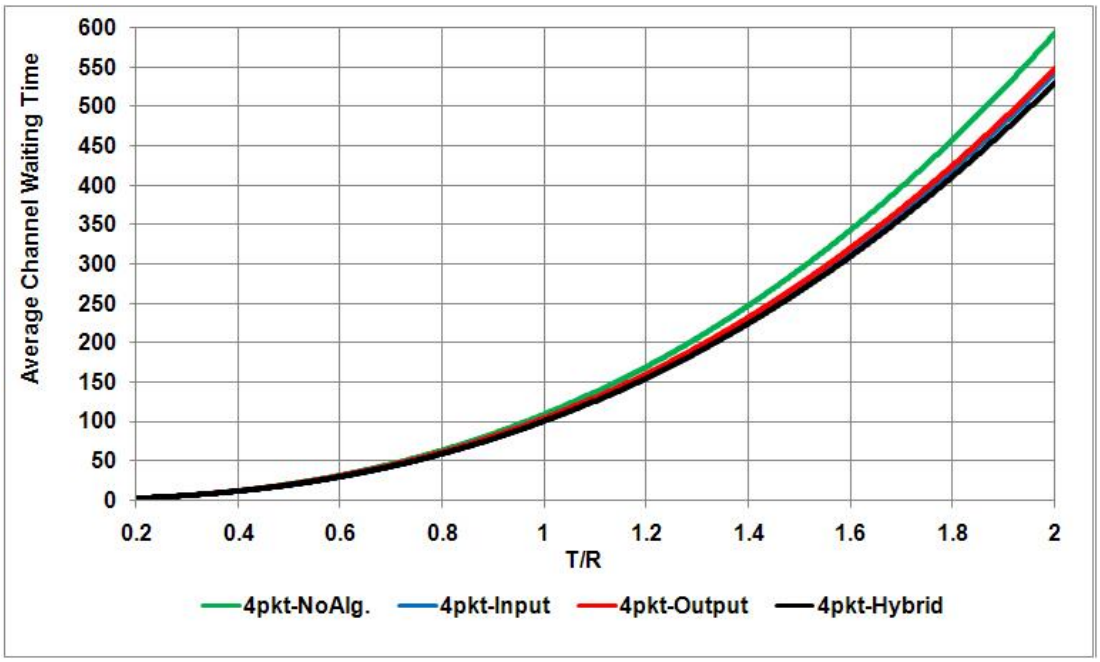

Figure 5.18.a. Average channel waiting time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

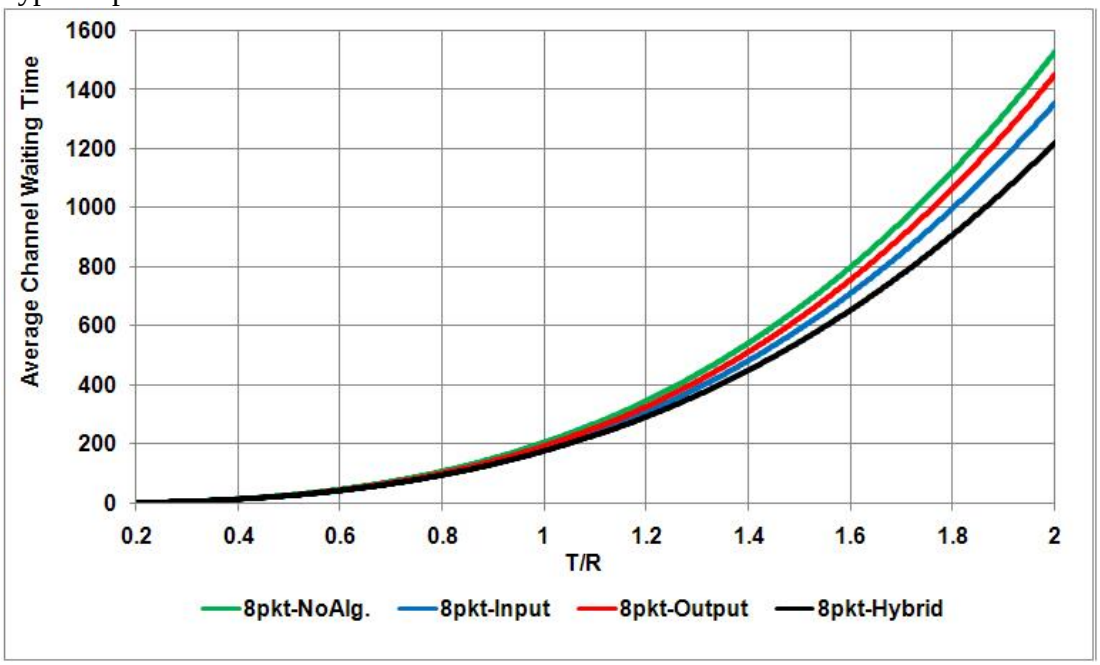

Figure 5.18.b. Average channel waiting time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

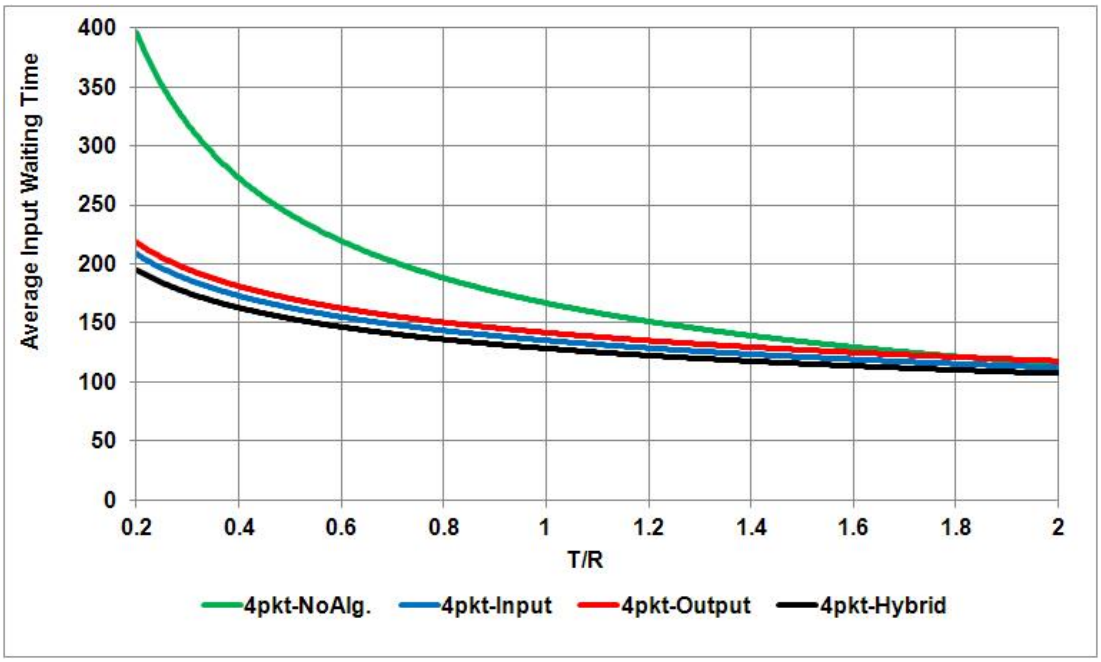

Figure 5.19.a. Average input waiting time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

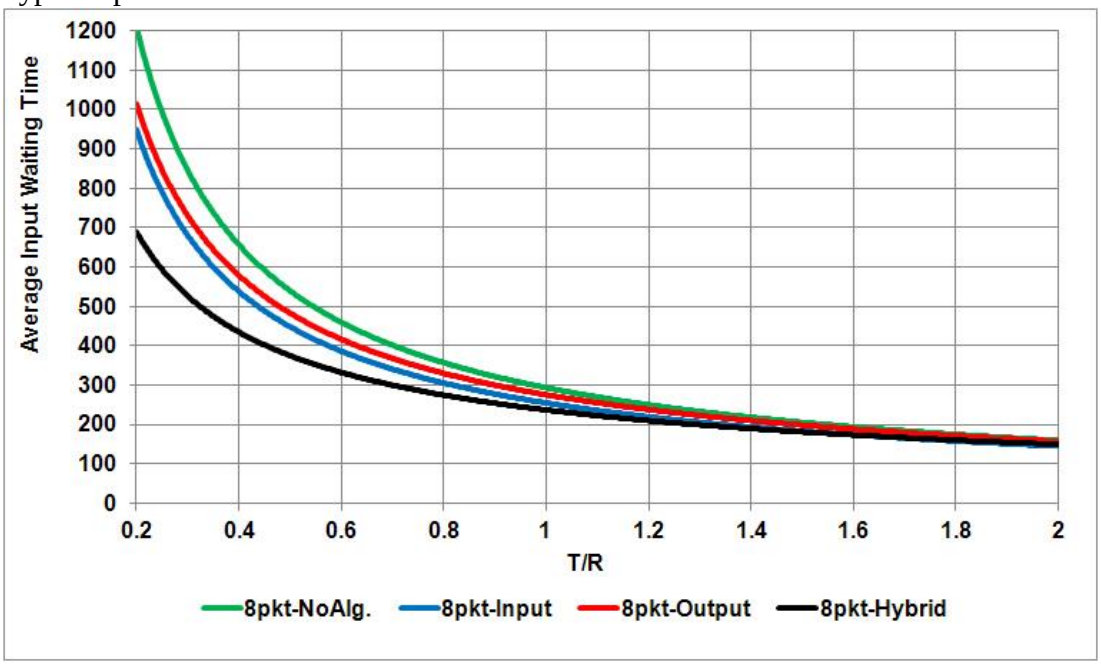

Figure 5.19.b. Average input waiting time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

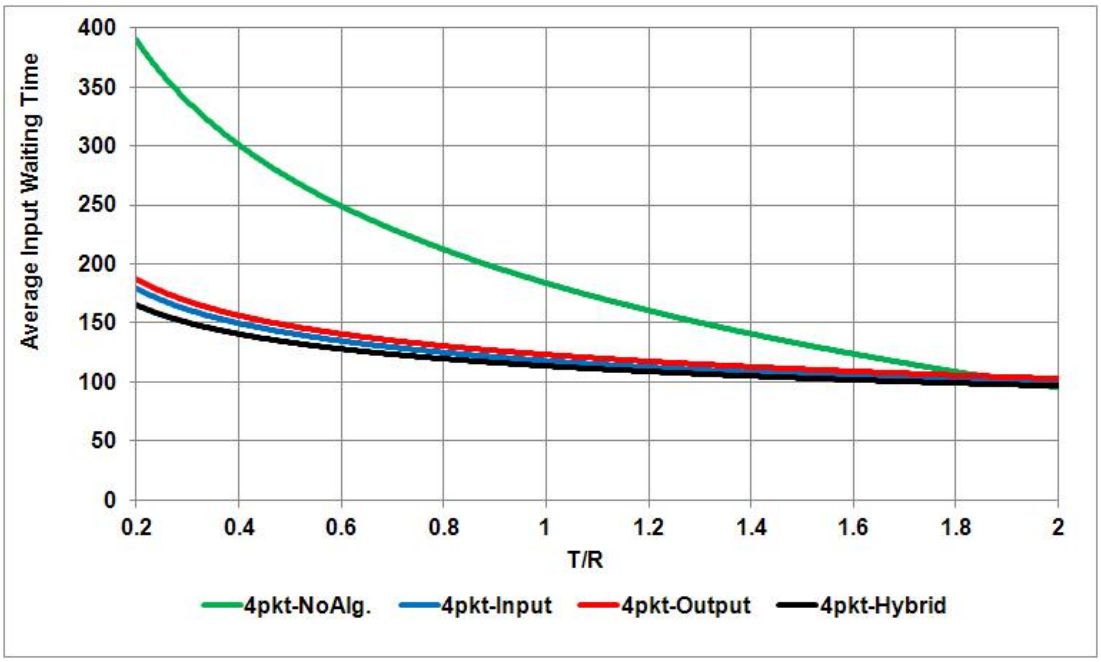

Figure 5.20.a. Average input waiting time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

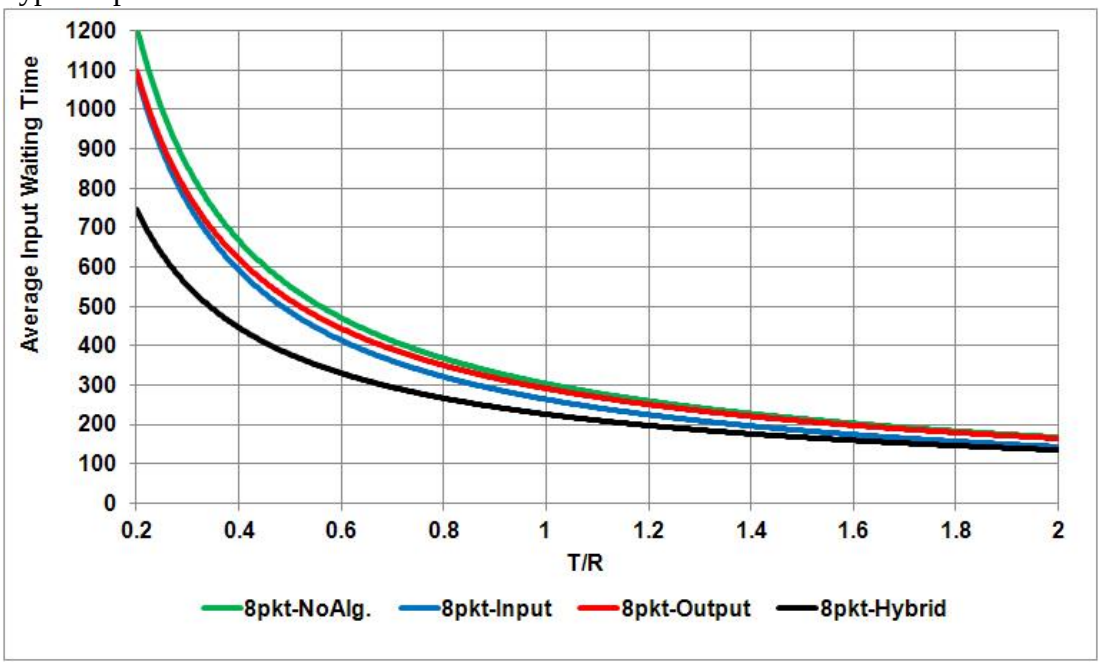

Figure 5.20.b. Average input waiting time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

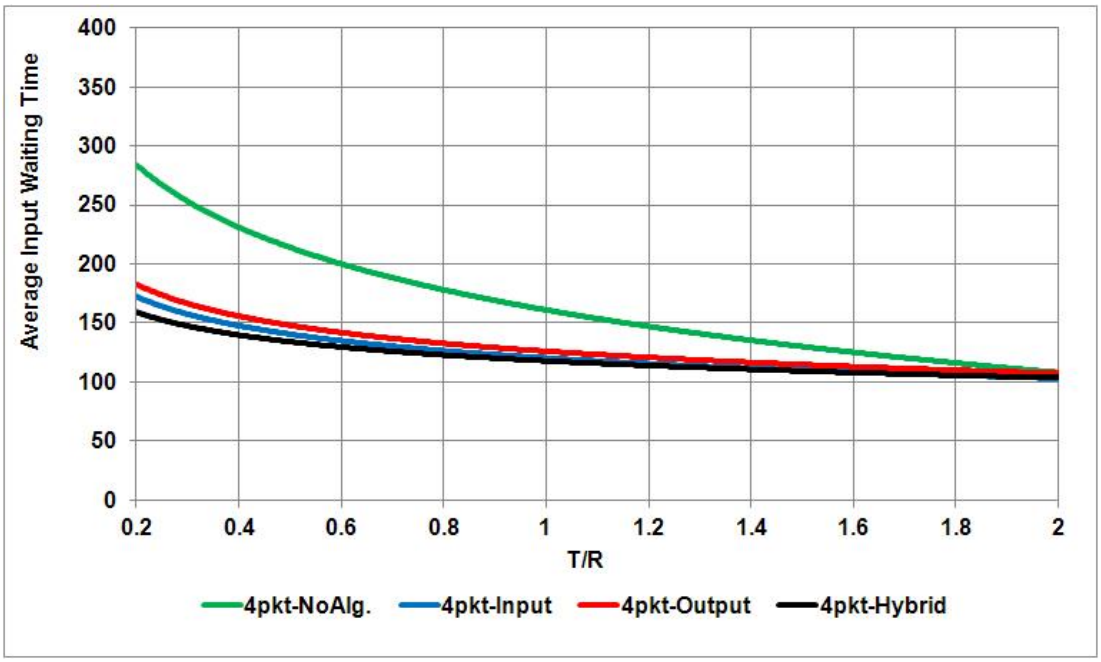

Figure 5.21.a. Average input waiting time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

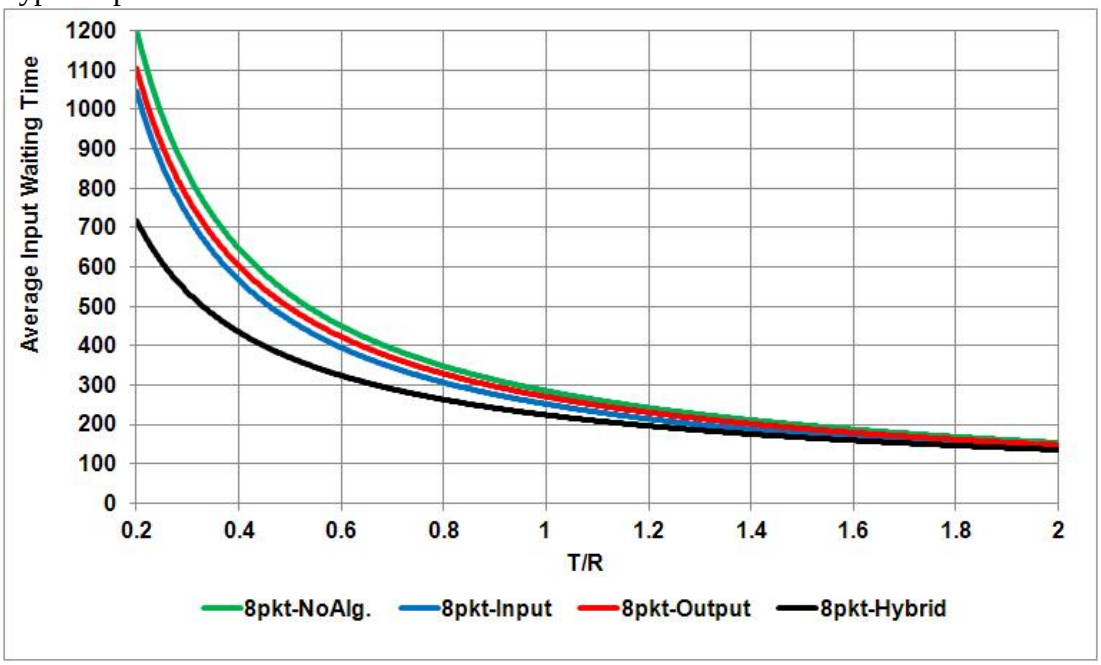

Figure 5.21.b. Average input waiting time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

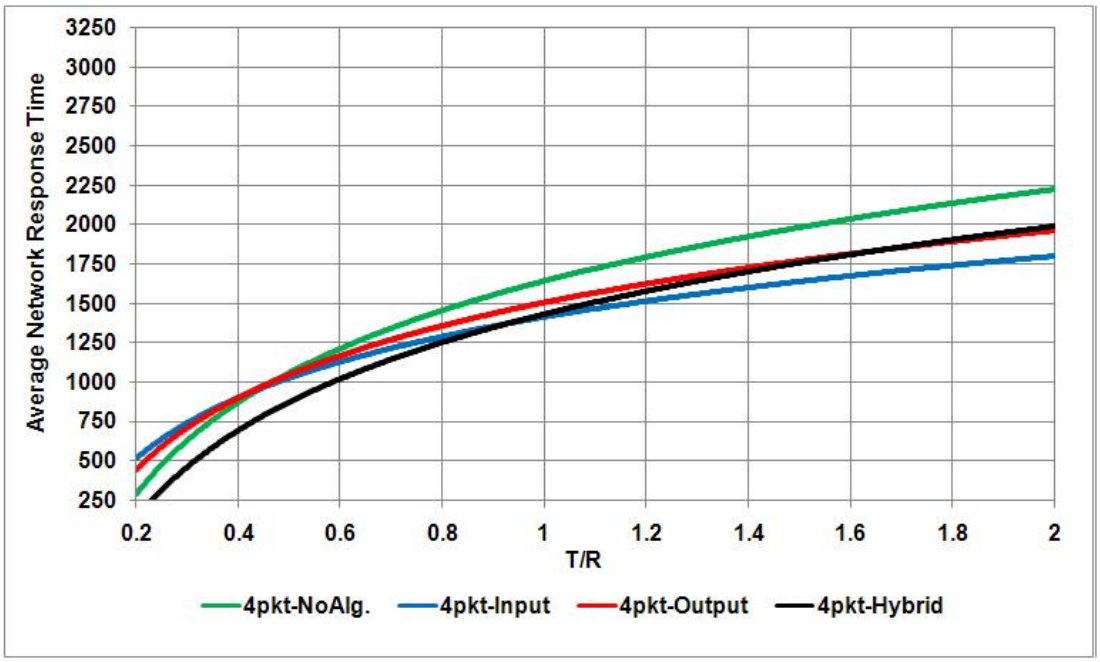

Figure 5.22.a. Average network response time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

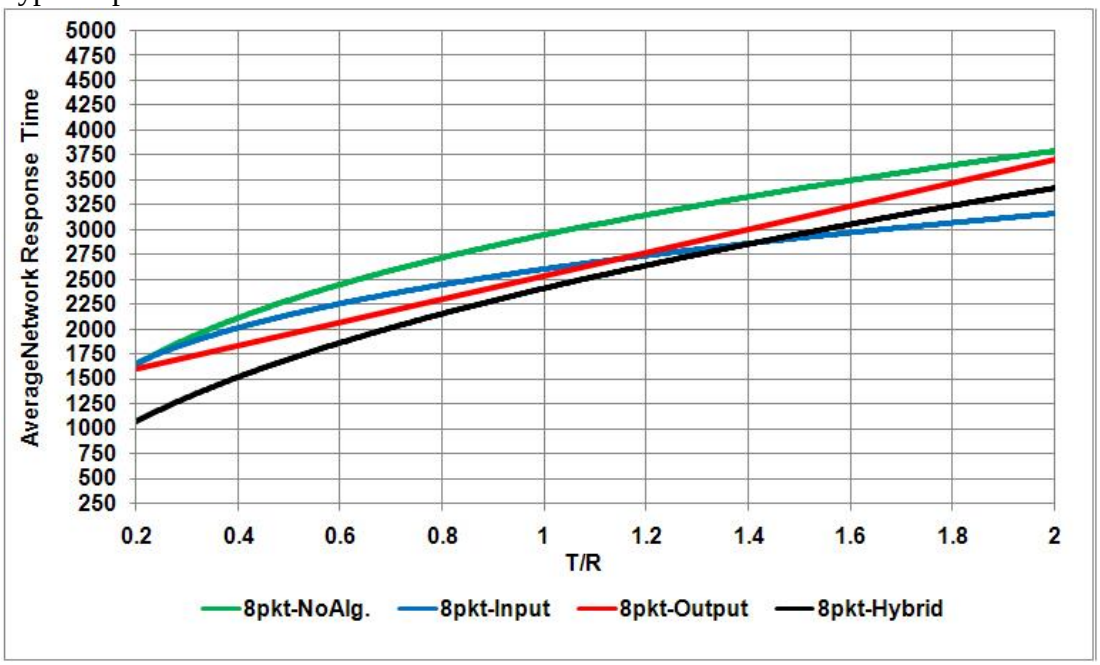

Figure 5.22.b. Average network response time for Client-Server traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

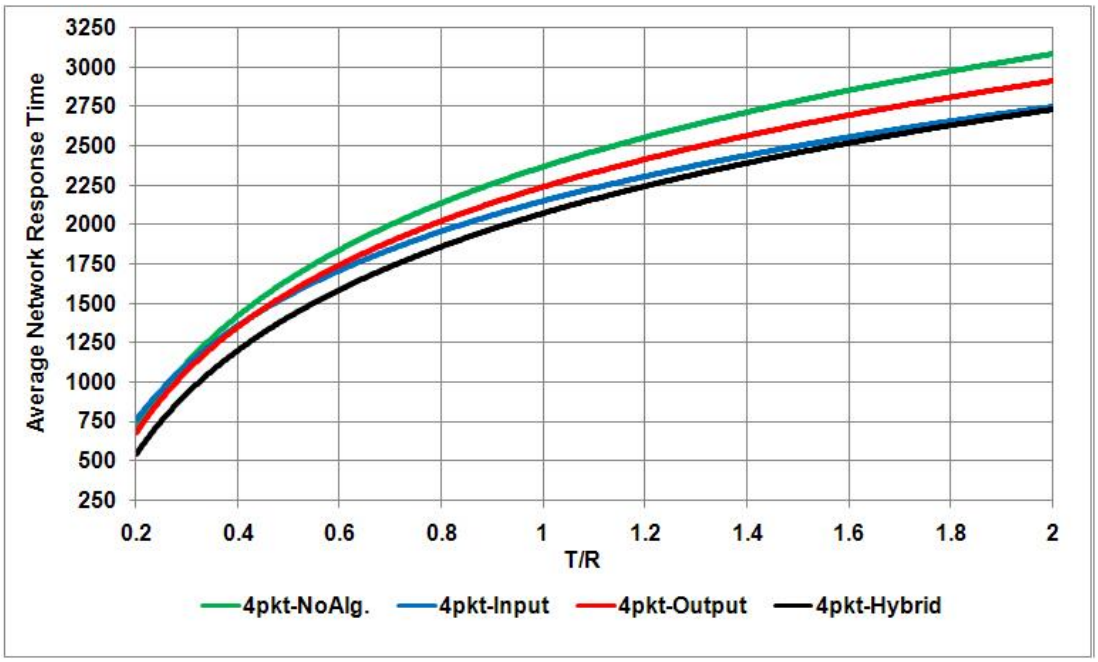

Figure 5.23.a. Average network response time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

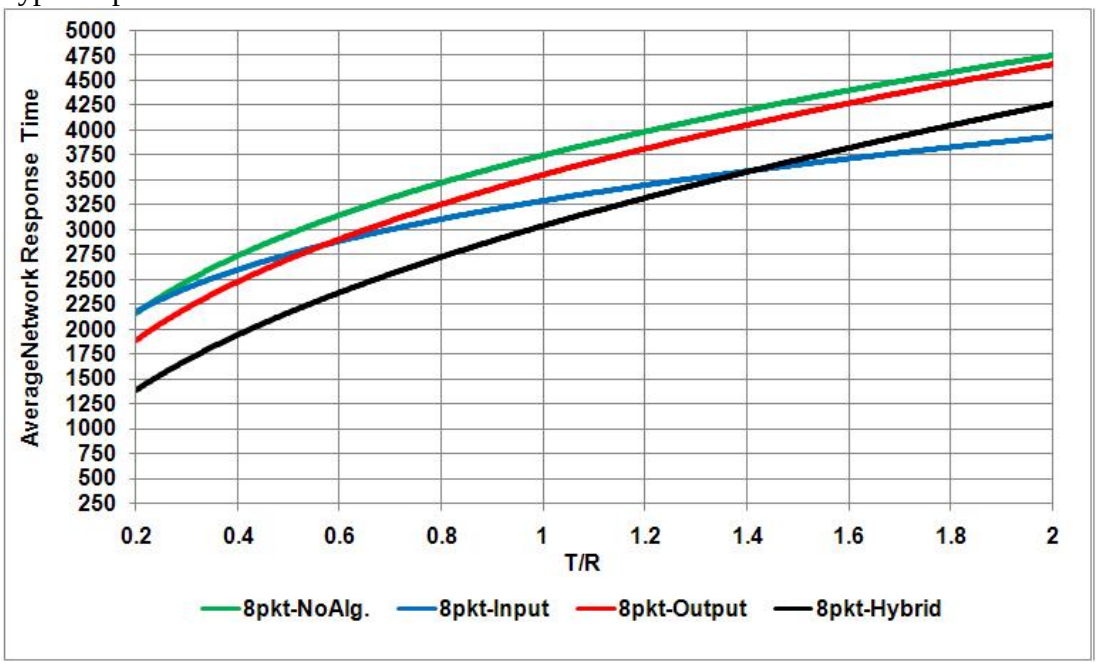

Figure 5.23.b. Average network response time for Client-Server traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

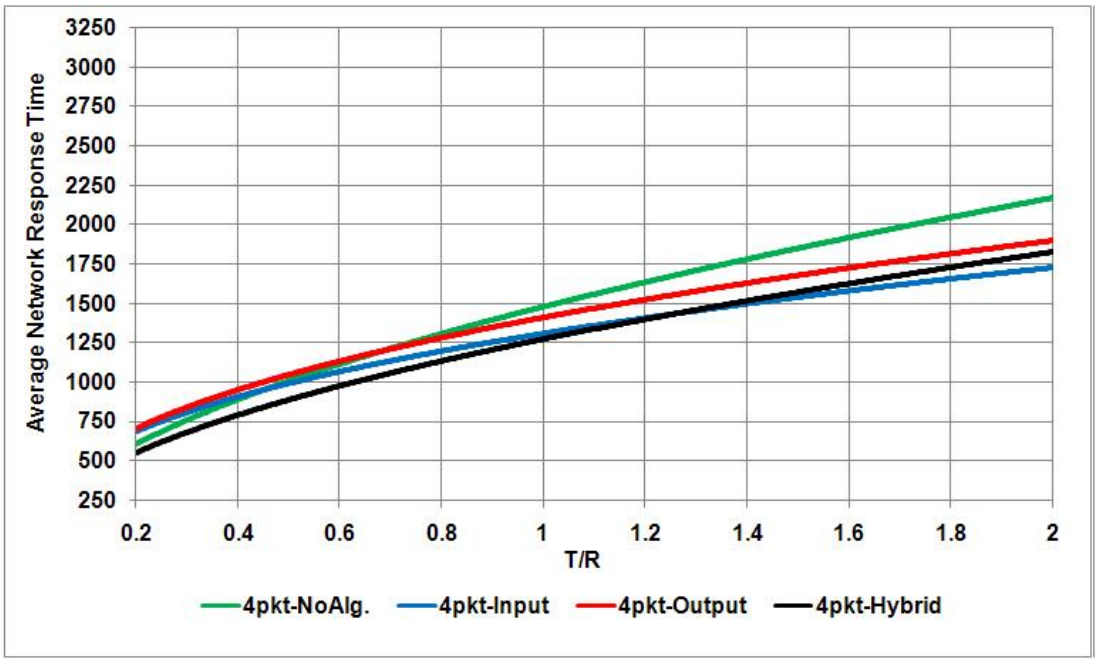

Figure 5.24.a. Average network response time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

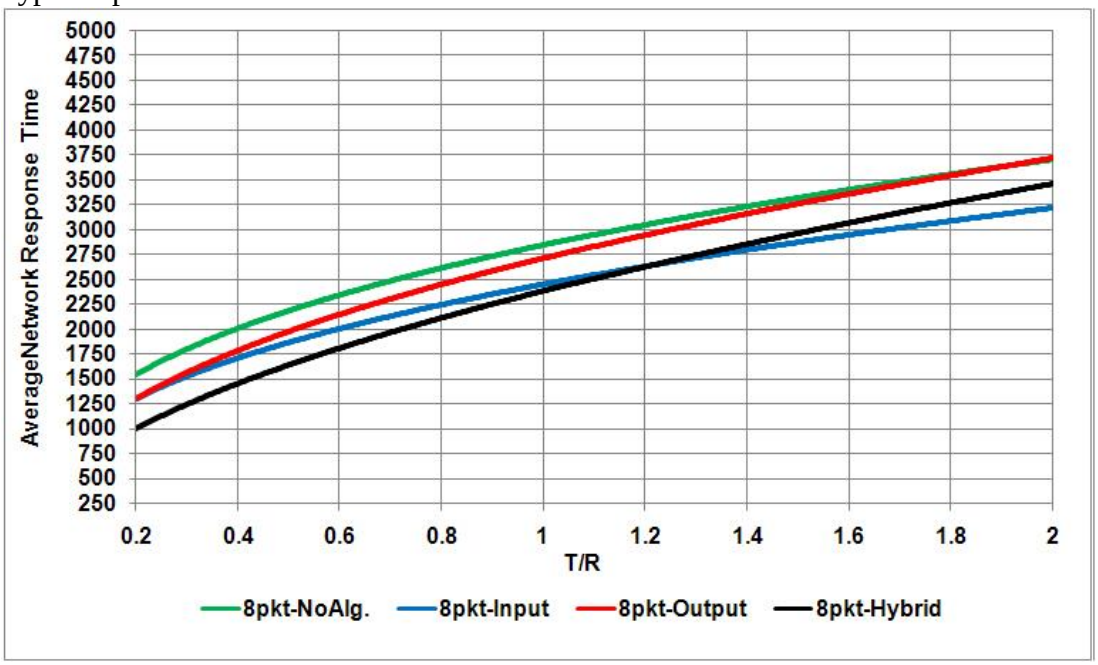

Figure 5.24.b. Average network response time for Client-Server traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

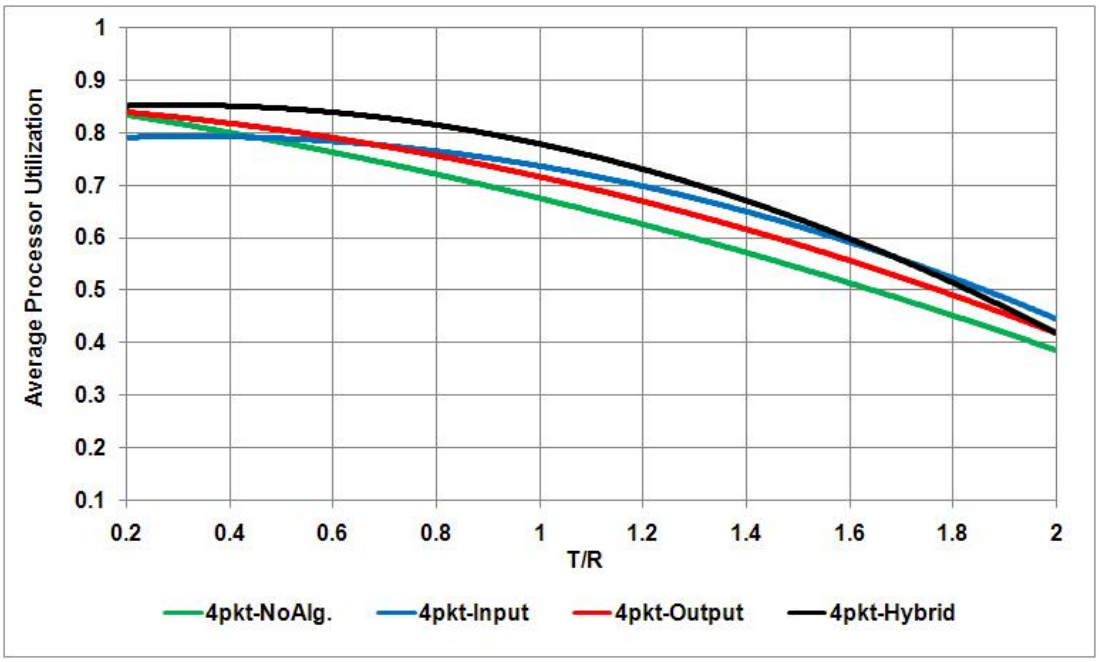

Figure 5.25.a. Average processor utilization for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

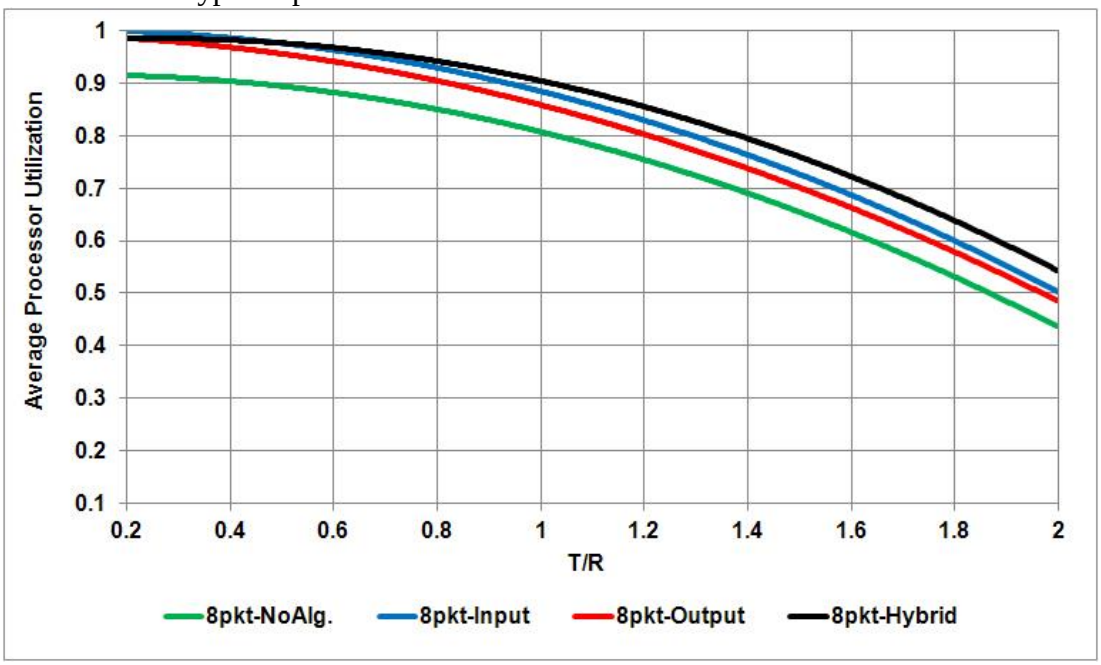

Figure 5.25.b. Average processor utilization for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

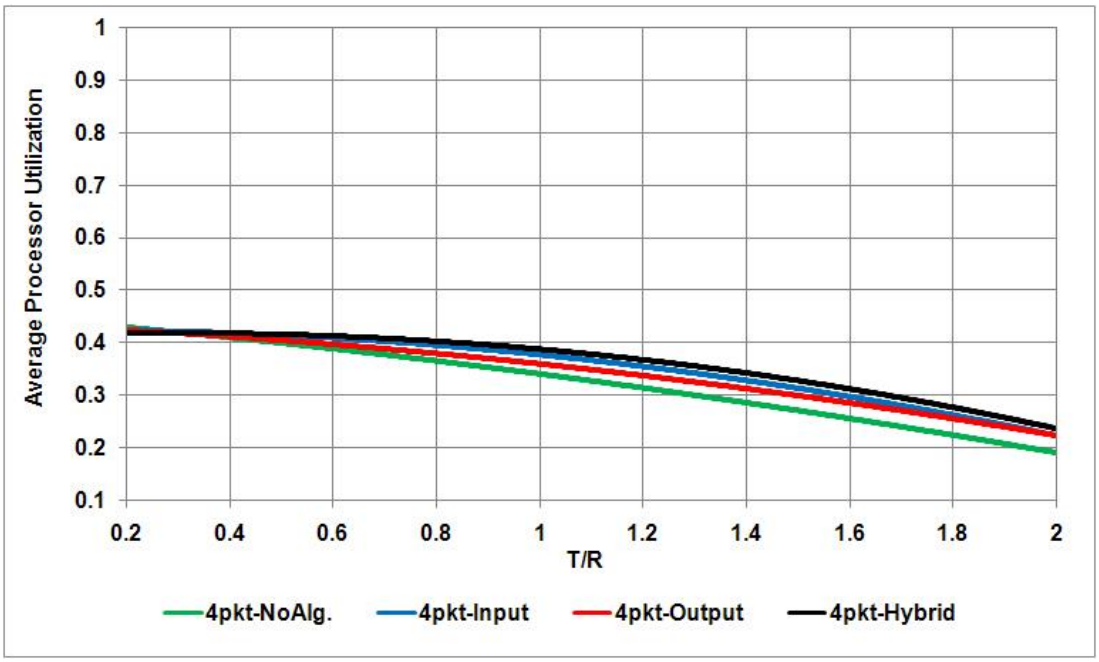

Figure 5.26.a. Average processor utilization for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

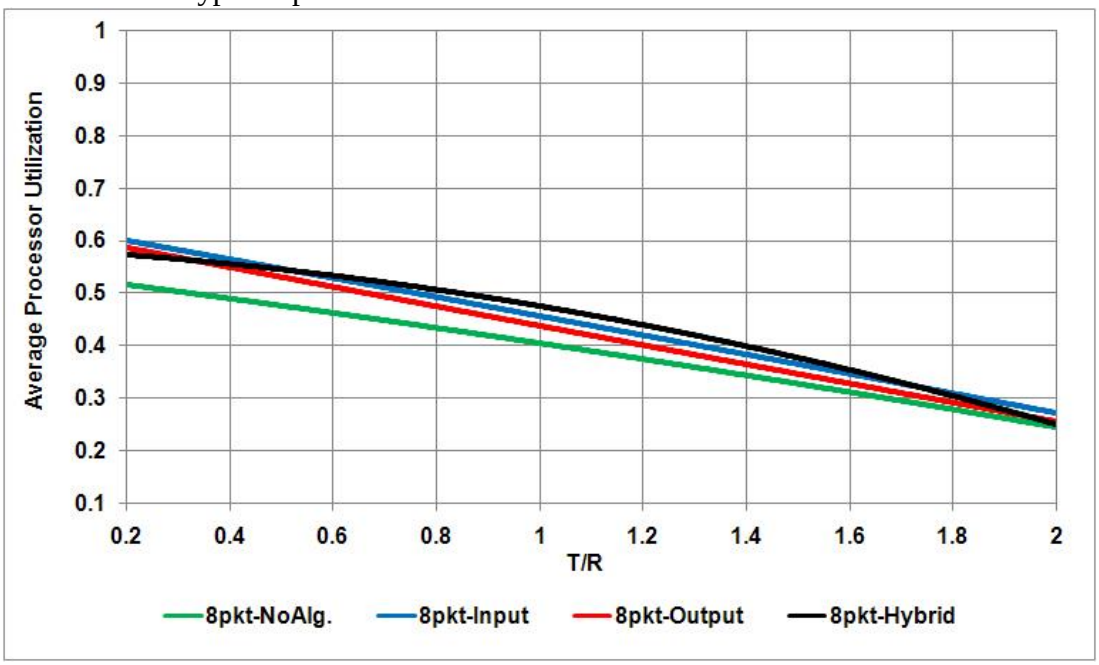

Figure 5.26.b. Average processor utilization for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

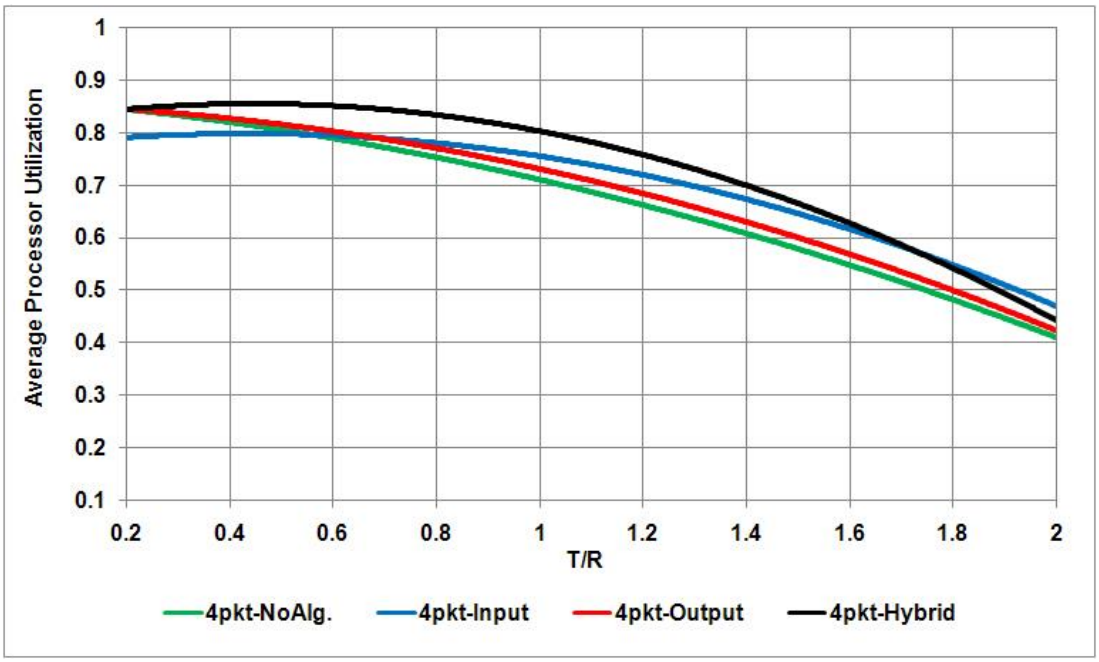

Figure 5.27.a. Average processor utilization for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

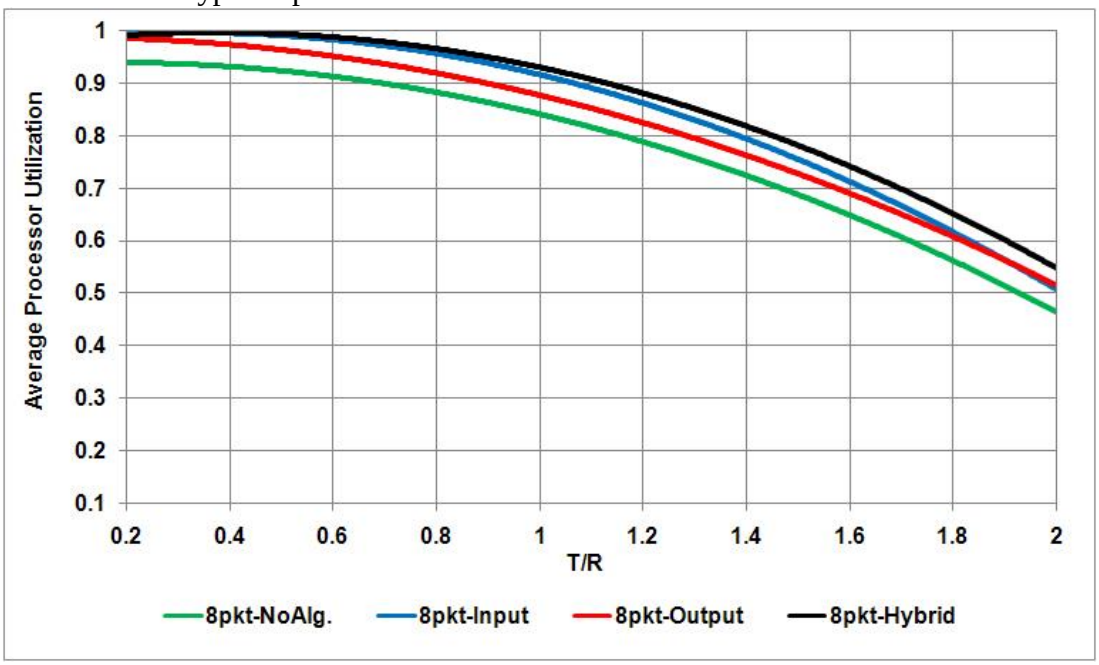

Figure 5.27.b. Average processor utilization for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

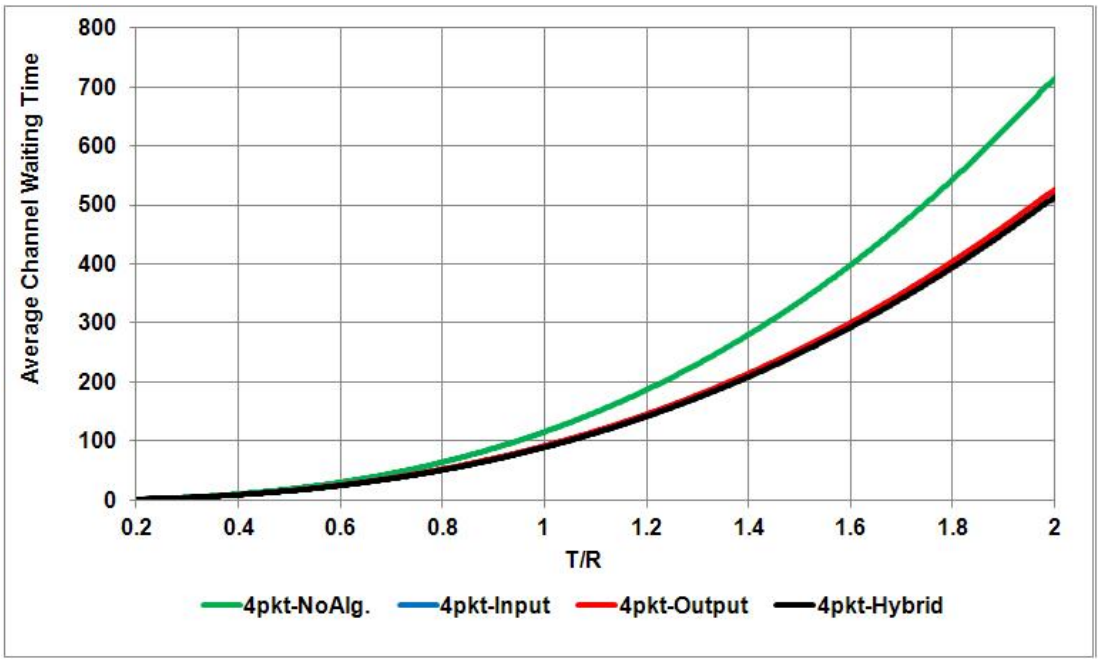

Figure 5.28.a. Average channel waiting time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

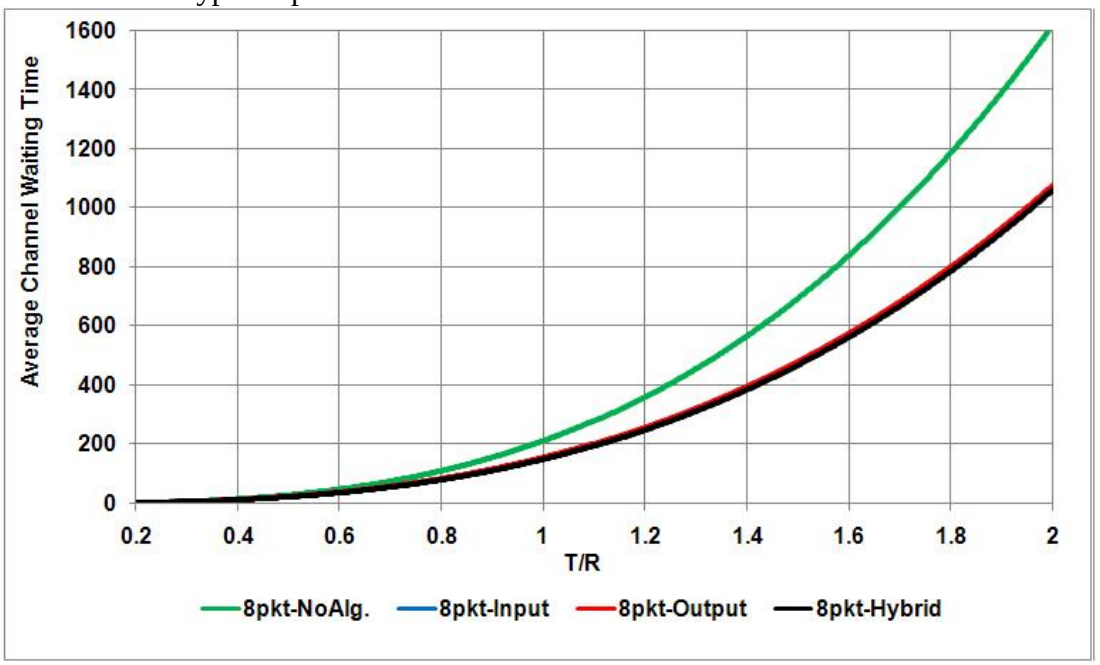

Figure 5.28.b. Average channel waiting time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

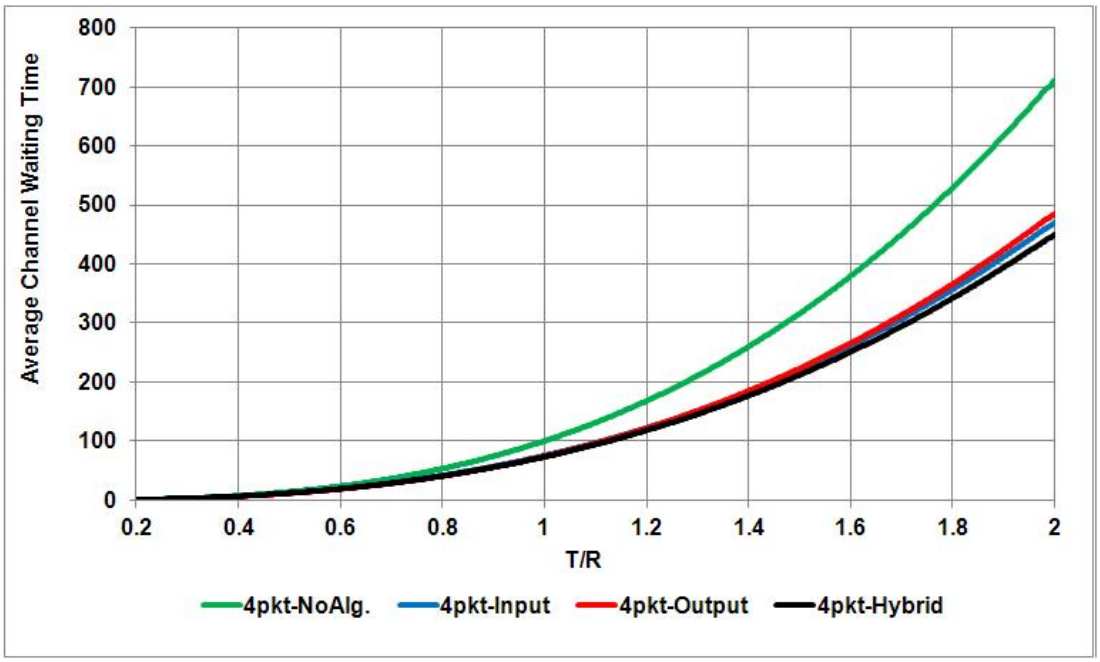

Figure 5.29.a. Average channel waiting time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

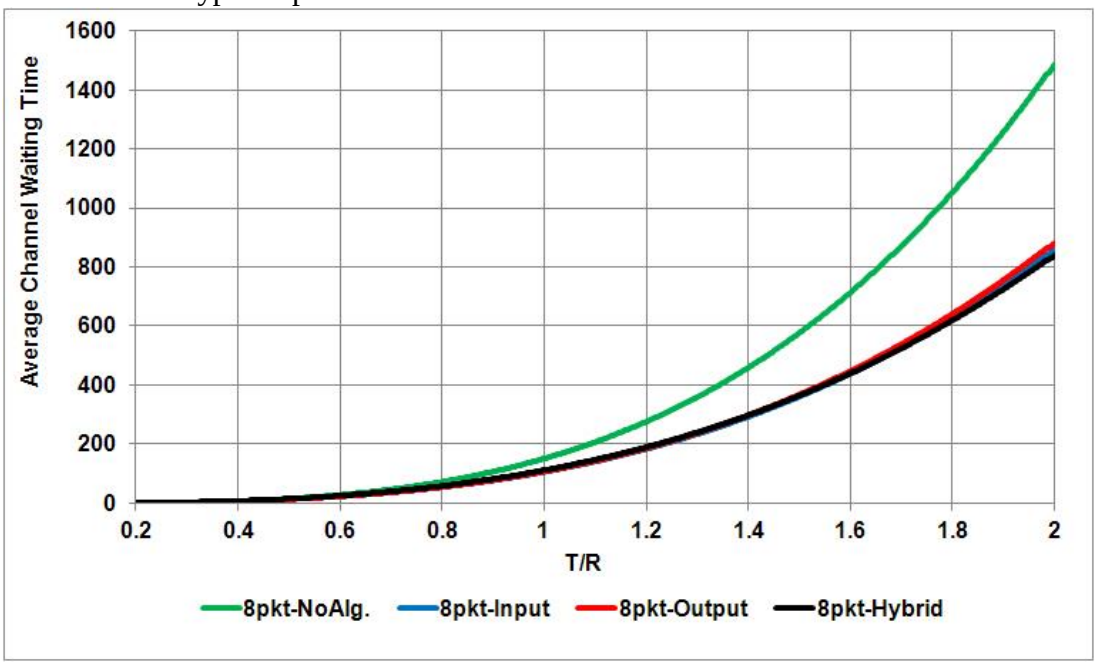

Figure 5.29.b. Average channel waiting time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

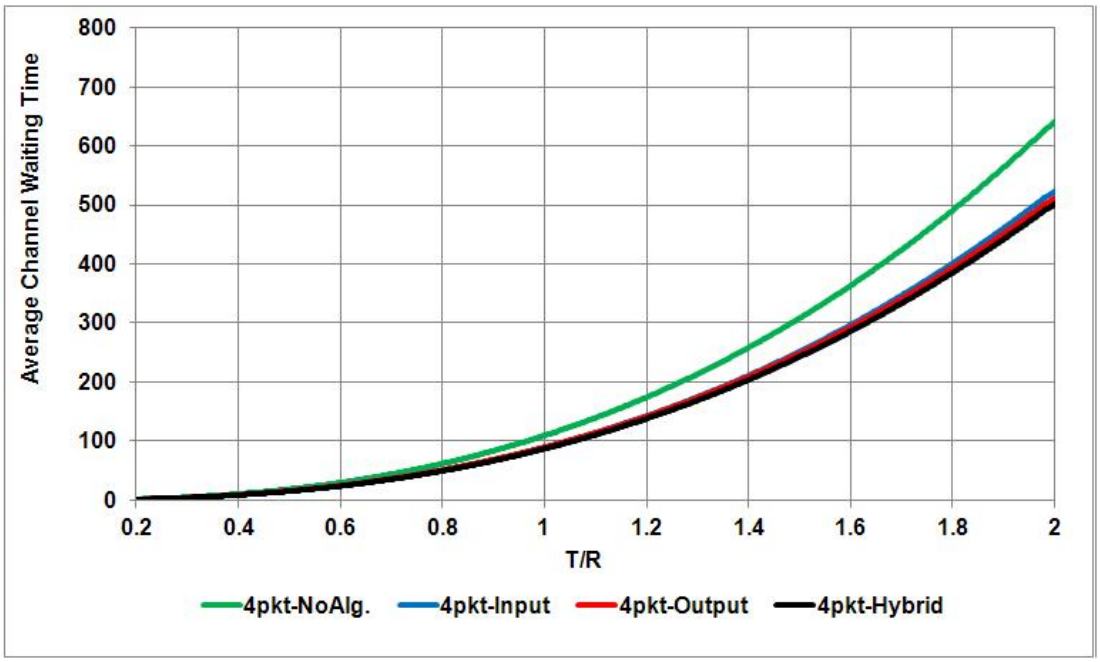

Figure 5.30.a. Average channel waiting time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

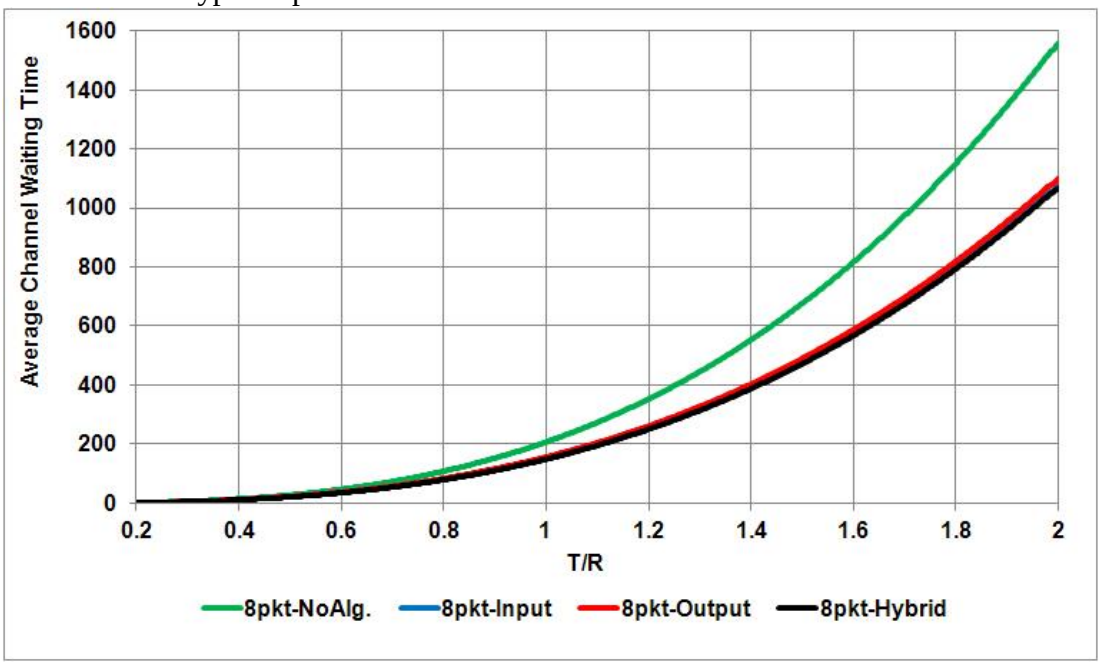

Figure 5.30.b. Average channel waiting time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

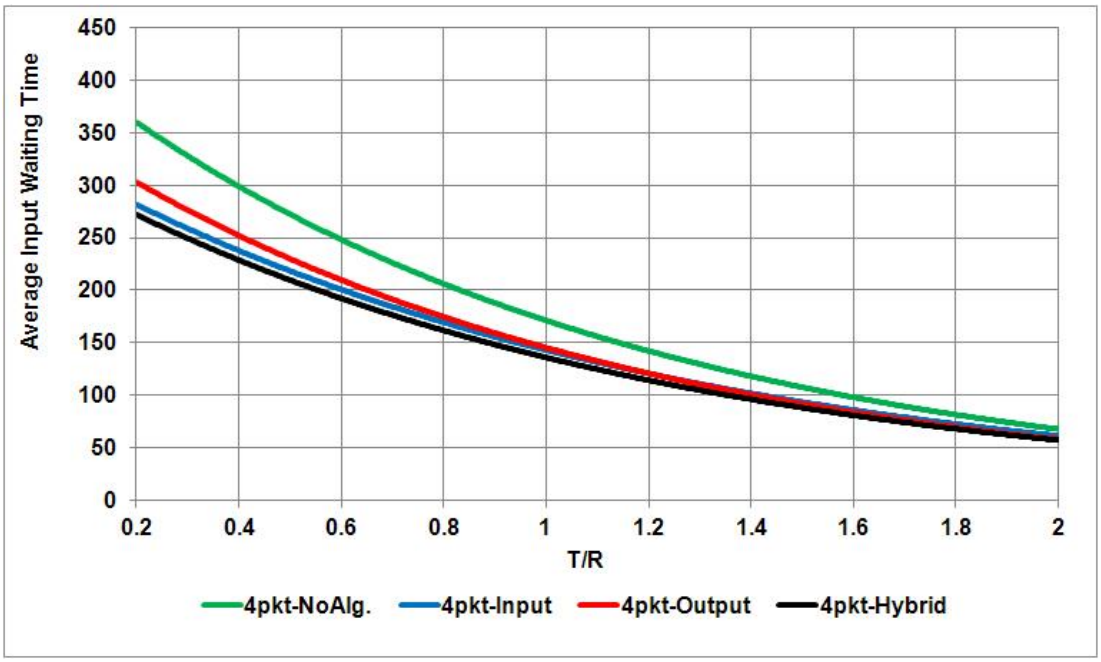

Figure 5.31.a. Average input waiting time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

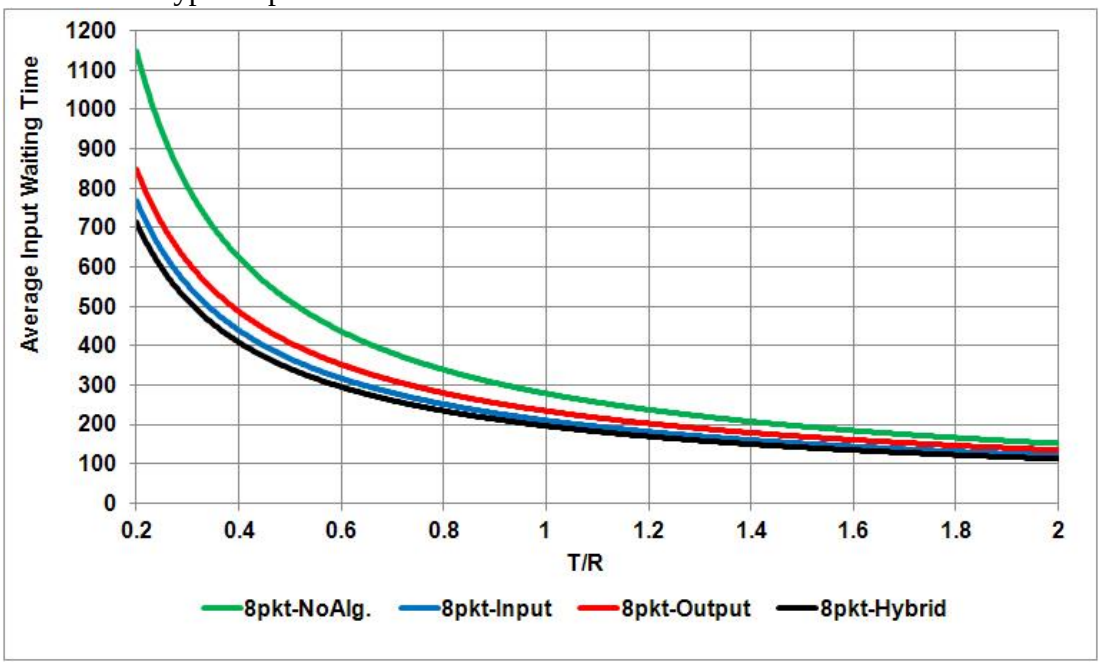

Figure 5.31.b. Average input waiting time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

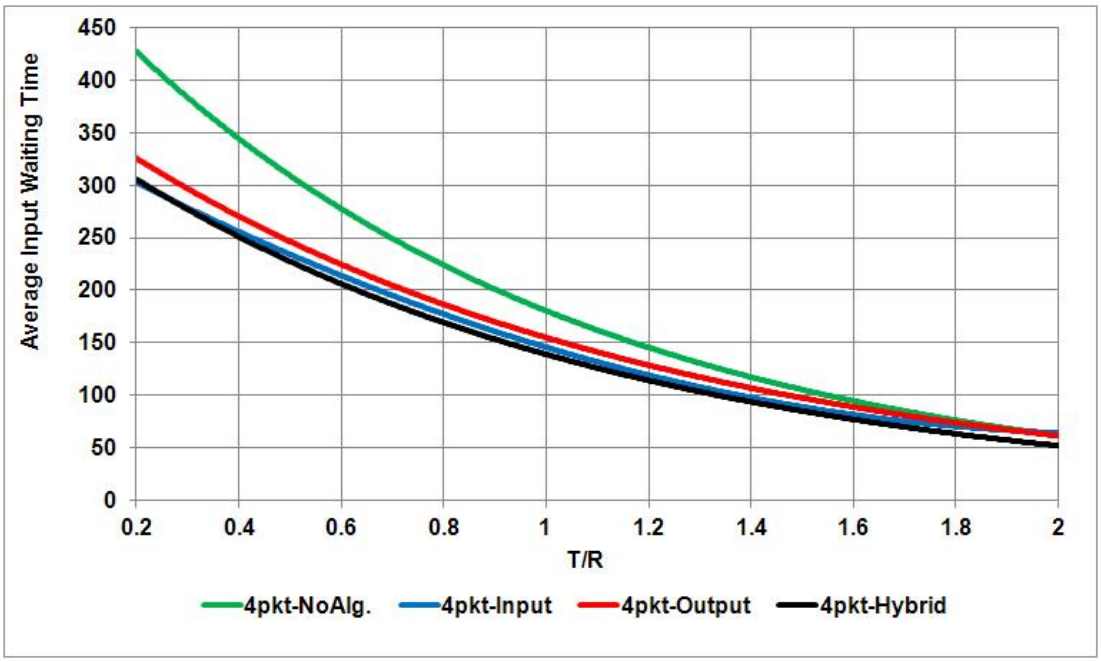

Figure 5.32.a. Average input waiting time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

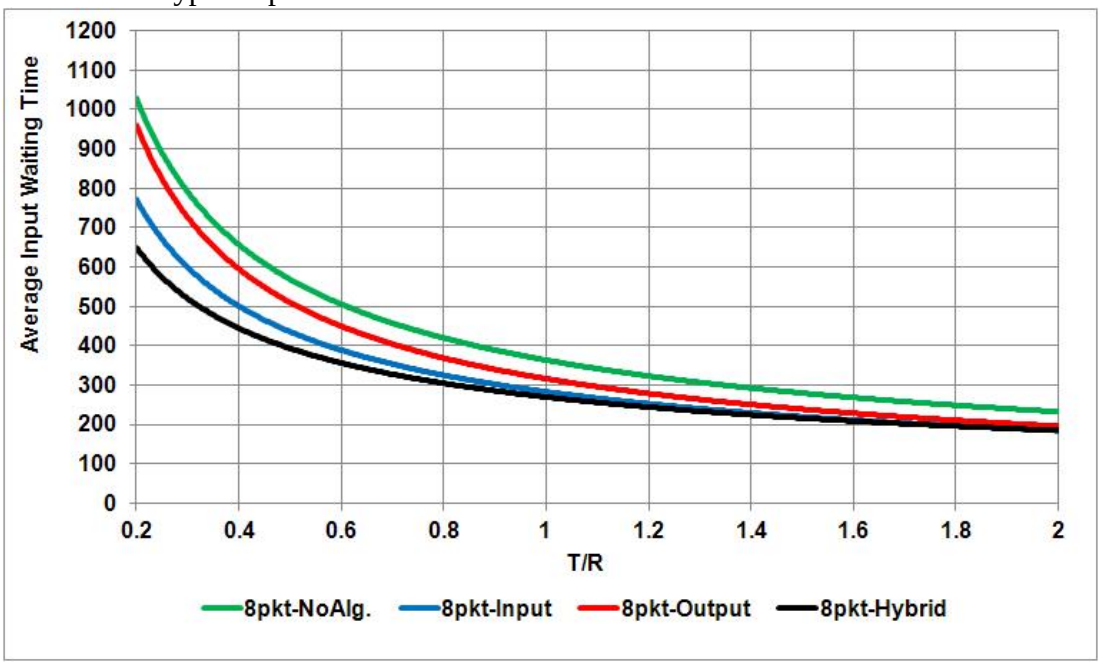

Figure 5.32.b. Average input waiting time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

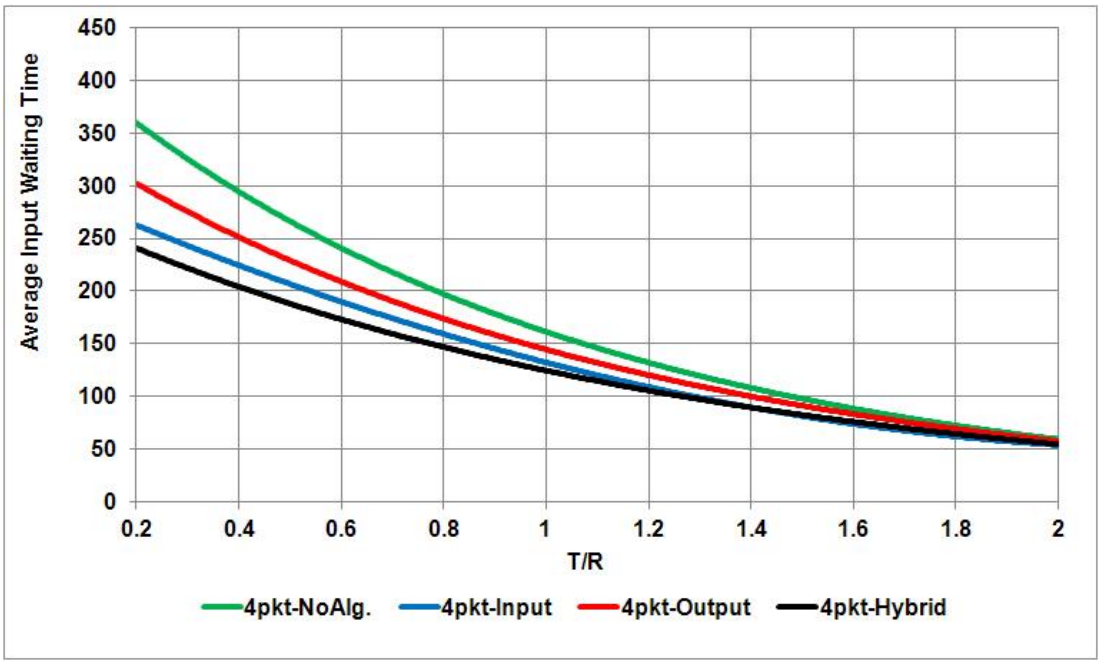

Figure 5.33.a. Average input waiting time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

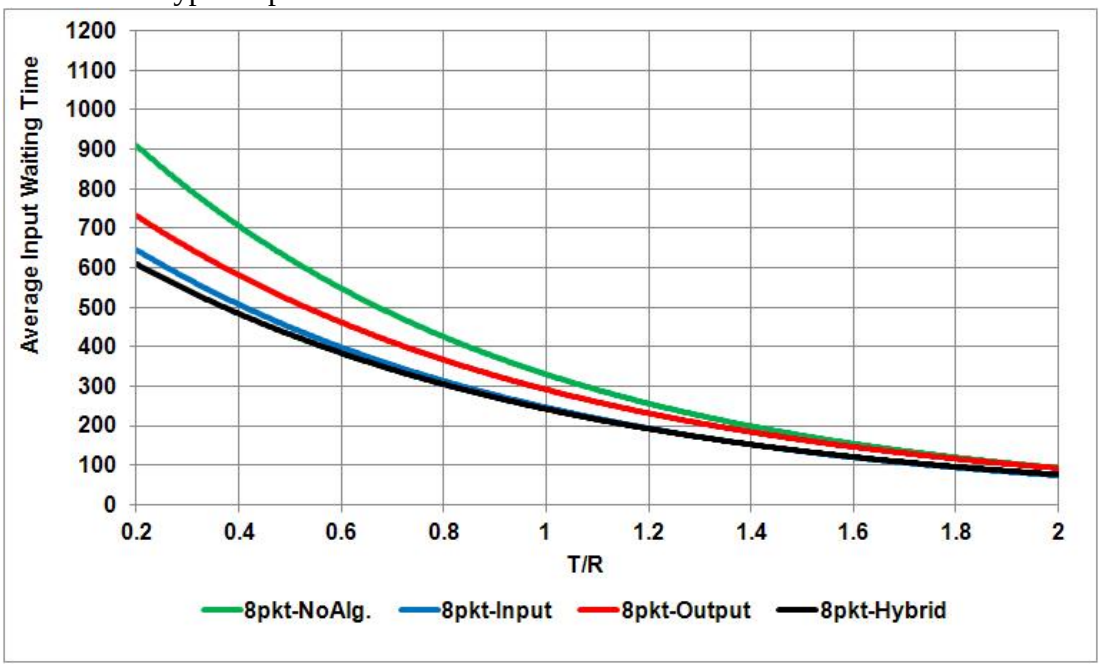

Figure 5.33.b. Average input waiting time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

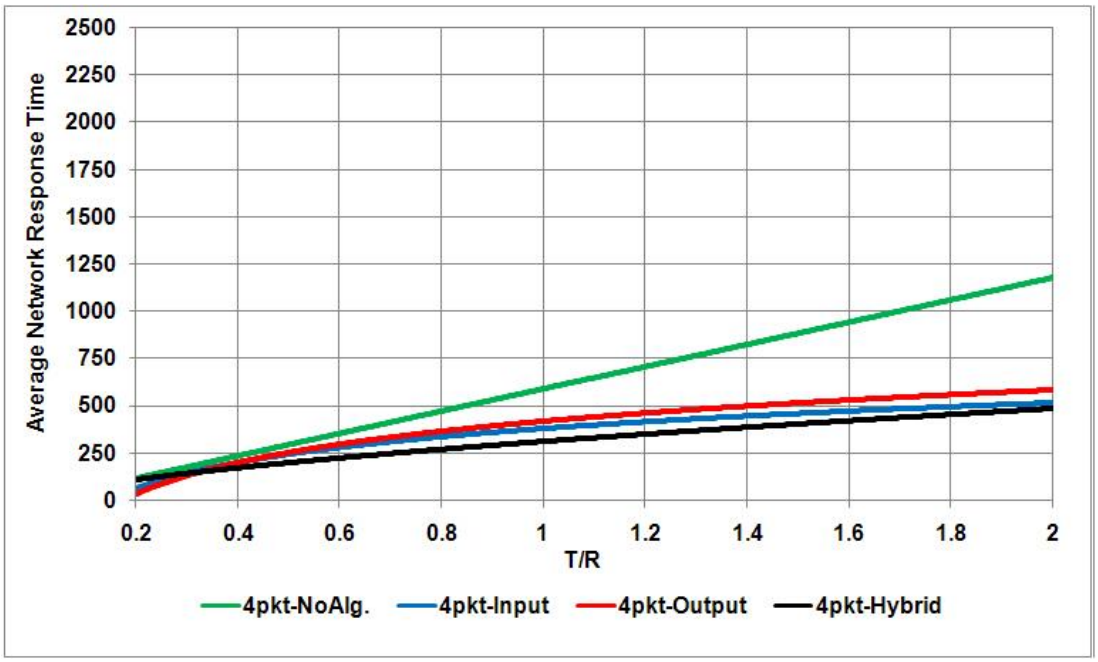

Figure 5.34.a. Average network response time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

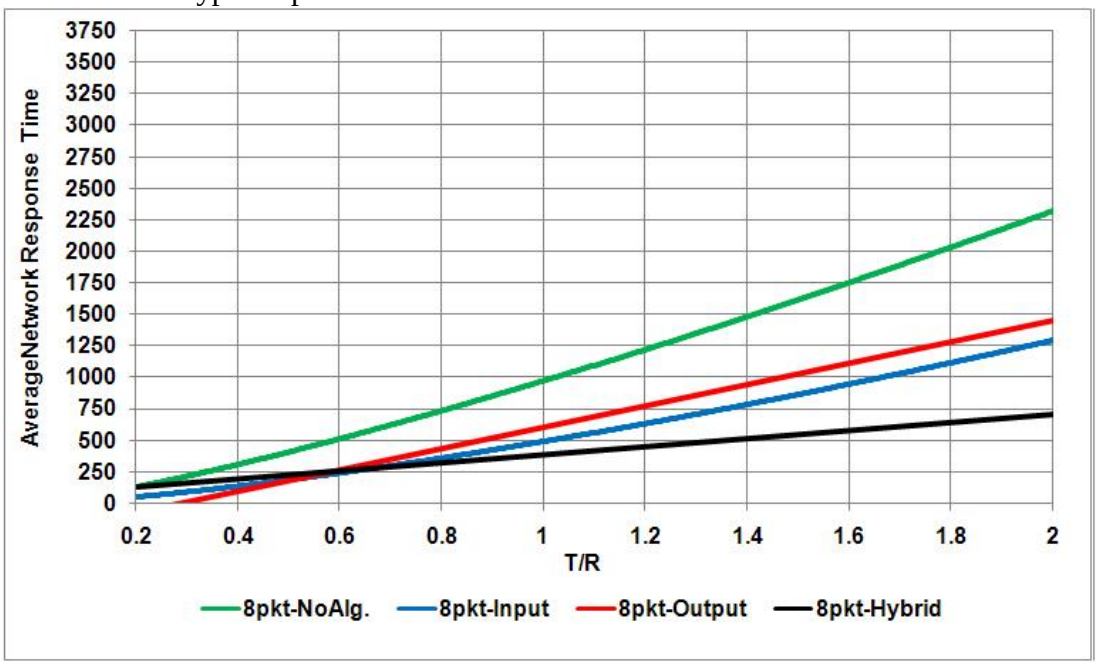

Figure 5.34.b. Average network response time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

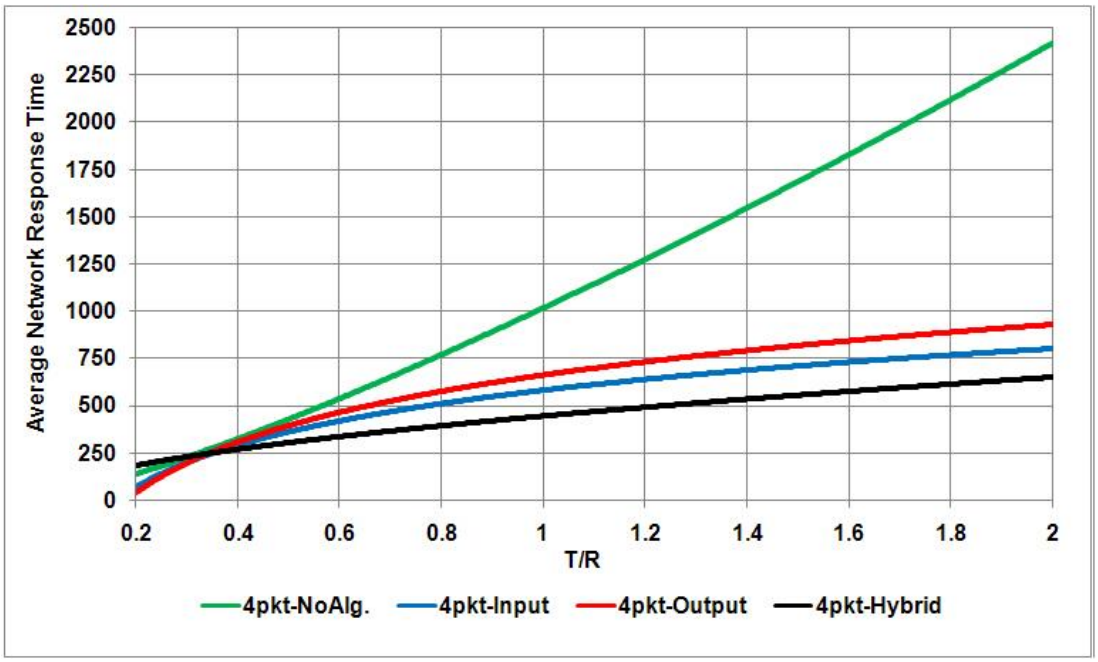

Figure 5.35.a. Average network response time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

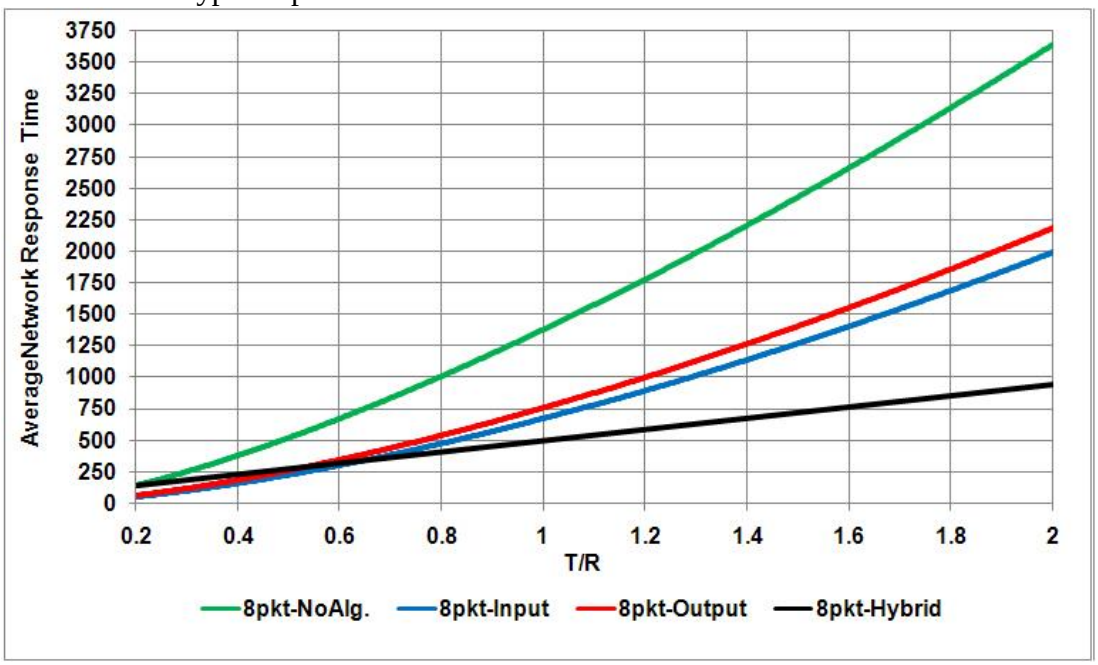

Figure 5.35.b. Average network response time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

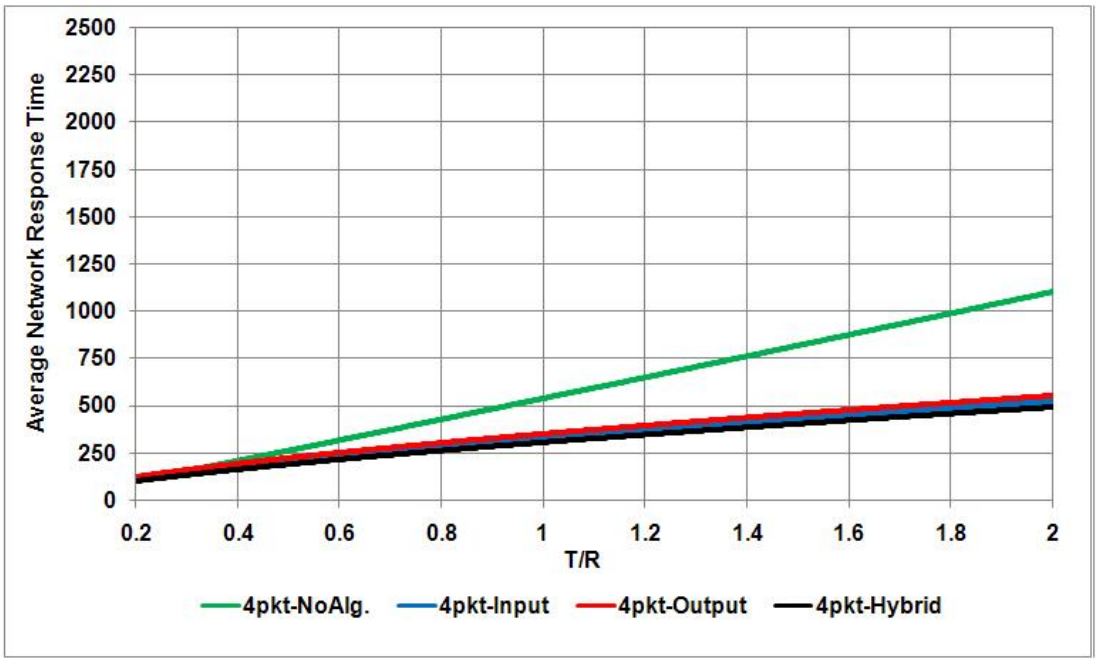

Figure 5.36.a. Average network response time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and bypass operation

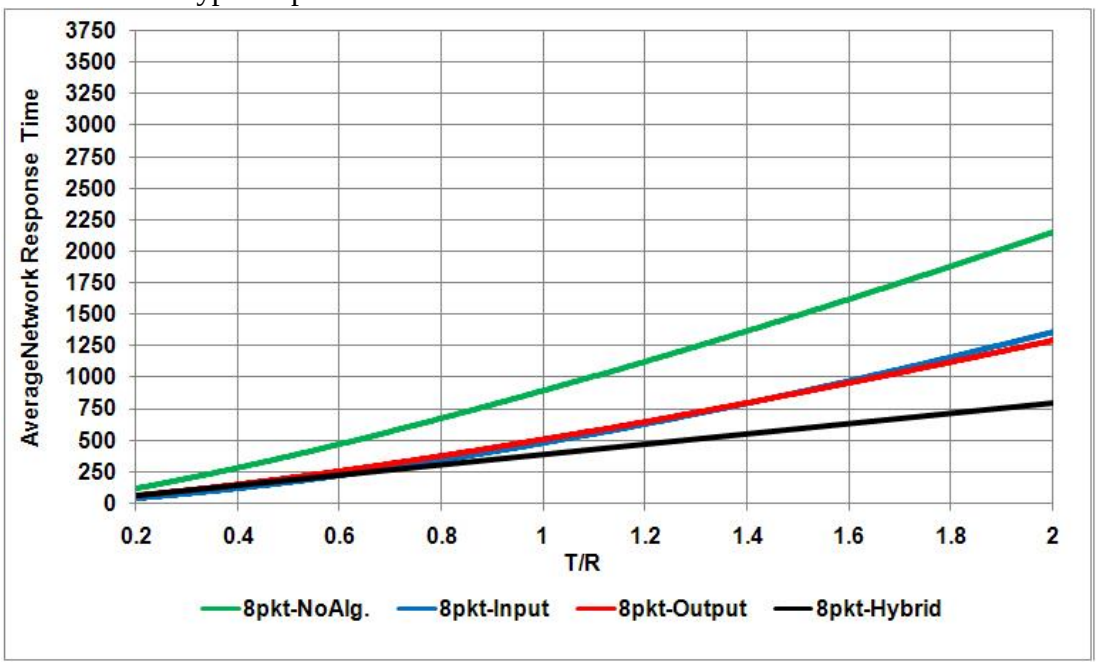

Figure 5.36.b. Average network response time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and bypass operation

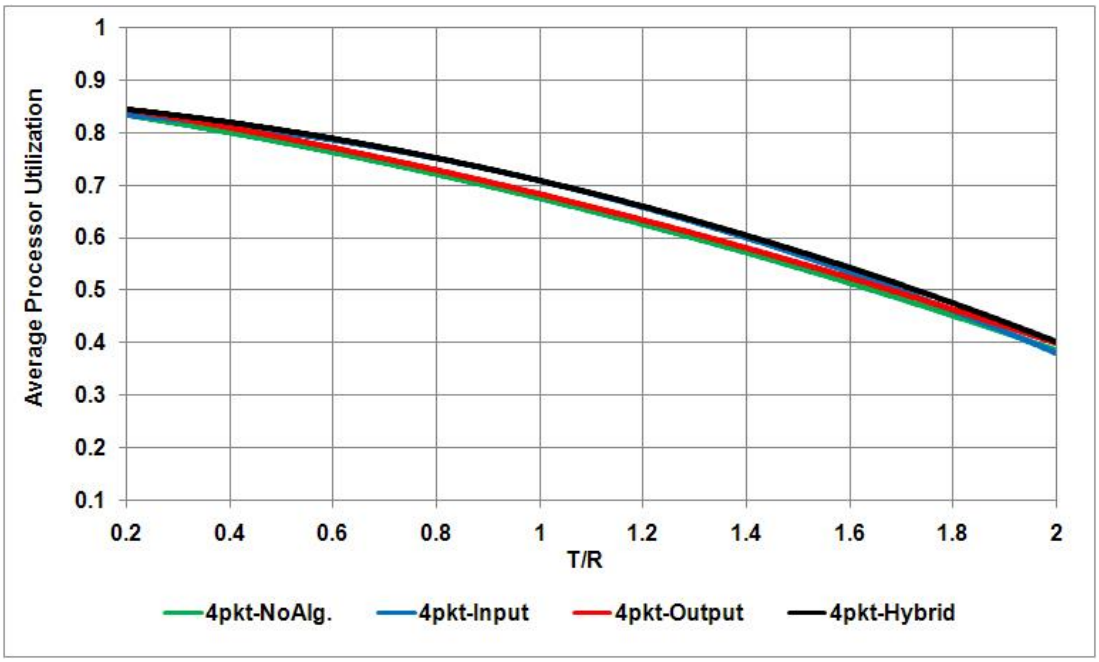

Figure 5.37.a. Average processor utilization for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

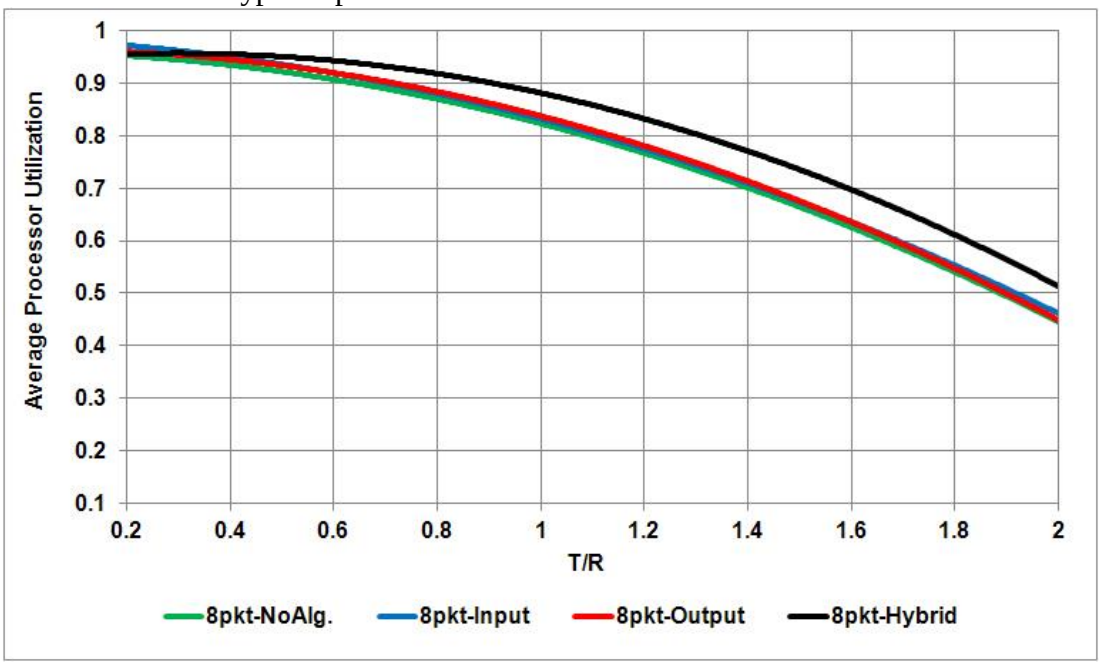

Figure 5.37.b. Average processor utilization for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

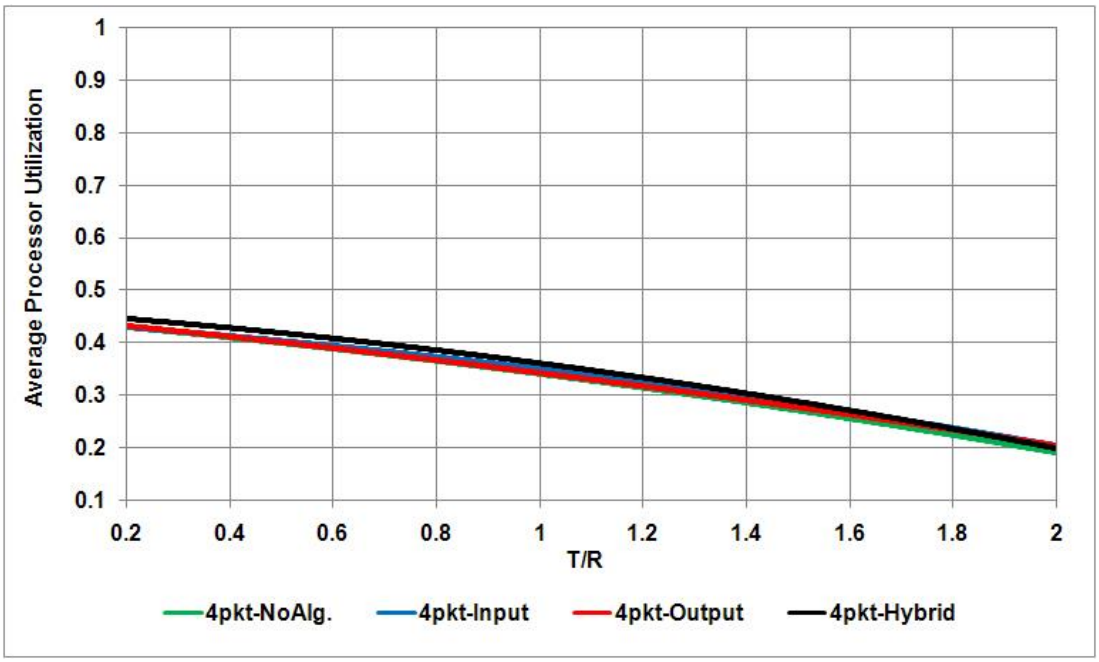

Figure 5.38.a. Average processor utilization for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

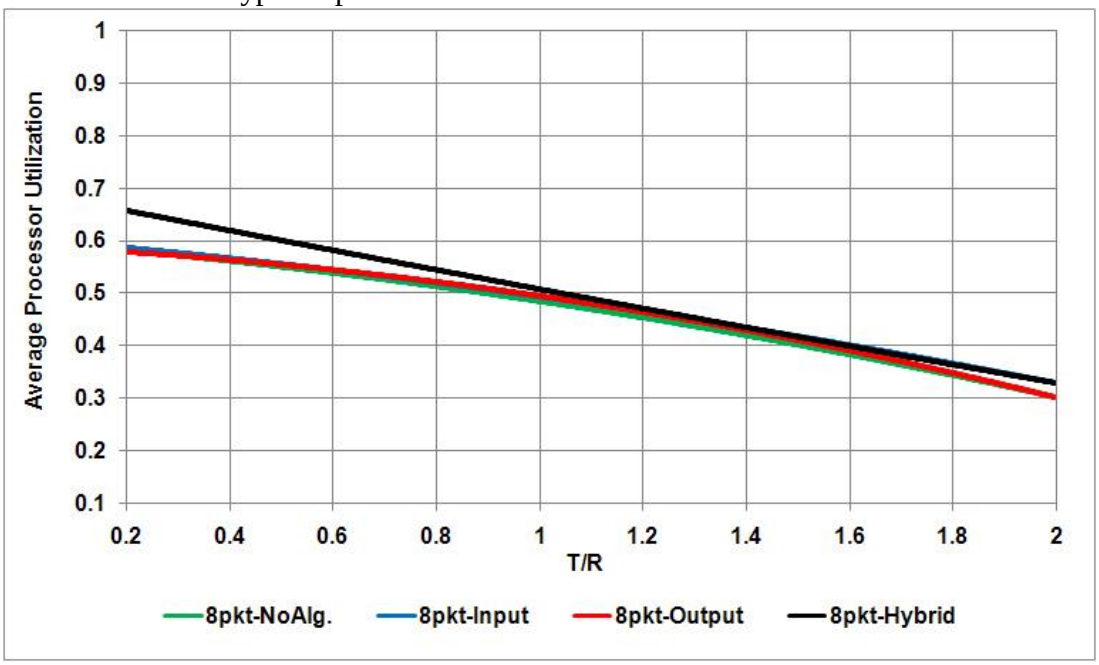

Figure 5.38.b. Average processor utilization for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

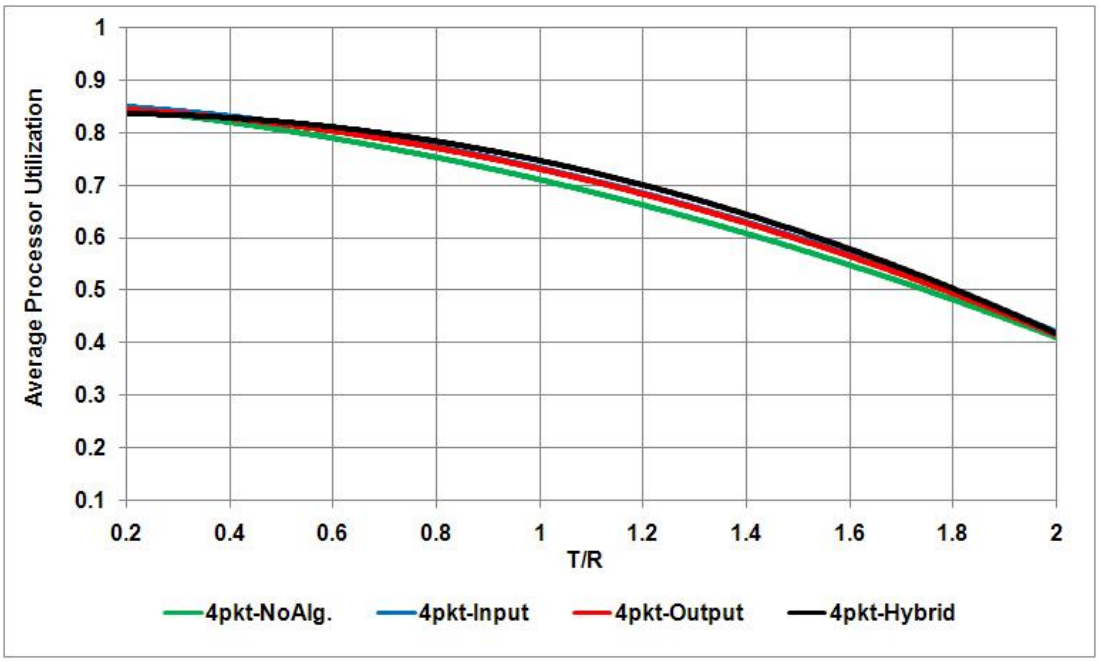

Figure 5.39.a. Average processor utilization for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

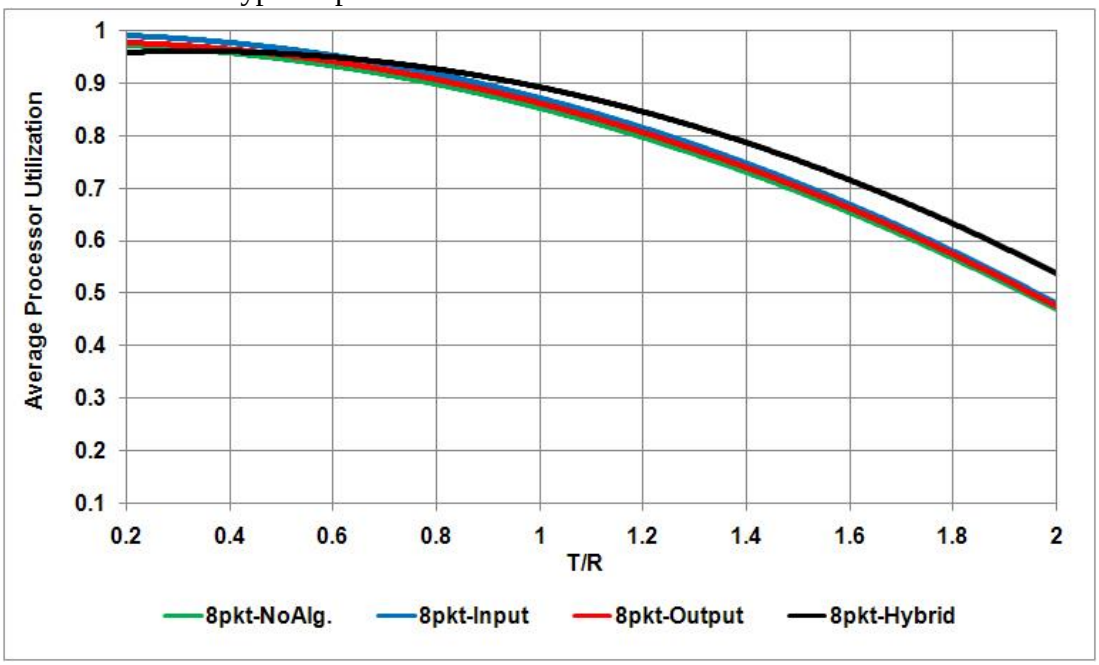

Figure 5.39.b. Average processor utilization for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

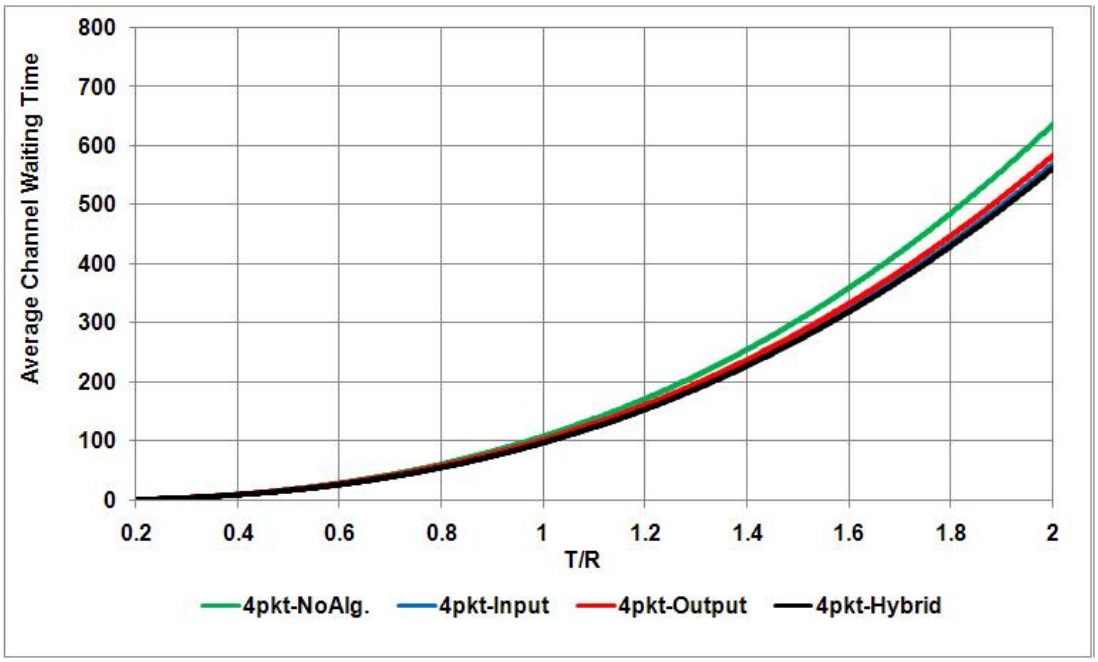

Figure 5.40.a. Average channel waiting time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

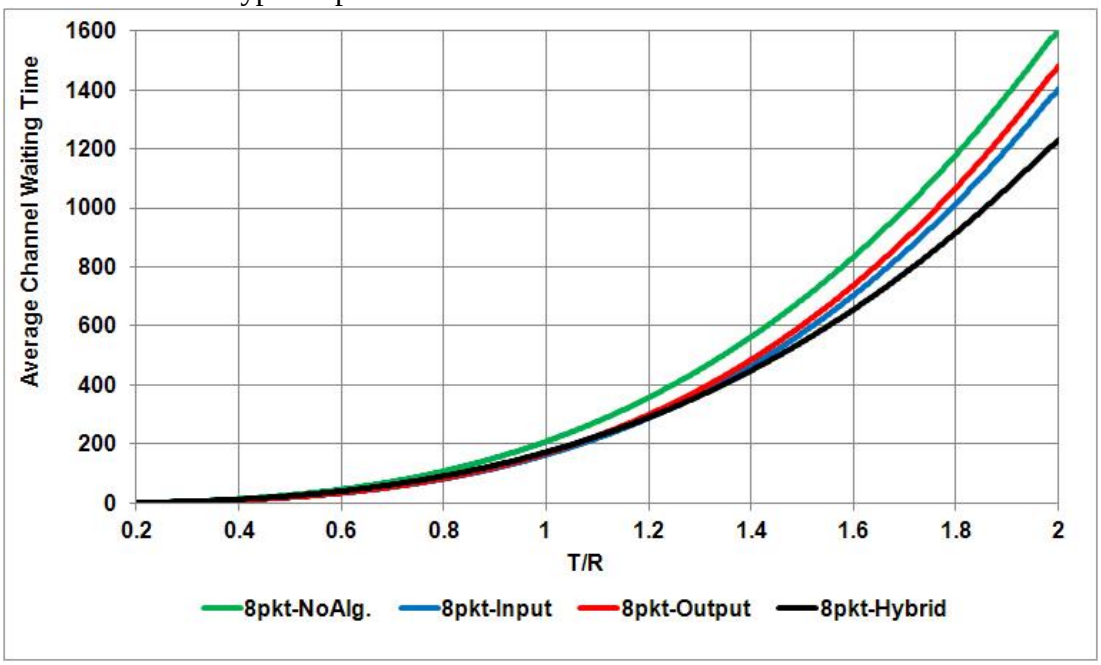

Figure 5.40.b. Average channel waiting time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

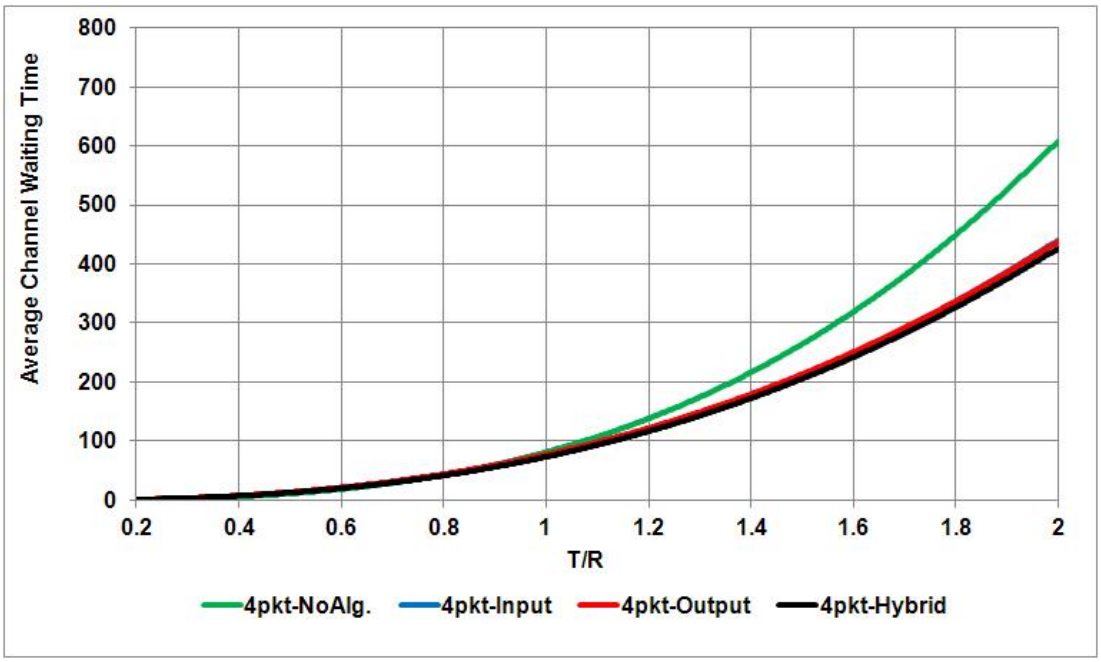

Figure 5.41.a. Average channel waiting time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

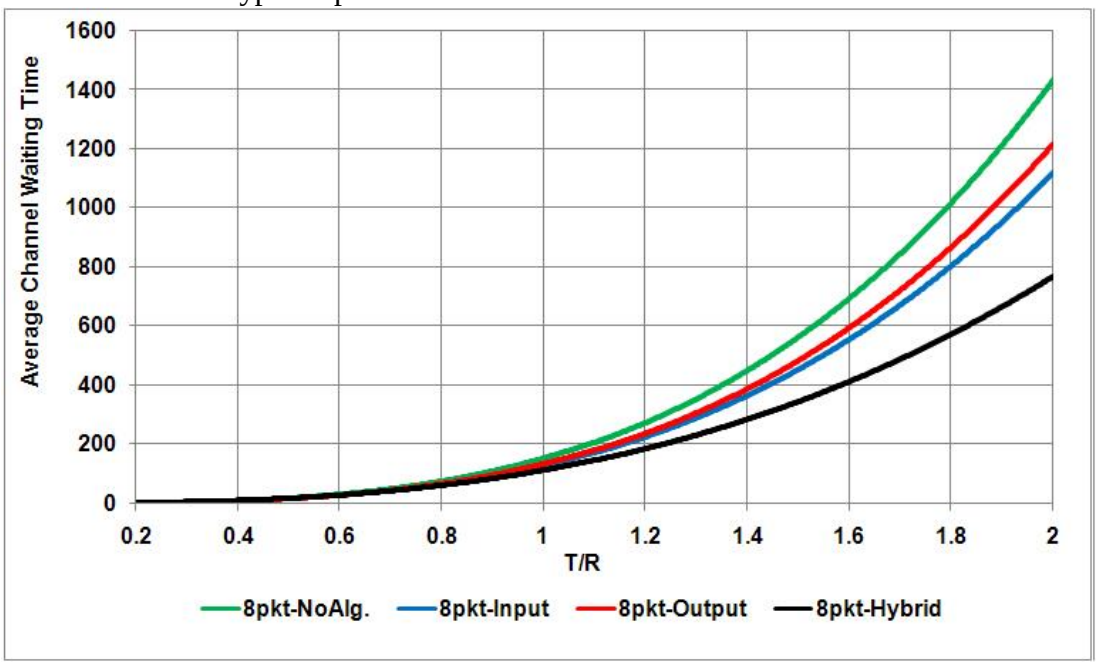

Figure 5.41.b. Average channel waiting time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

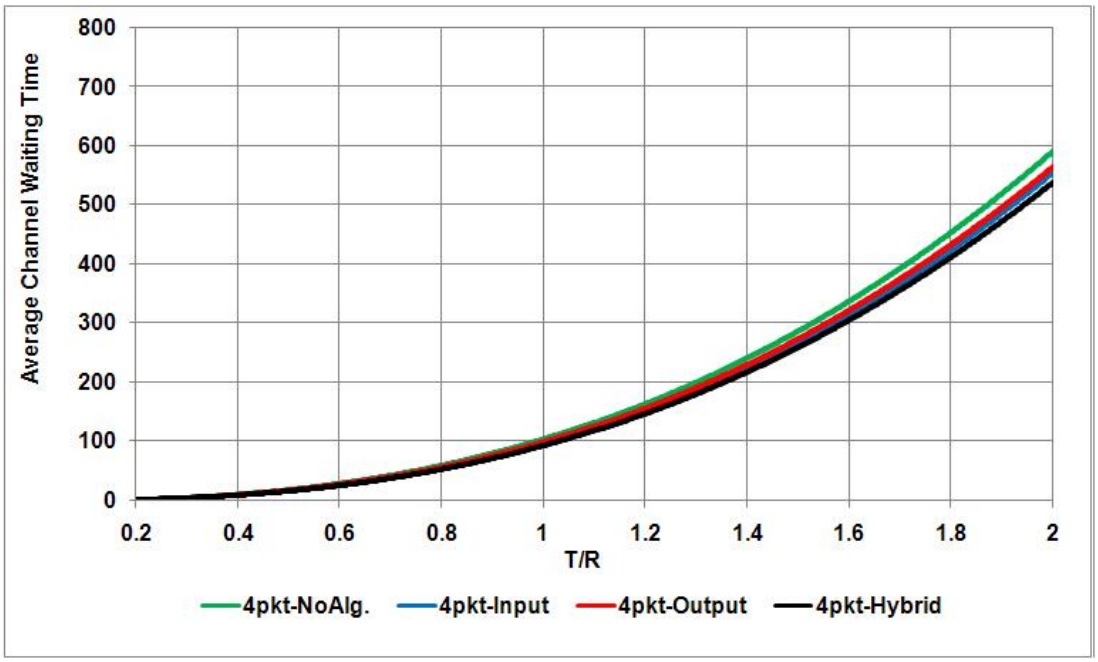

Figure 5.42.a. Average channel waiting time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

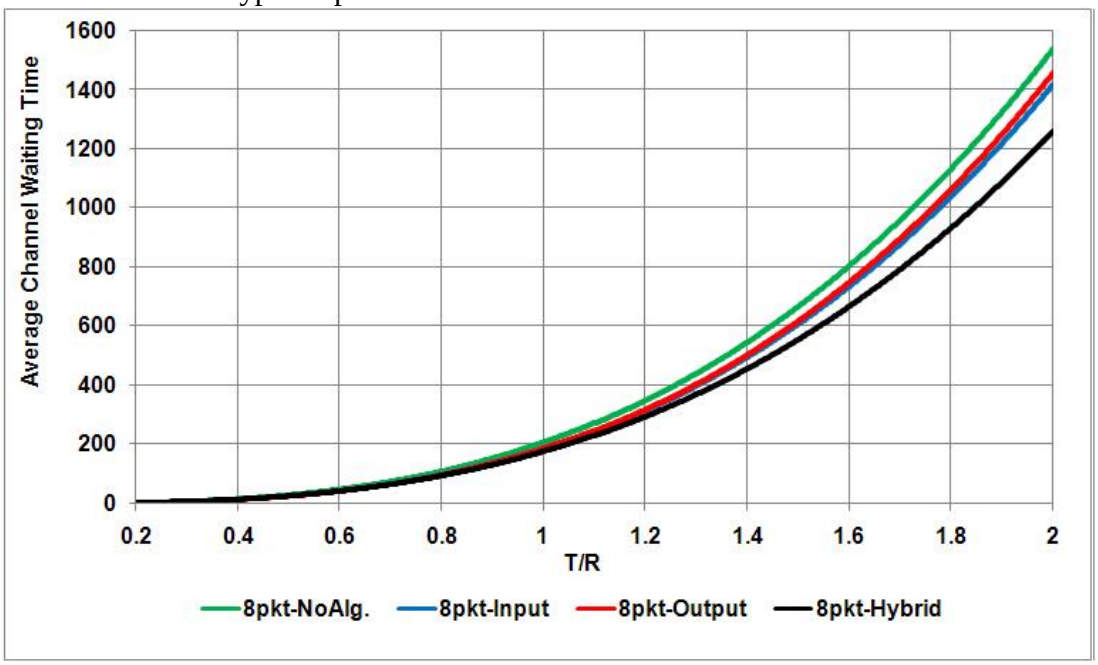

Figure 5.42.b. Average channel waiting time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

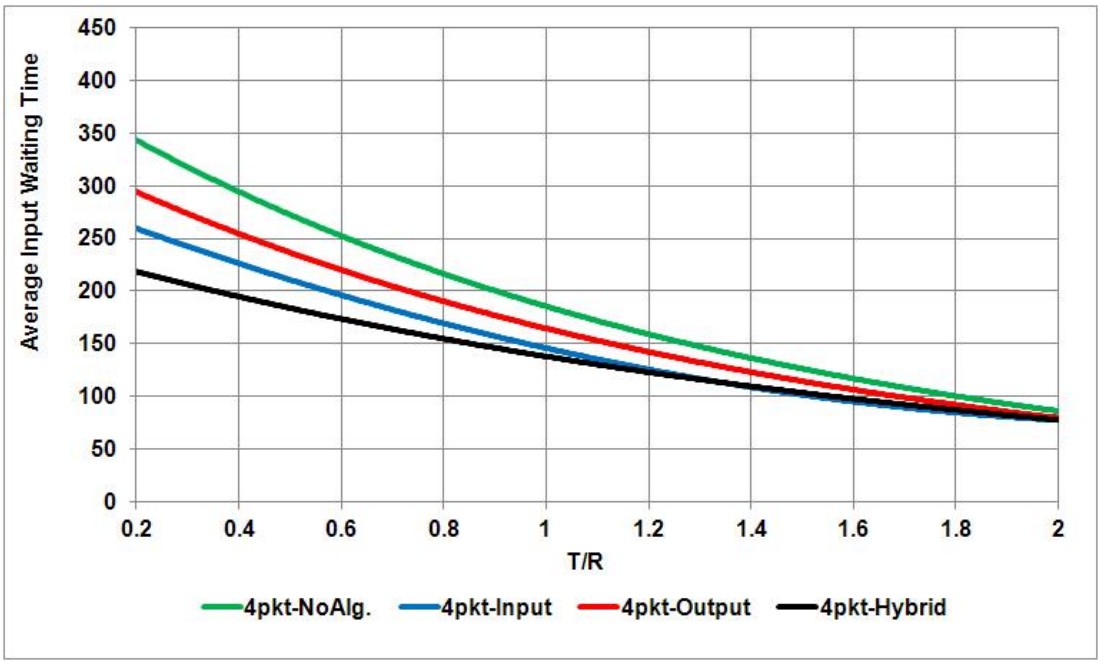

Figure 5.43.a. Average input waiting time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

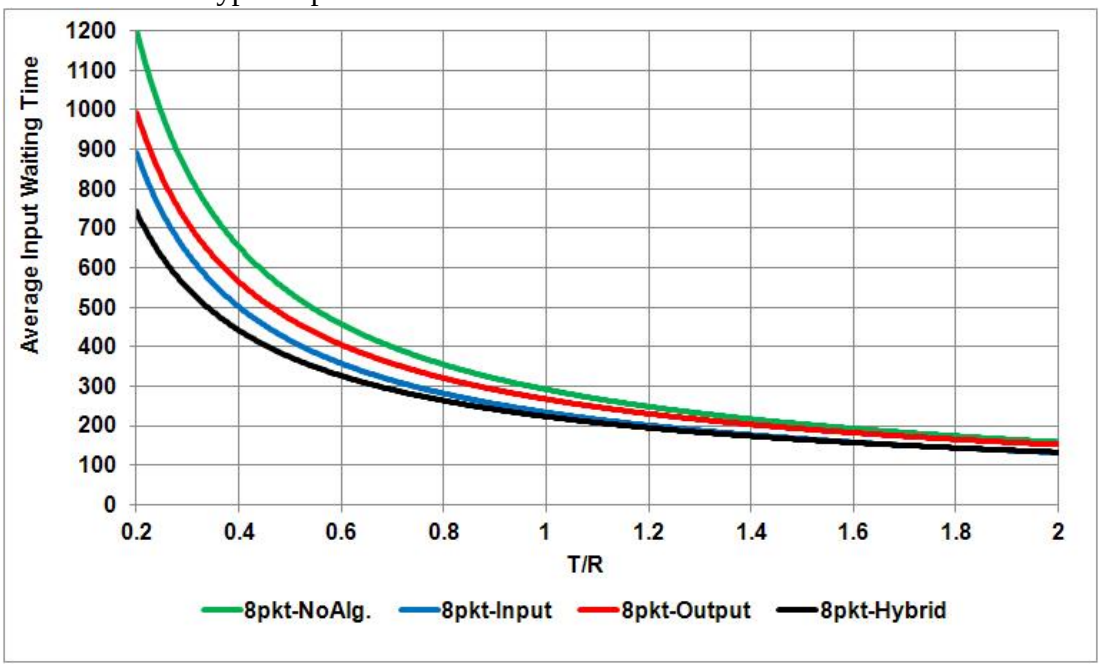

Figure 5.43.b. Average input waiting time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

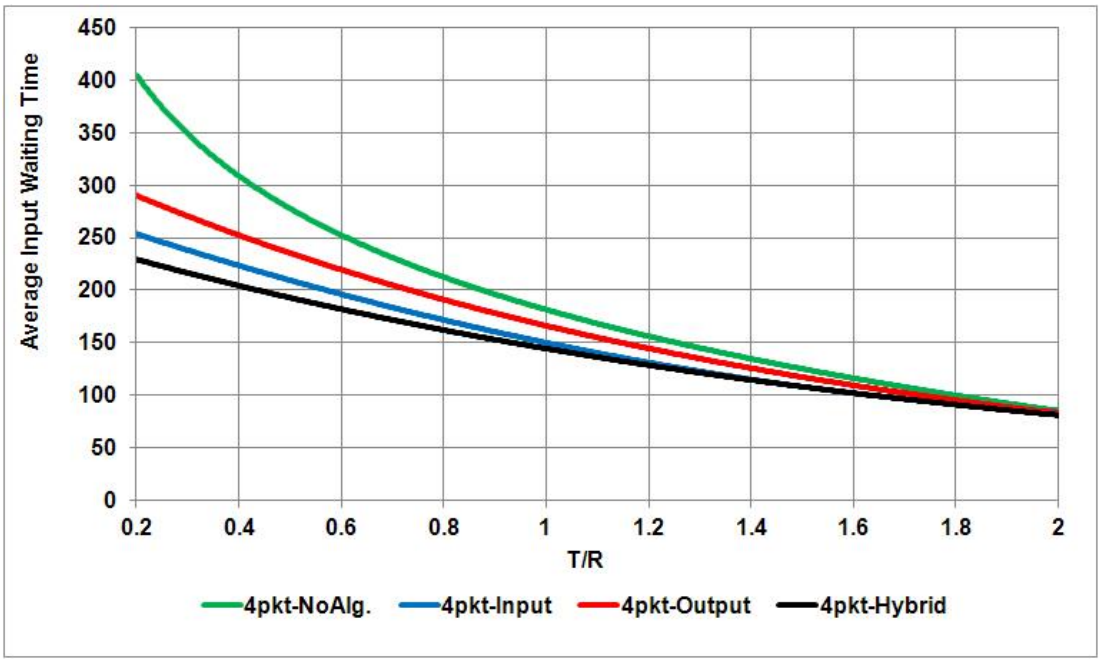

Figure 5.44.a. Average input waiting time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

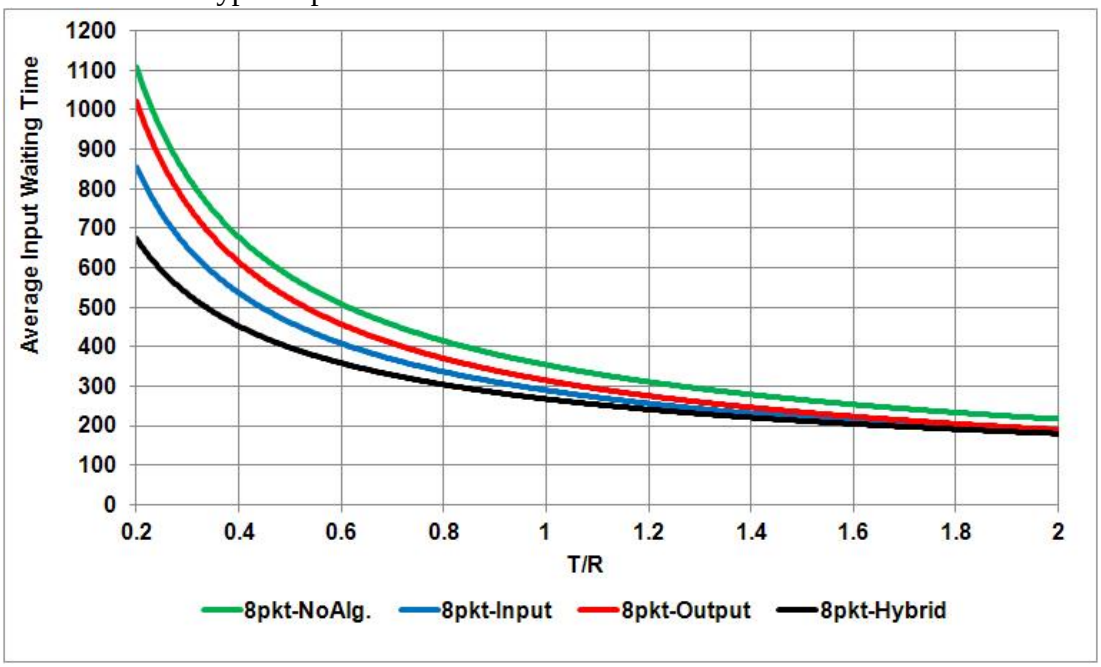

Figure 5.44.b. Average input waiting time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

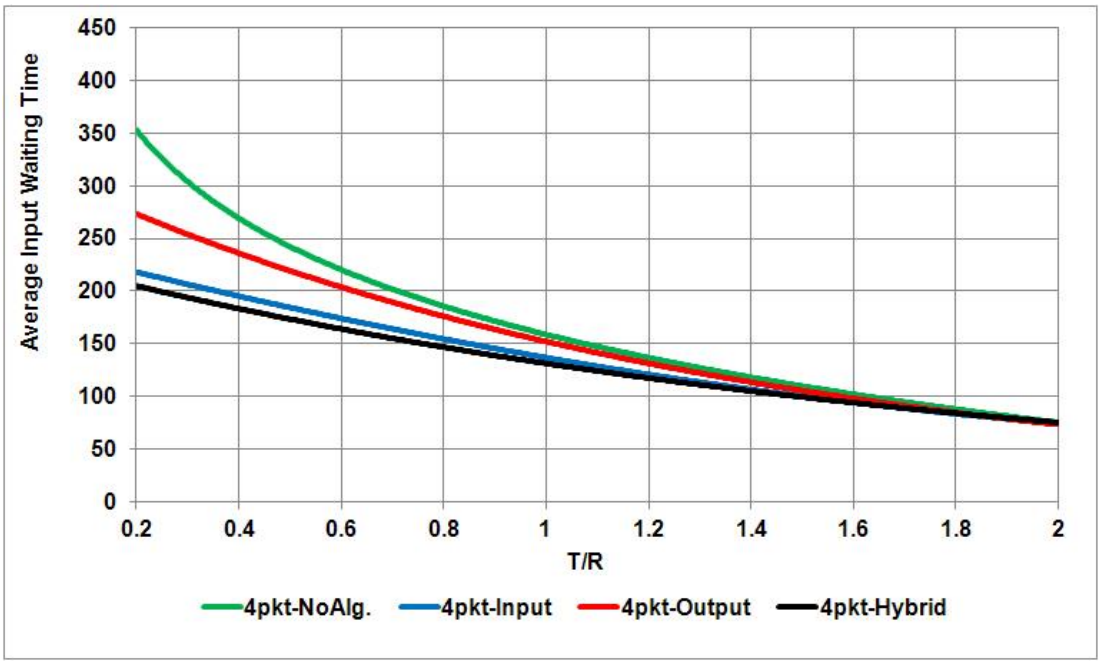

Figure 5.45.a. Average input waiting time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

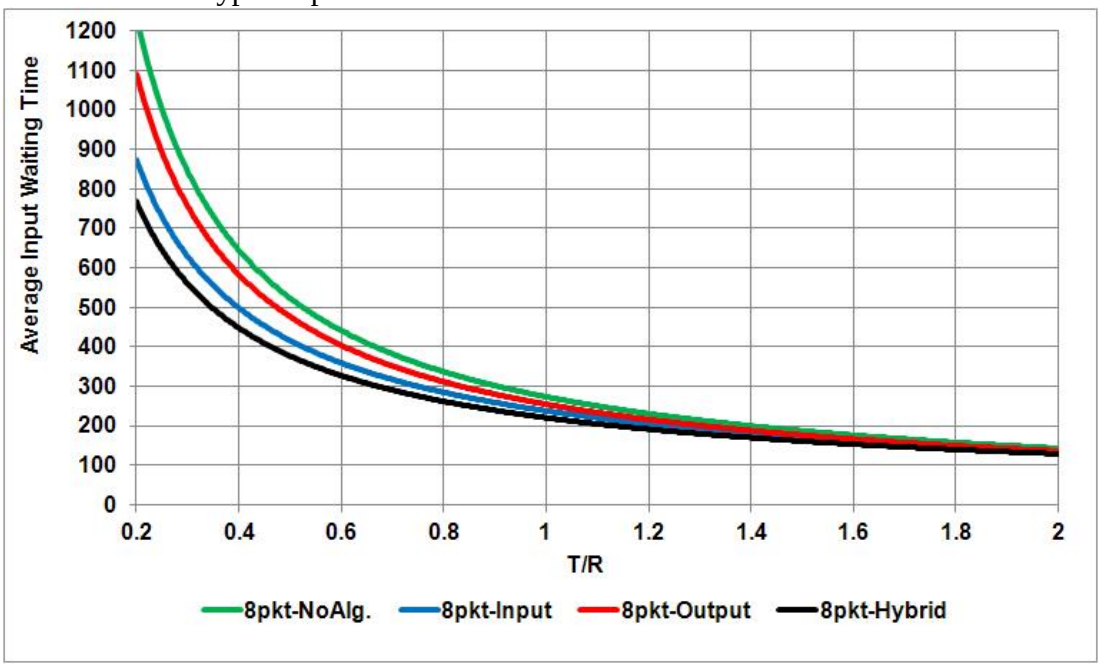

Figure 5.45.b. Average input waiting time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

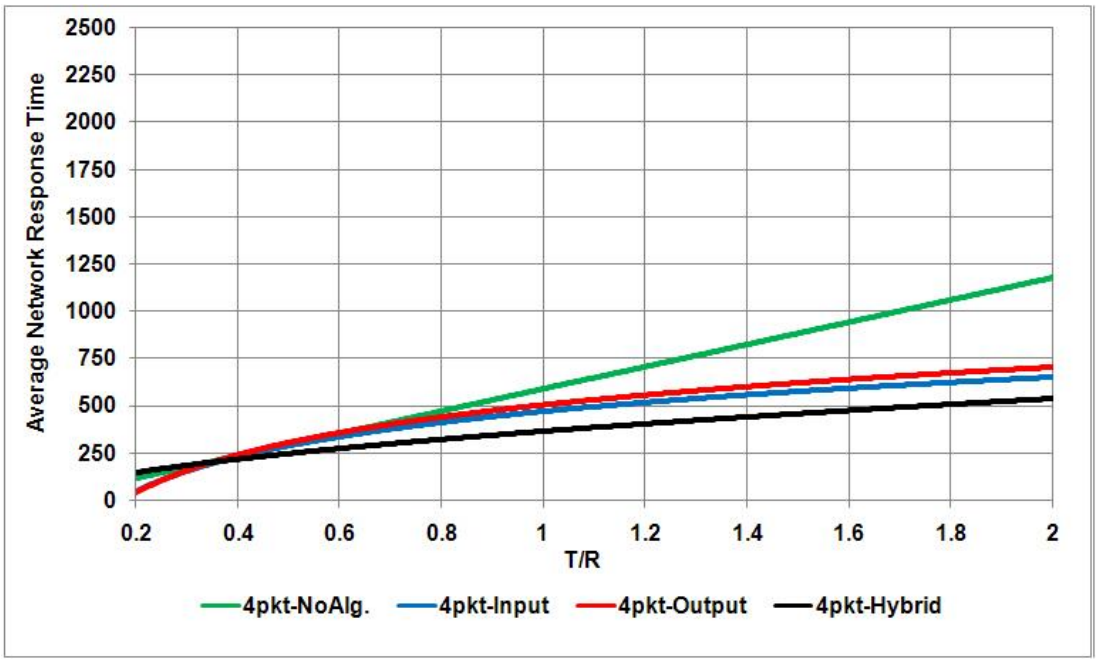

Figure 5.46.a. Average network response time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

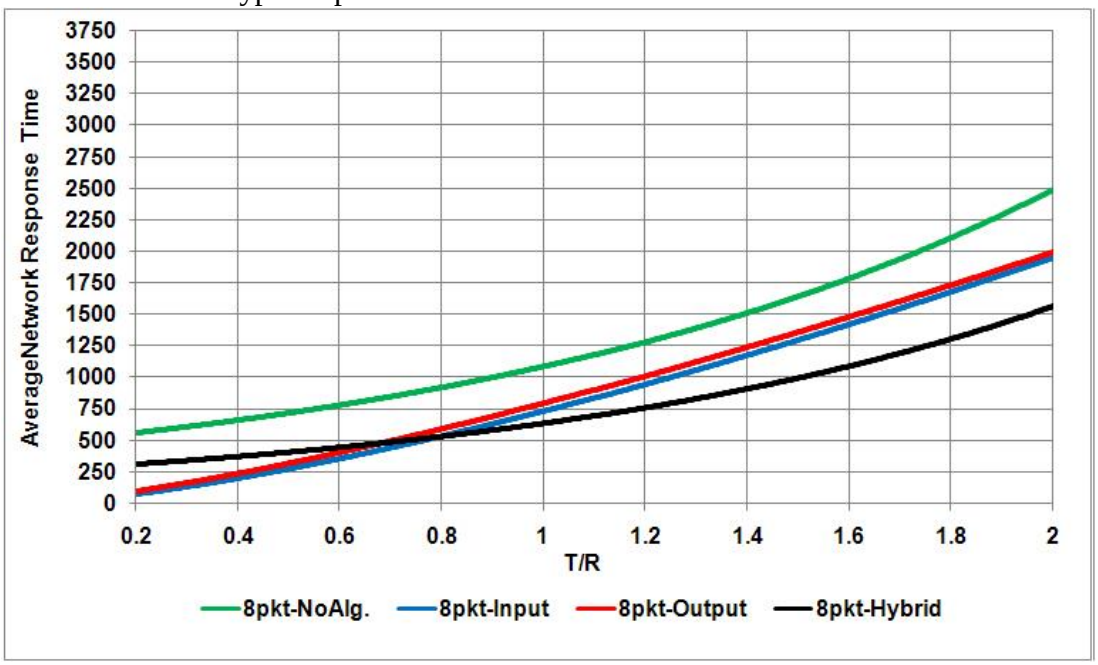

Figure 5.46.b. Average network response time for Asynchronous Message Passing traffic model under UN traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

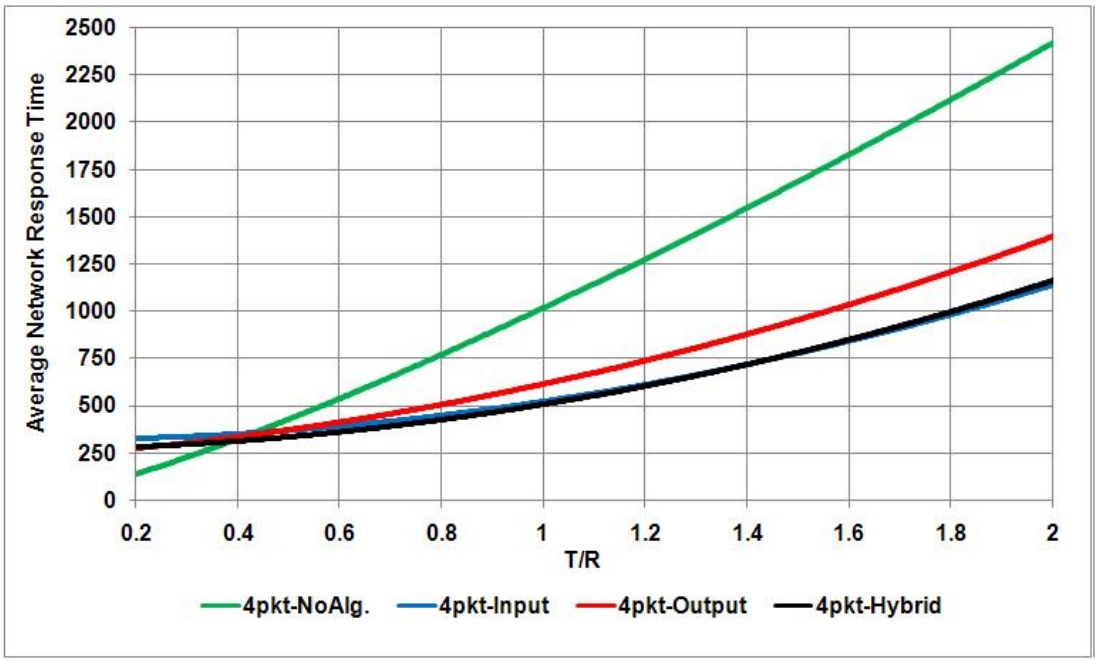

Figure 5.47.a. Average network response time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

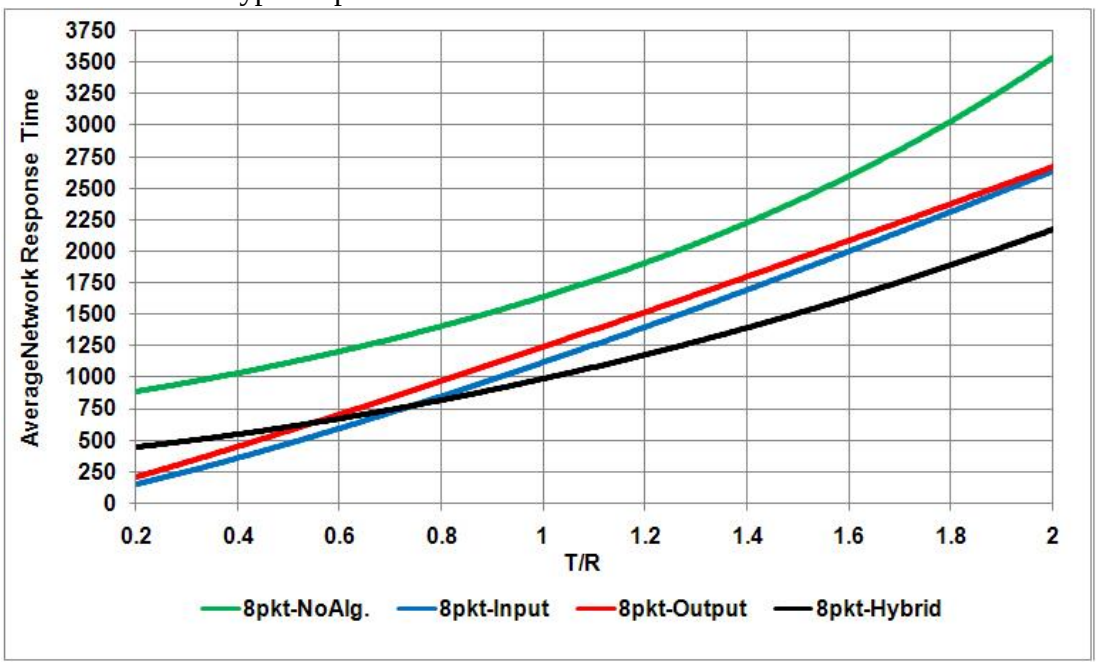

Figure 5.47.b. Average network response time for Asynchronous Message Passing traffic model under HR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

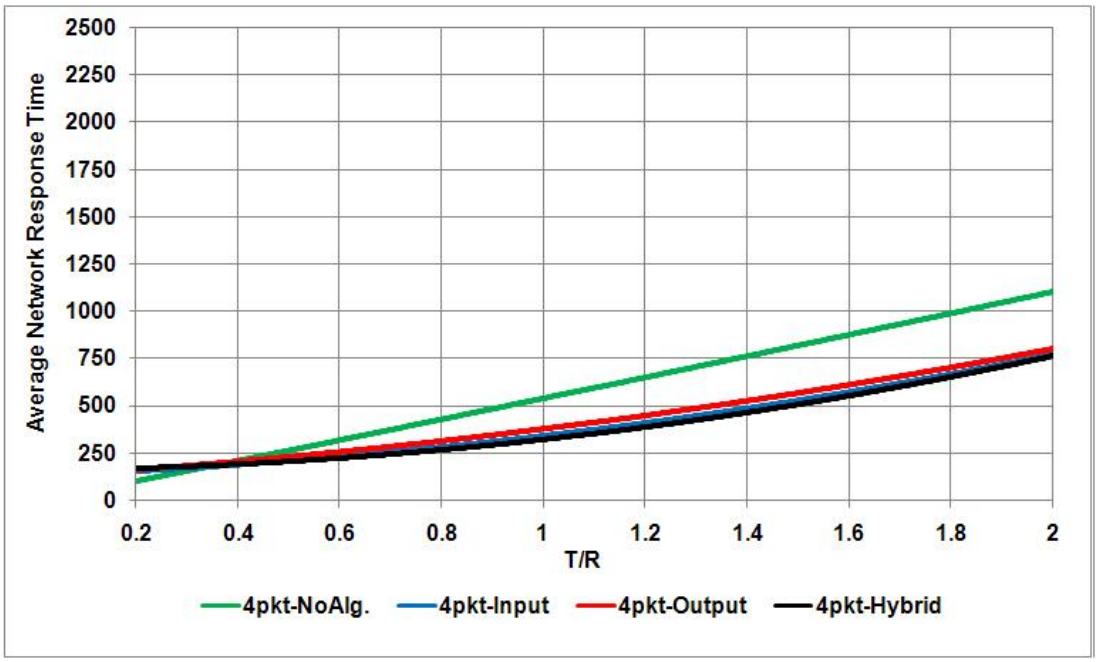

Figure 5.48.a. Average network response time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 4 threads and no-bypass operation

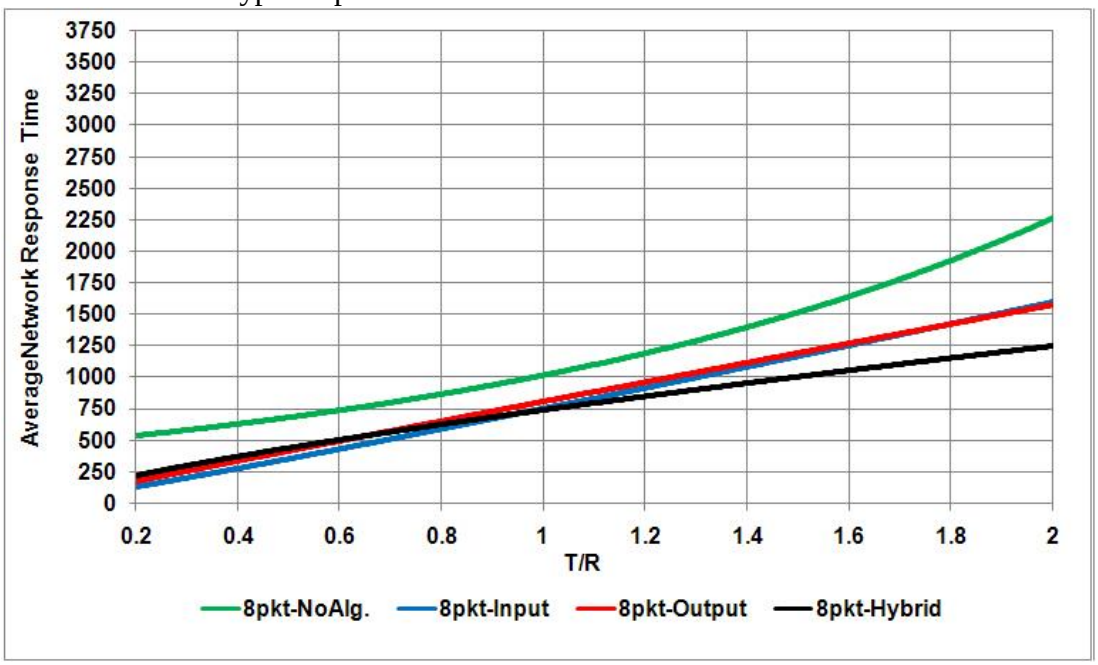

Figure 5.48.b. Average network response time for Asynchronous Message Passing traffic model under BR traffic pattern using input, output and hybrid parameters with 8 threads and no-bypass operation

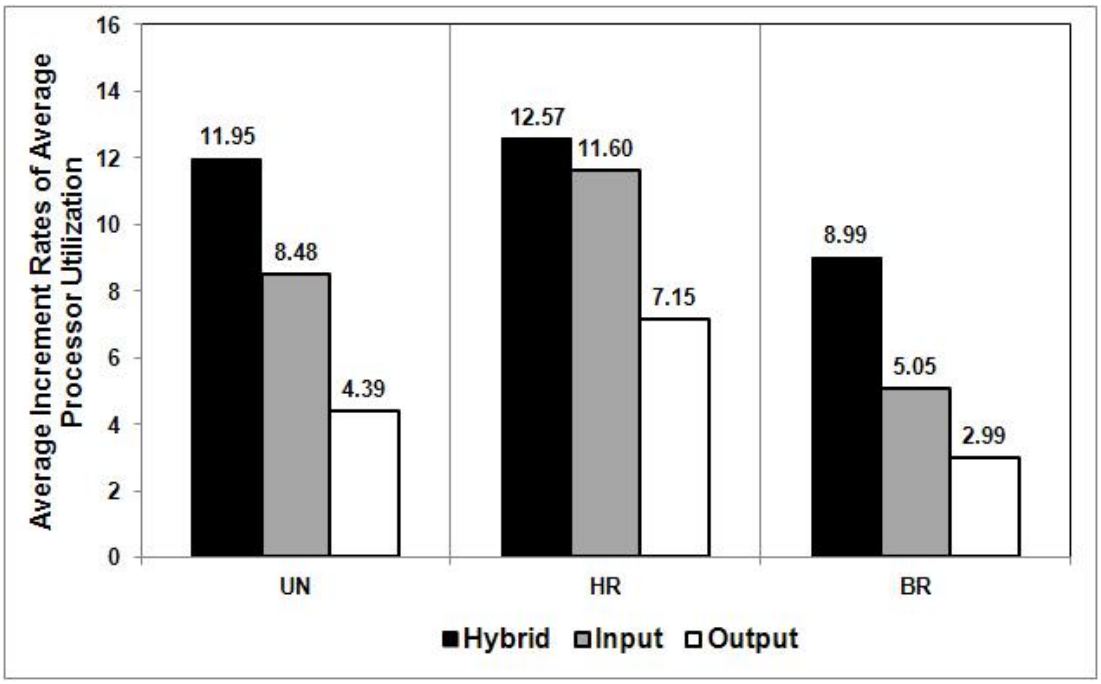

Figure 5.49.a. The average rate of processor utilization increment rates for all traffic patterns under Client-Server traffic model with 4 threads and bypass operation

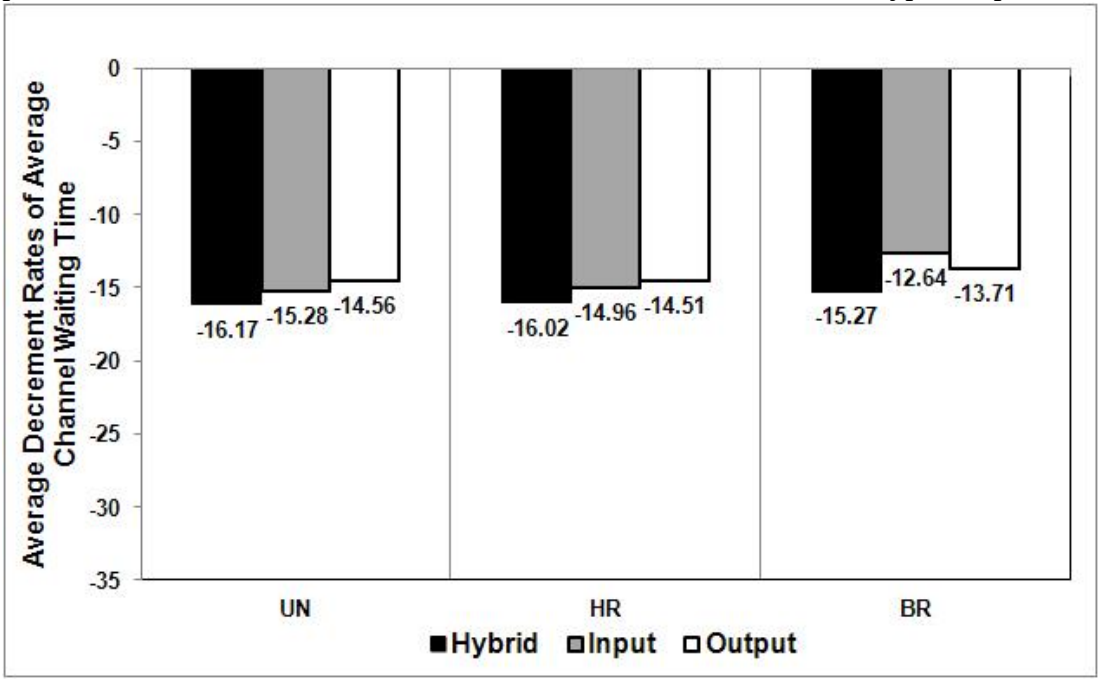

Figure 5.49.b. The average rate of channel waiting time decrement rates for all traffic patterns under Client-Server traffic model with 4 threads and bypass operation

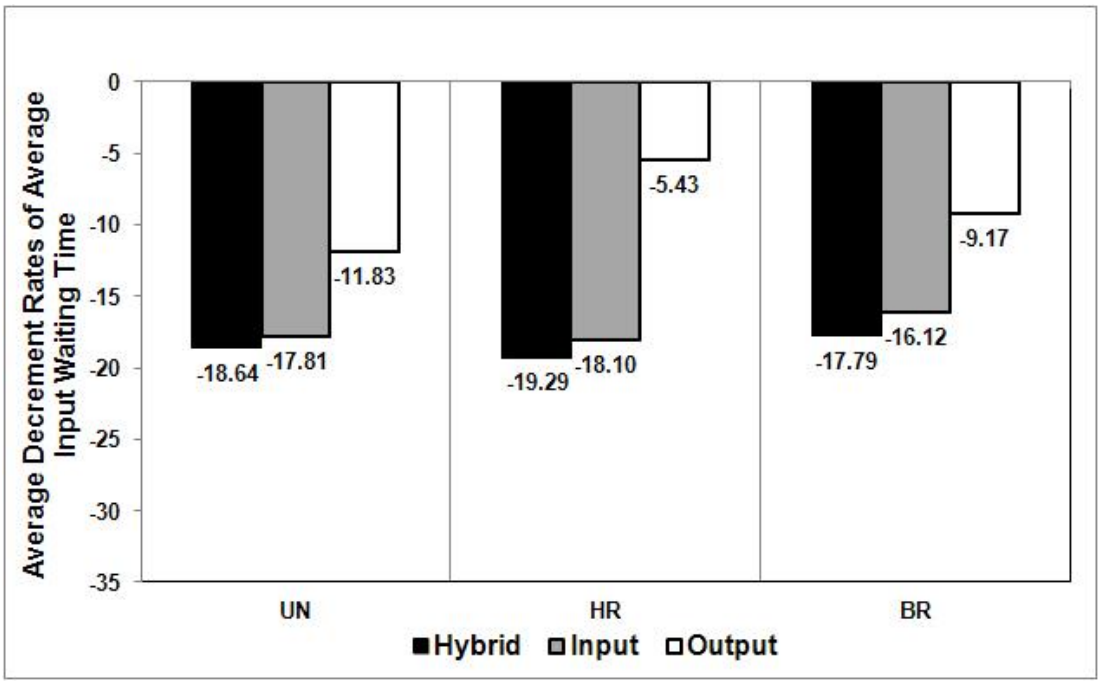

Figure 5.49.c. The average rate of input waiting time decrement rates for all traffic patterns under Client-Server traffic model with 4 threads and bypass operation

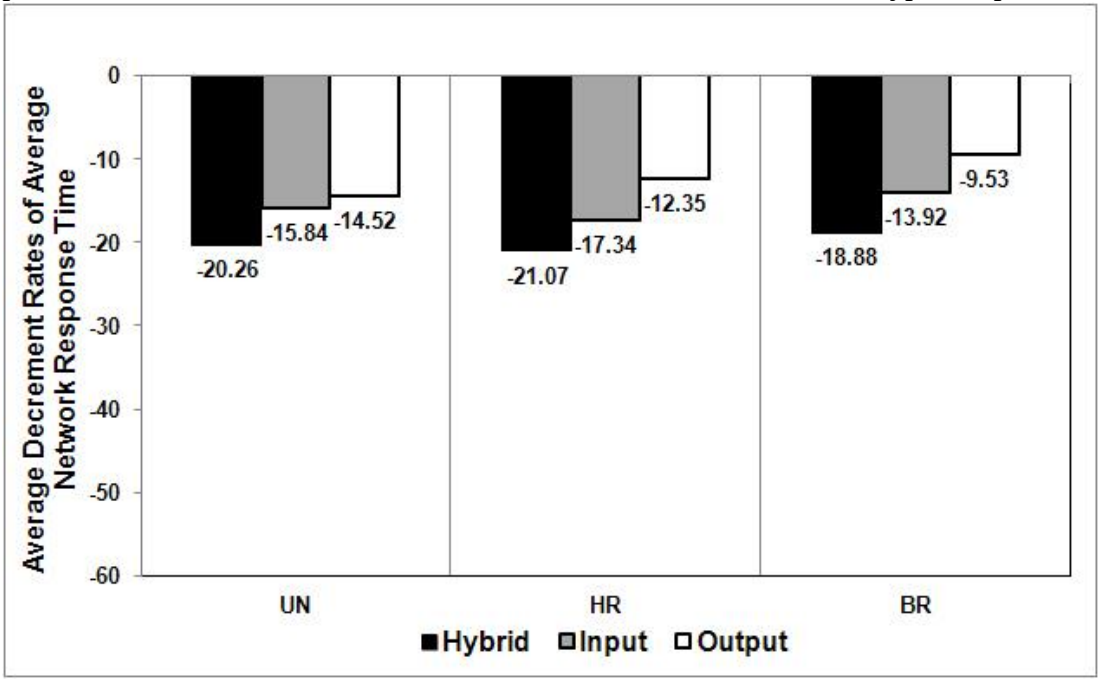

Figure 5.49.d. The average rate of network response time decrement rates for all traffic patterns under Client-Server traffic model with 4 threads and bypass operation

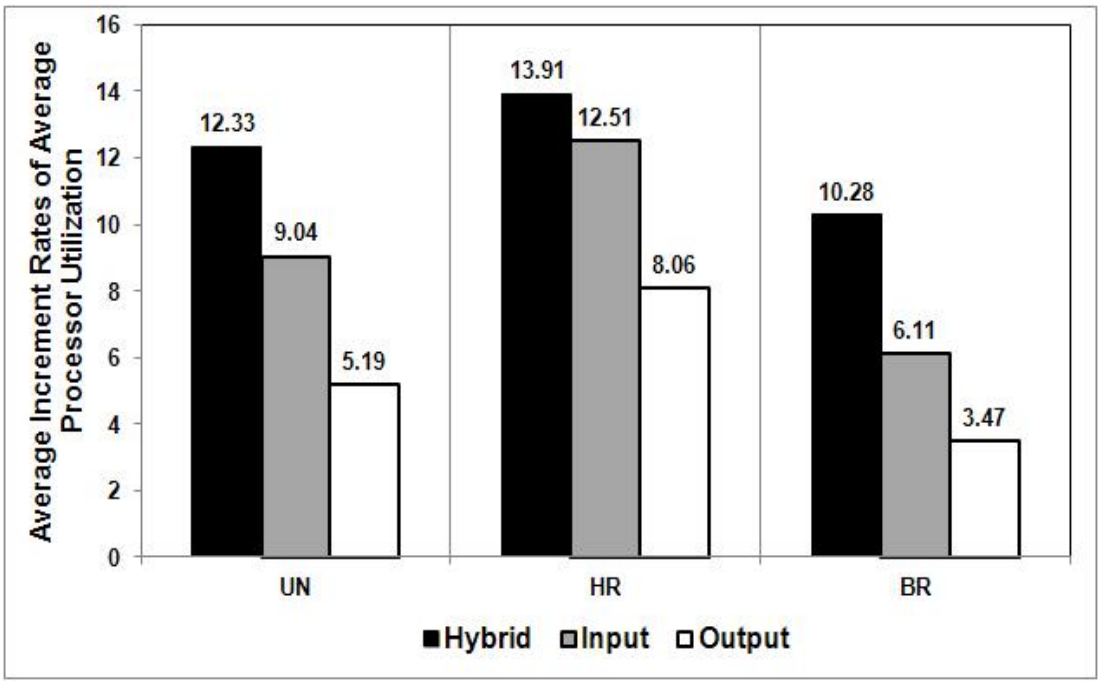

Figure 5.50.a. The average rate of processor utilization increment rates for all traffic patterns under Client-Server traffic model with 8 threads and bypass operation

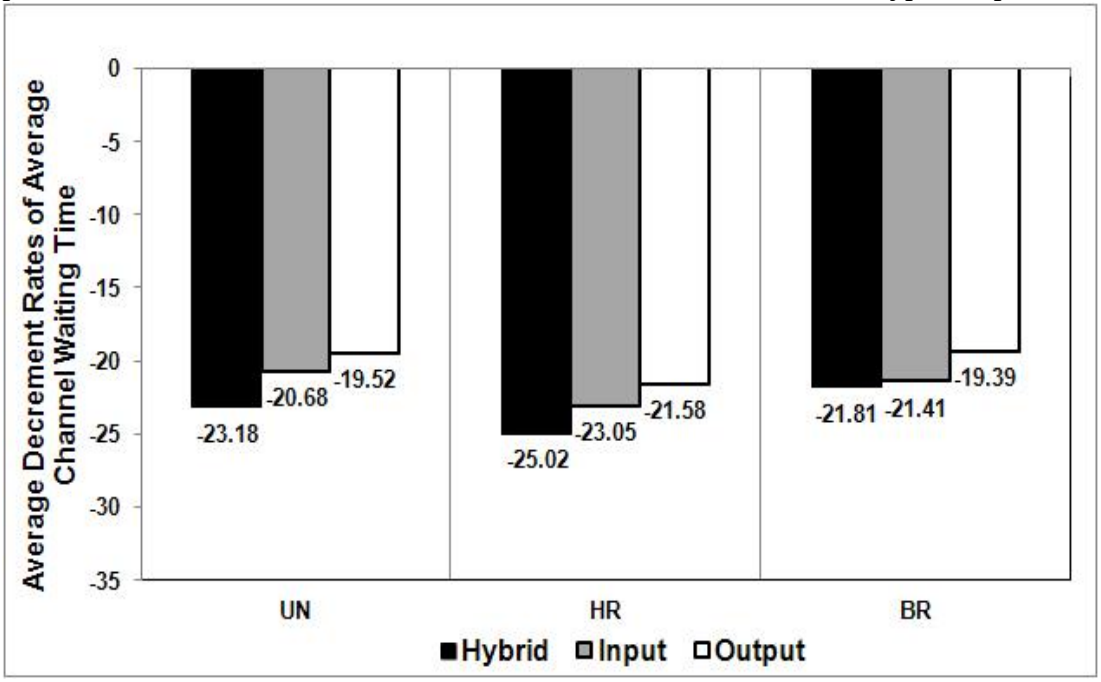

Figure 5.50.b. The average rate of channel waiting time decrement rates for all traffic patterns under Client-Server traffic model with 8 threads and bypass operation

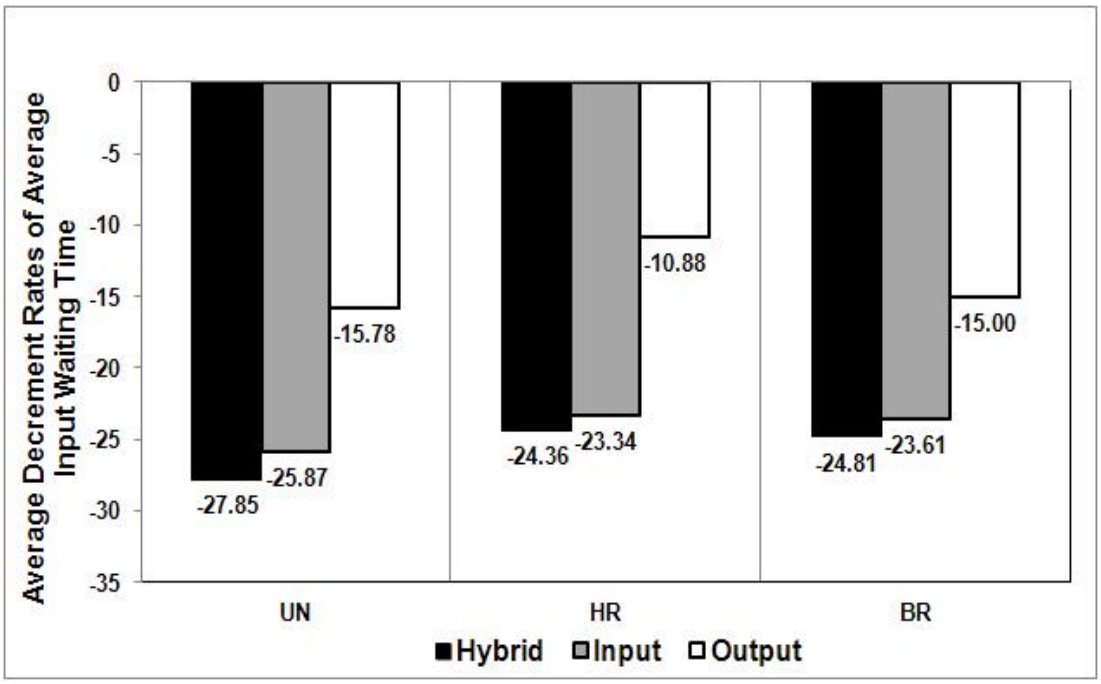

Figure 5.50.c. The average rate of input waiting time decrement rates for all traffic patterns under Client-Server traffic model with 8 threads and bypass operation

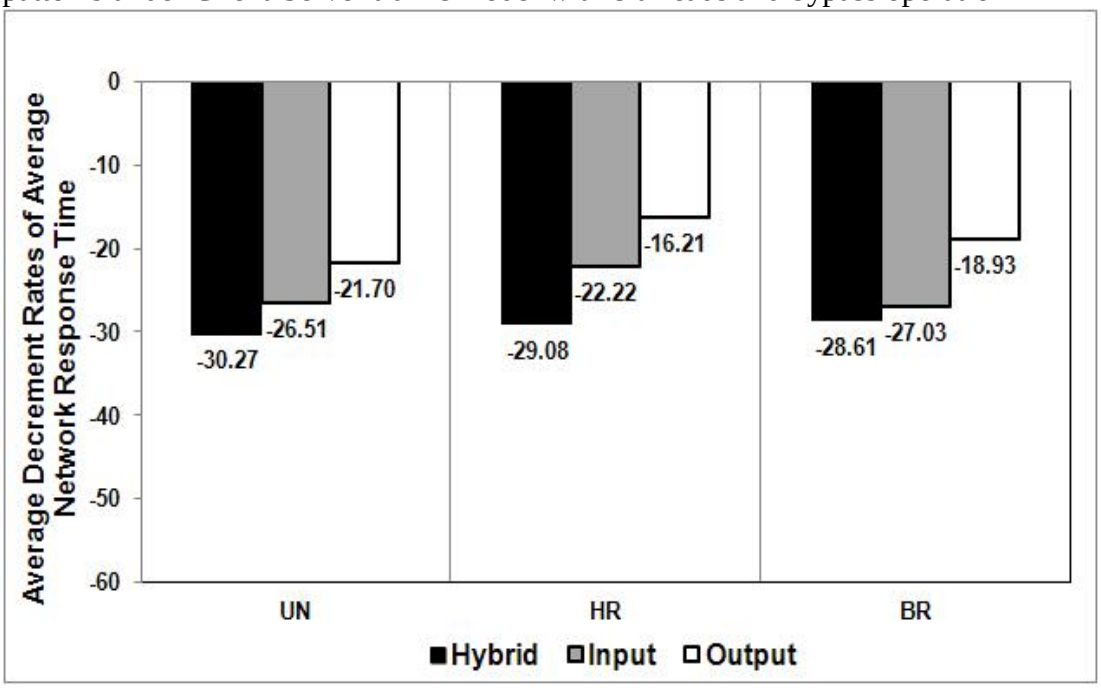

Figure 5.50.d. The average rate of network response time decrement rates for all traffic patterns under Client-Server traffic model with 8 threads and bypass operation

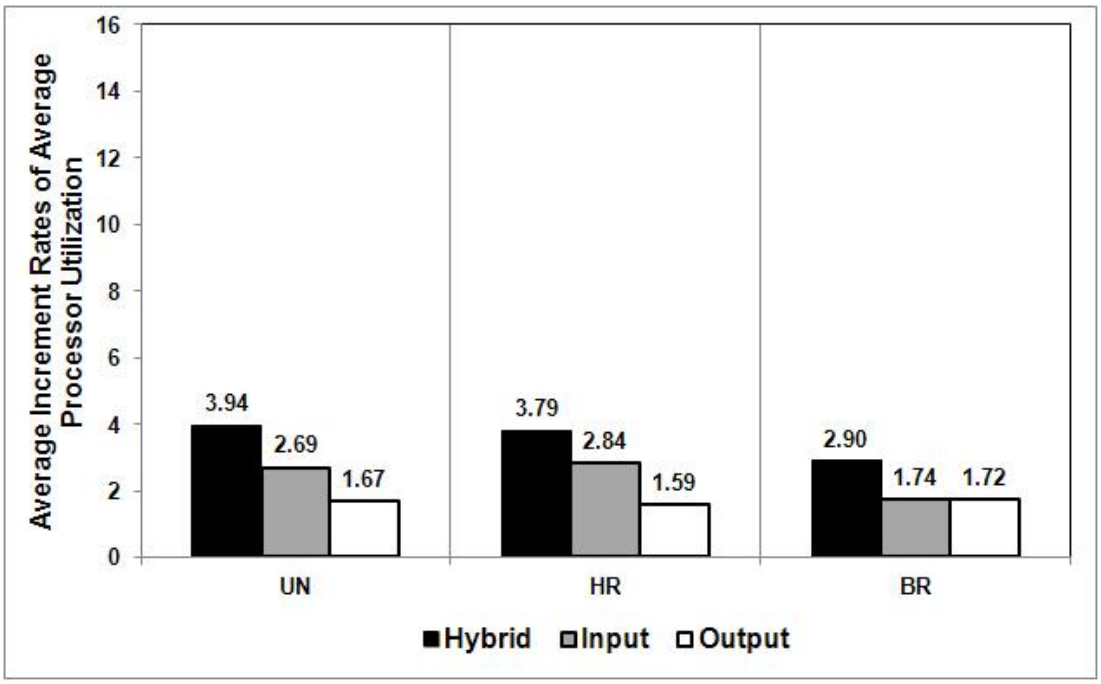

Figure 5.51.a. The average rate of processor utilization increment rates for all traffic patterns under Client-Server traffic model with 4 threads and no-bypass operation

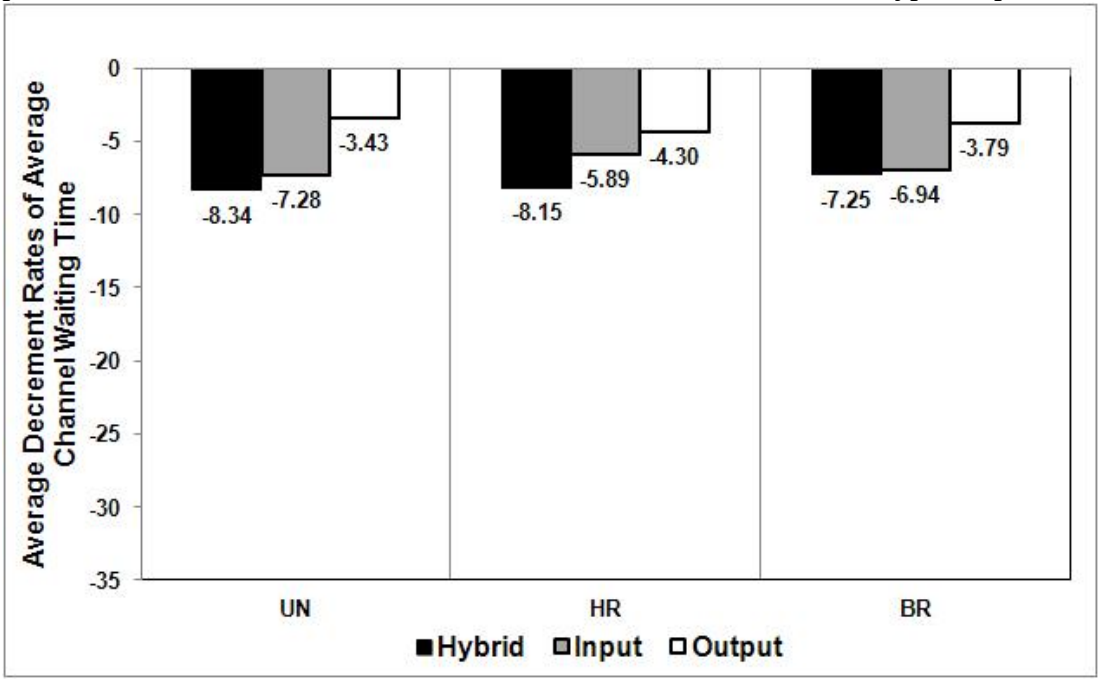

Figure 5.51.b. The average rate of channel waiting time decrement rates for all traffic patterns under Client-Server traffic model with 4 threads and no-bypass operation

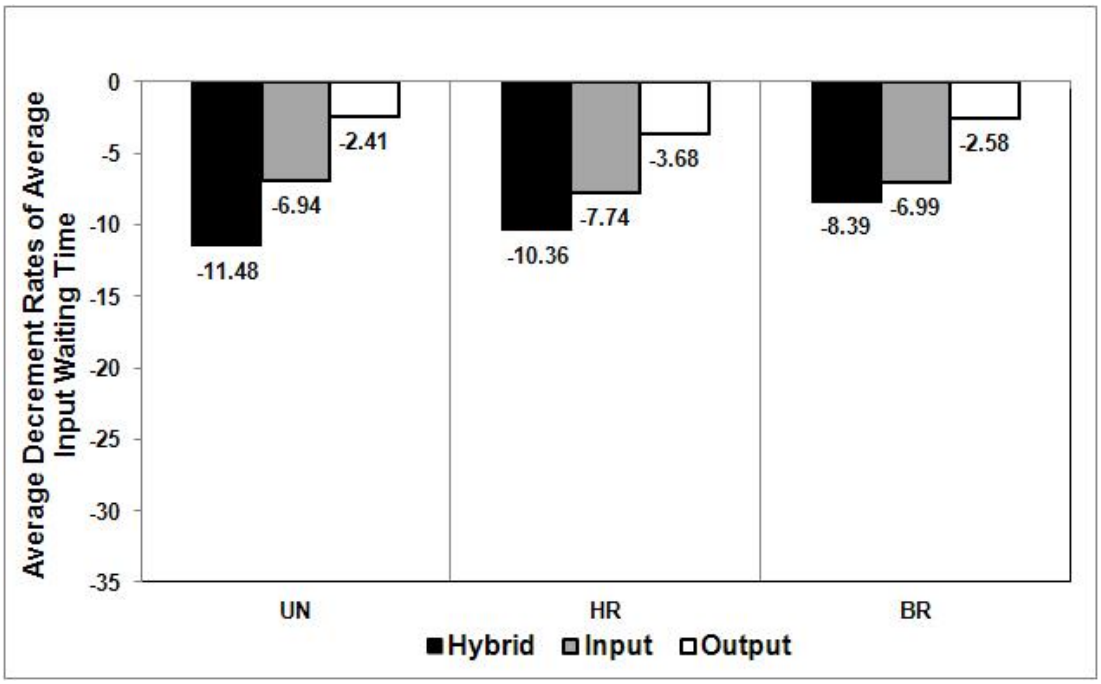

Figure 5.51.c. The average rate of input waiting time decrement rates for all traffic patterns under Client-Server traffic model with 4 threads and no-bypass operation

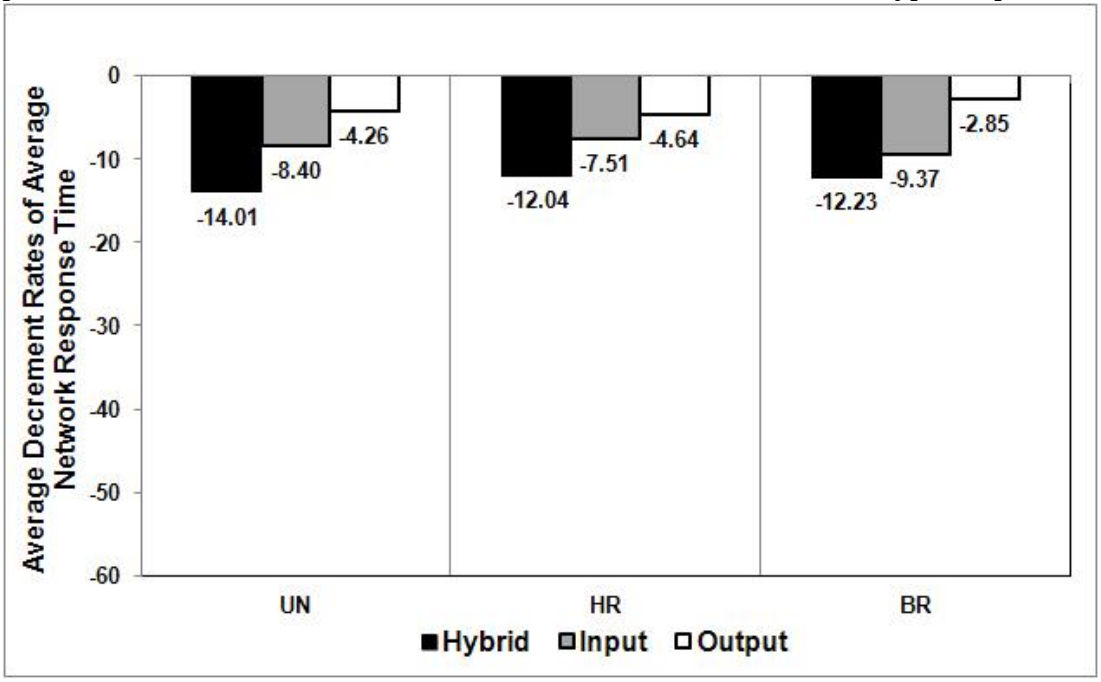

Figure 5.51.d. The average rate of network response time decrement rates for all traffic patterns under Client-Server traffic model with 4 threads and no-bypass operation

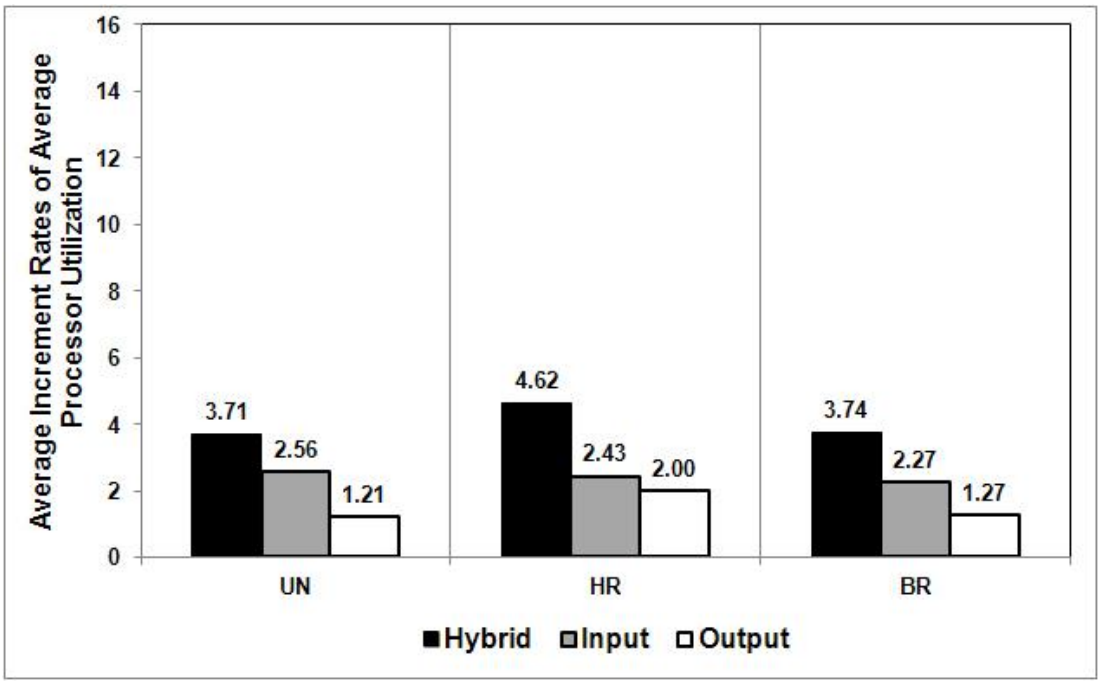

Figure 5.52.a. The average rate of processor utilization increment rates for all traffic patterns under Client-Server traffic model with 8 threads and no-bypass operation

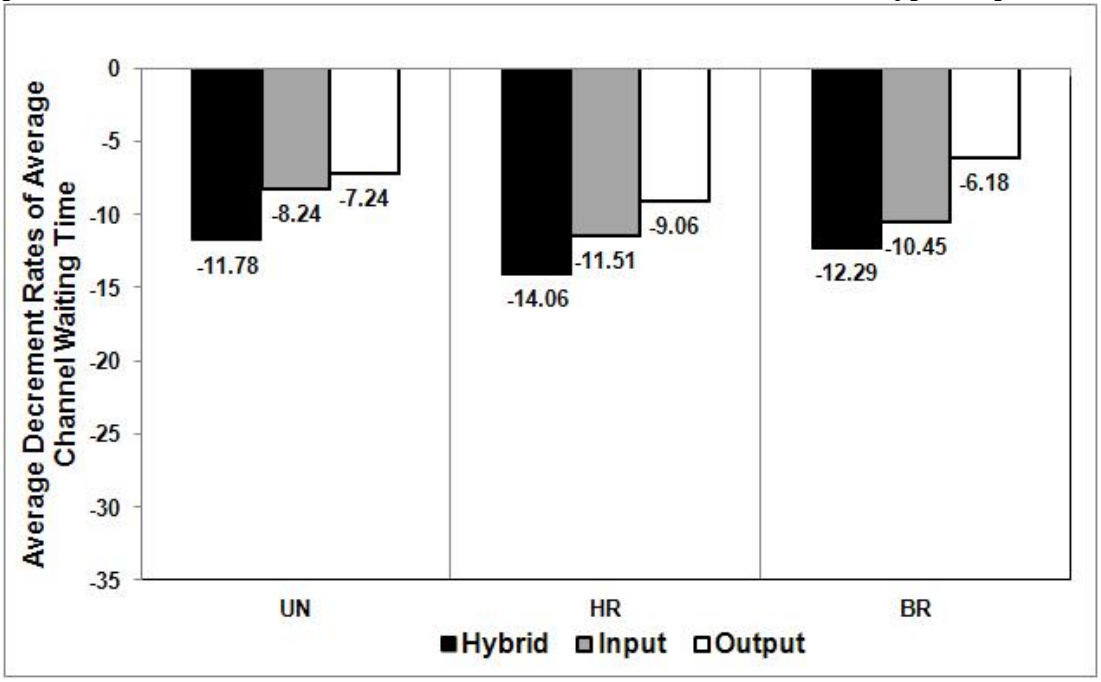

Figure 5.52.b. The average rate of channel waiting time decrement rates for all traffic patterns under Client-Server traffic model with 8 threads and no-bypass operation

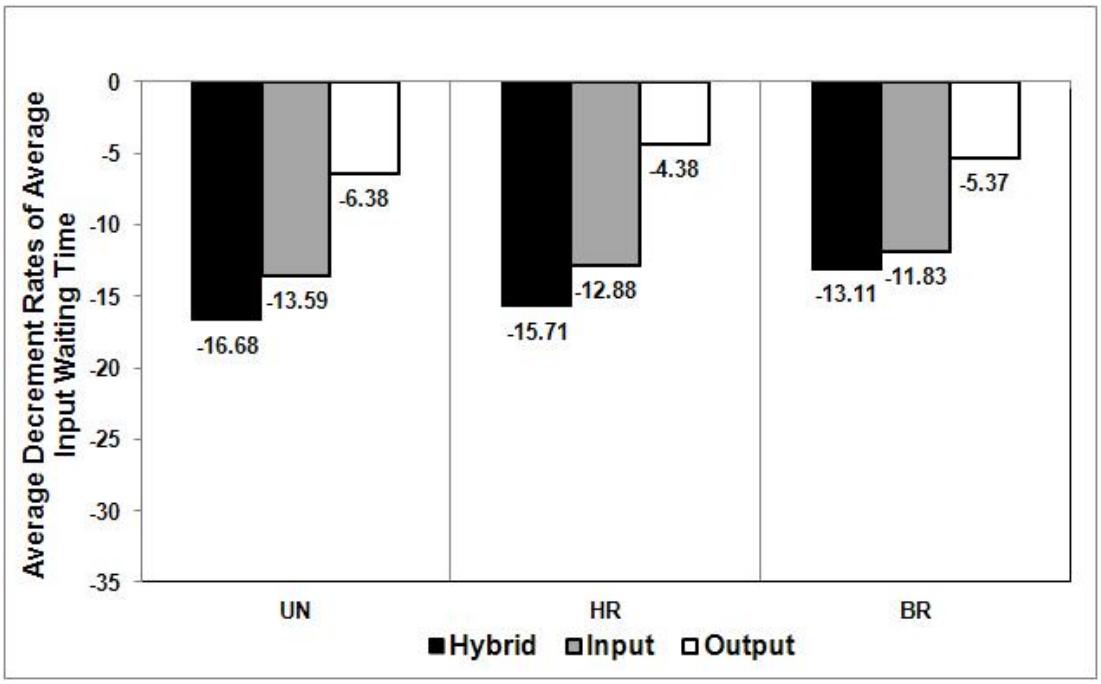

Figure 5.52.c. The average rate of input waiting time decrement rates for all traffic patterns under Client-Server traffic model with 8threads and no-bypass operation

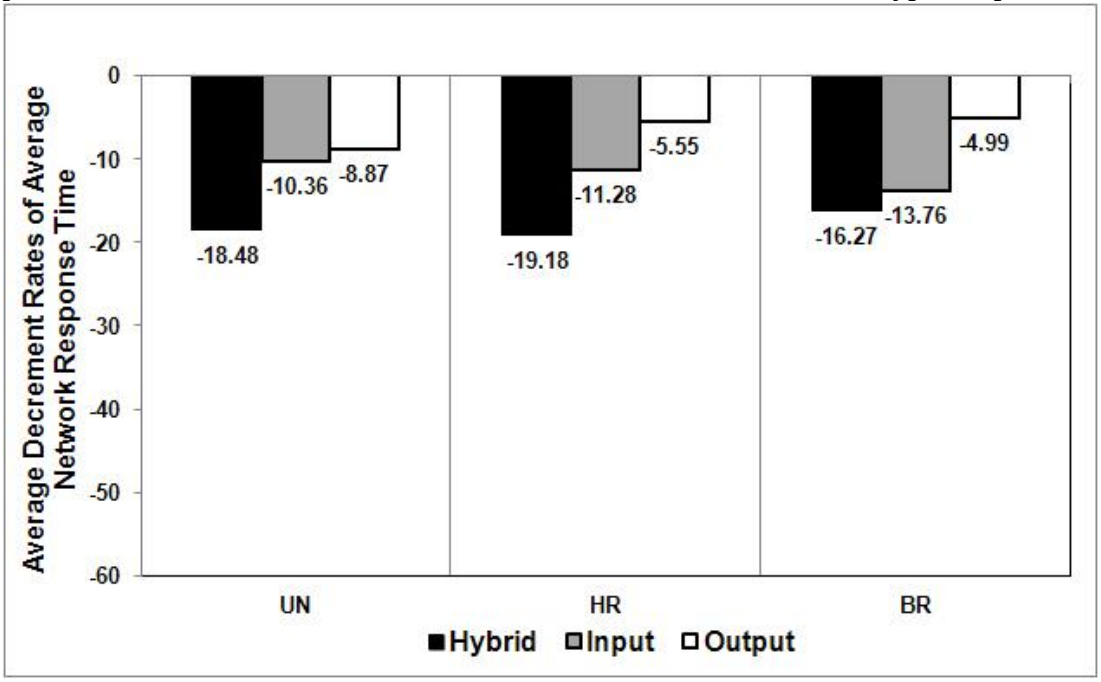

Figure 5.52.d. The average rate of network response time decrement rates for all traffic patterns under Client-Server traffic model with 8 threads and no-bypass operation

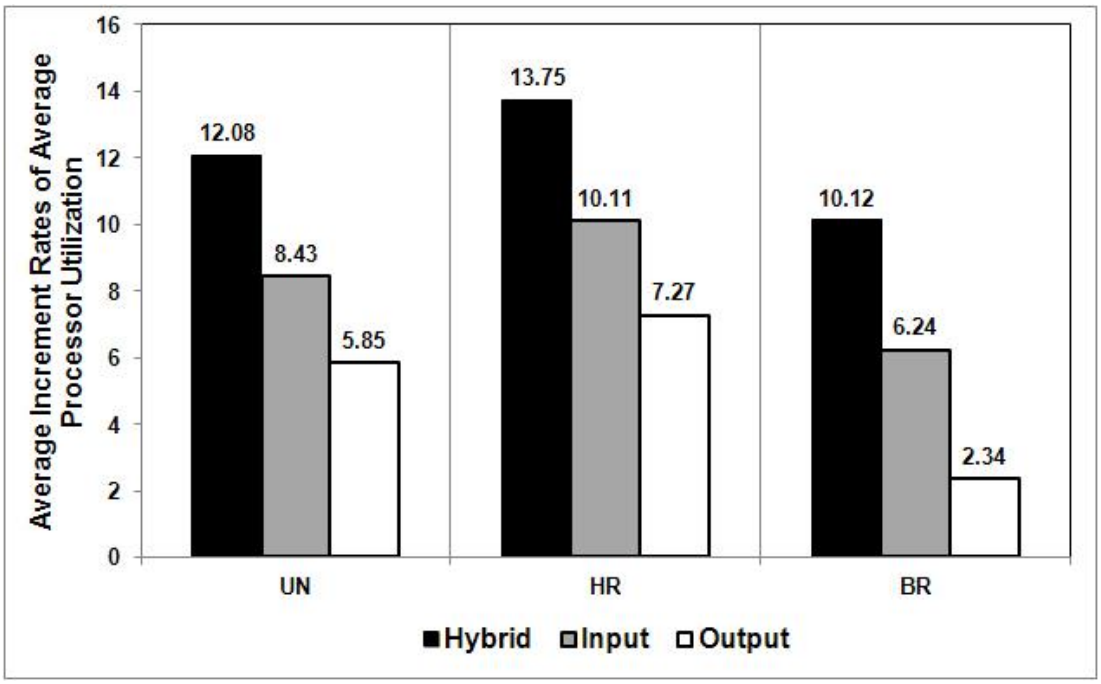

Figure 5.53.a. The average rate of processor utilization increment rates for all traffic patterns under Asynchronous Message Passing traffic model with 4 threads and bypass operation

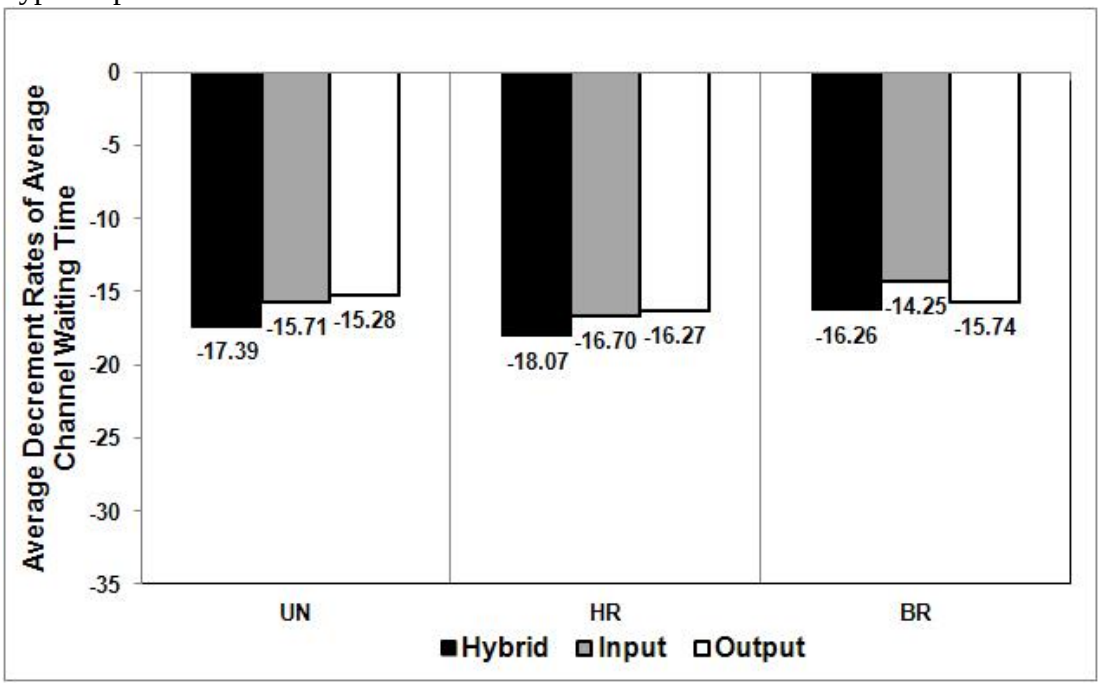

Figure 5.53.b. The average rate of channel waiting time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 4 threads and bypass operation

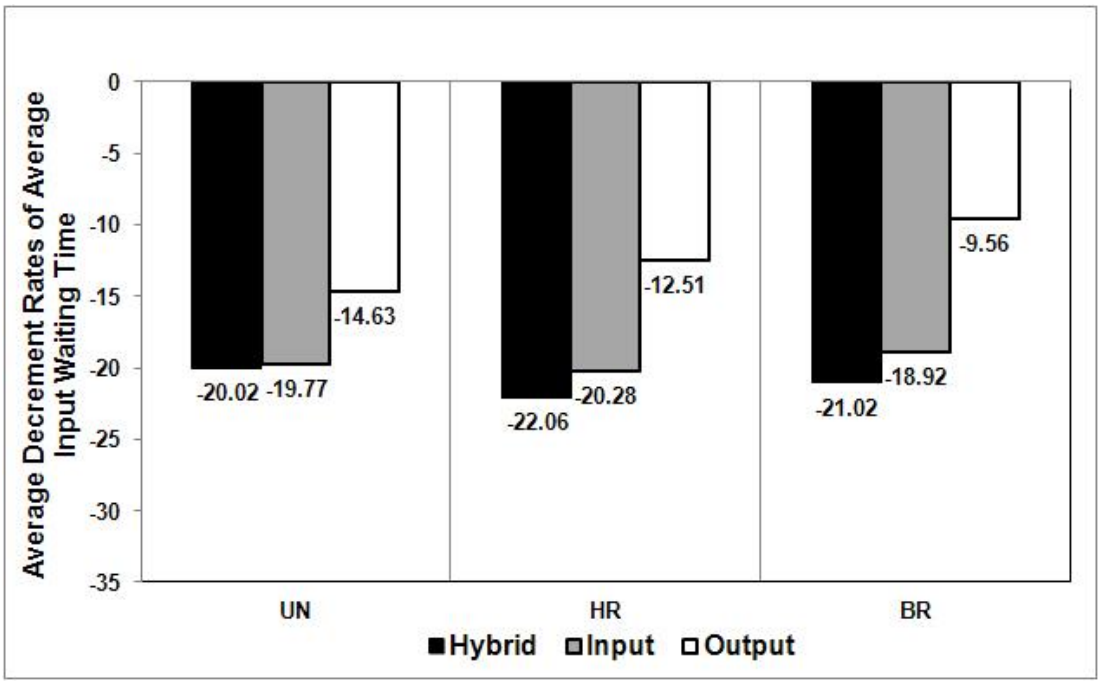

Figure 5.53.c. The average rate of input waiting time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 4 threads and bypass operation

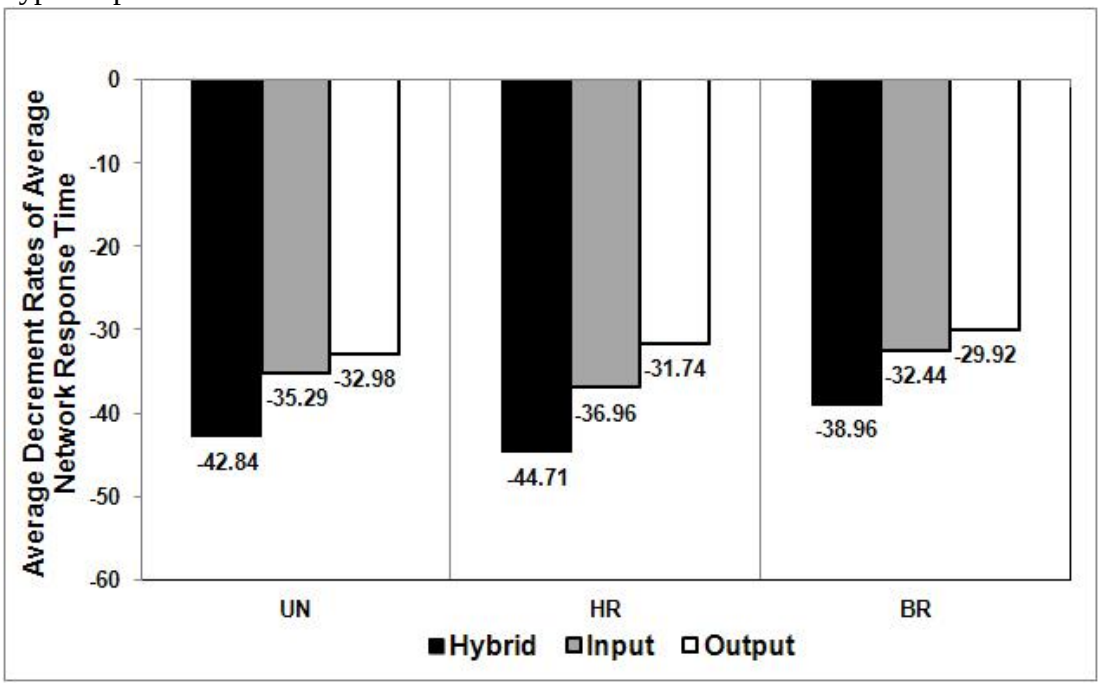

Figure 5.53.d. The average rate of network response time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 4 threads and bypass operation

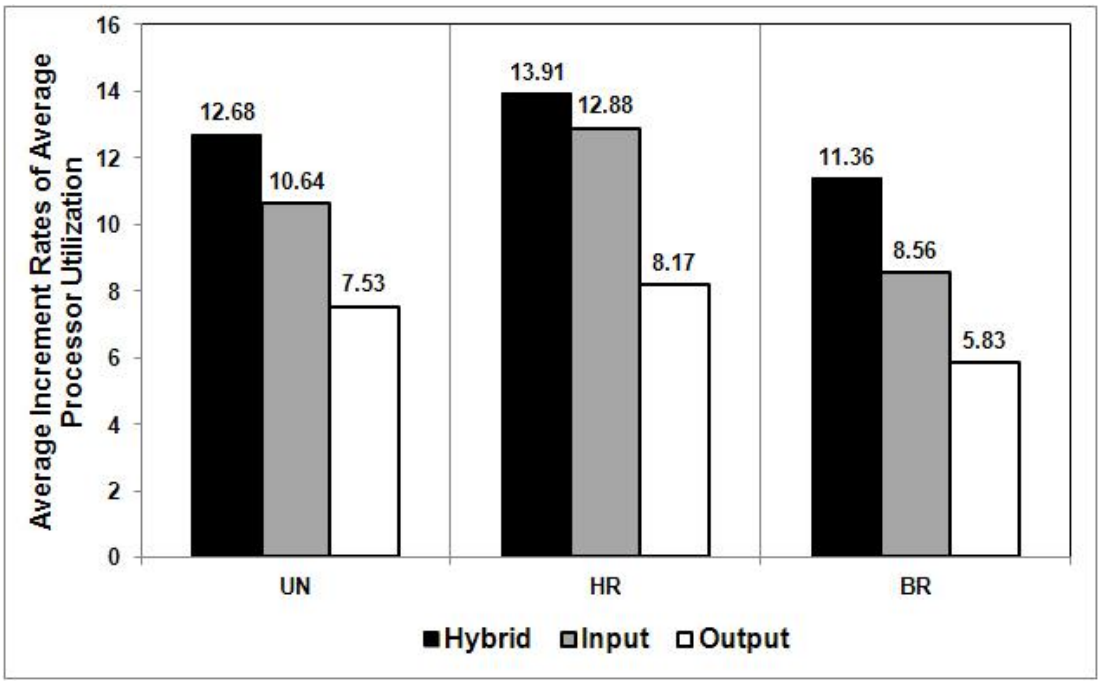

Figure 5.54.a. The average rate of processor utilization increment rates for all traffic patterns under Asynchronous Message Passing traffic model with 8 threads and bypass operation

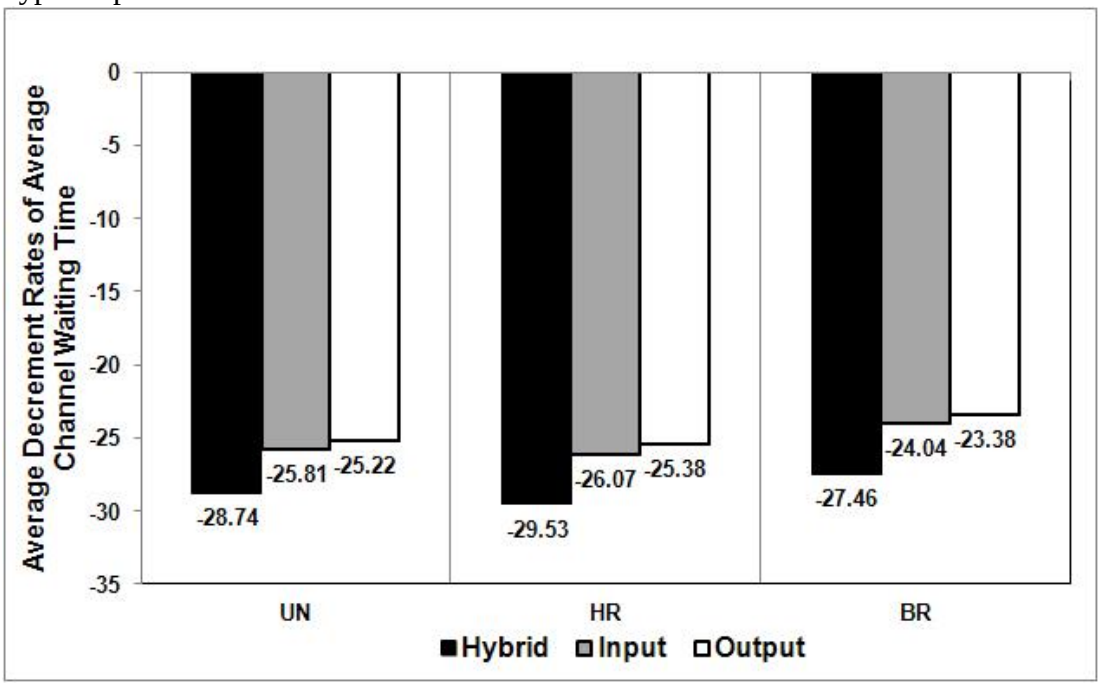

Figure 5.54.b. The average rate of channel waiting time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 8 threads and bypass operation

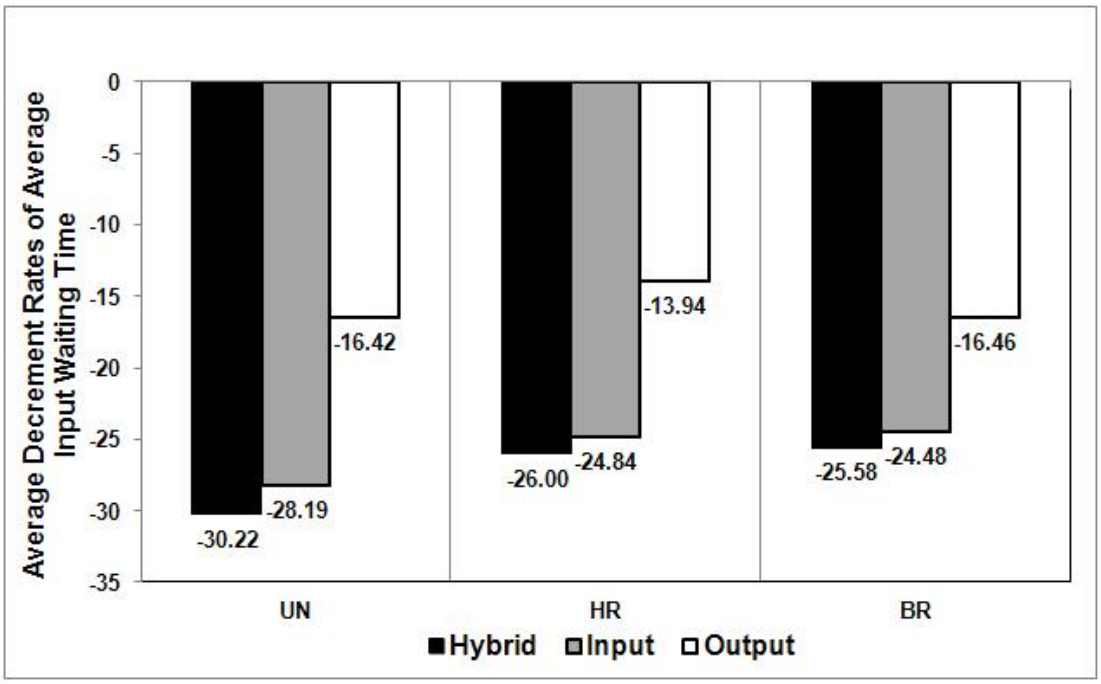

Figure 5.54.c. The average rate of input waiting time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 8 threads and bypass operation

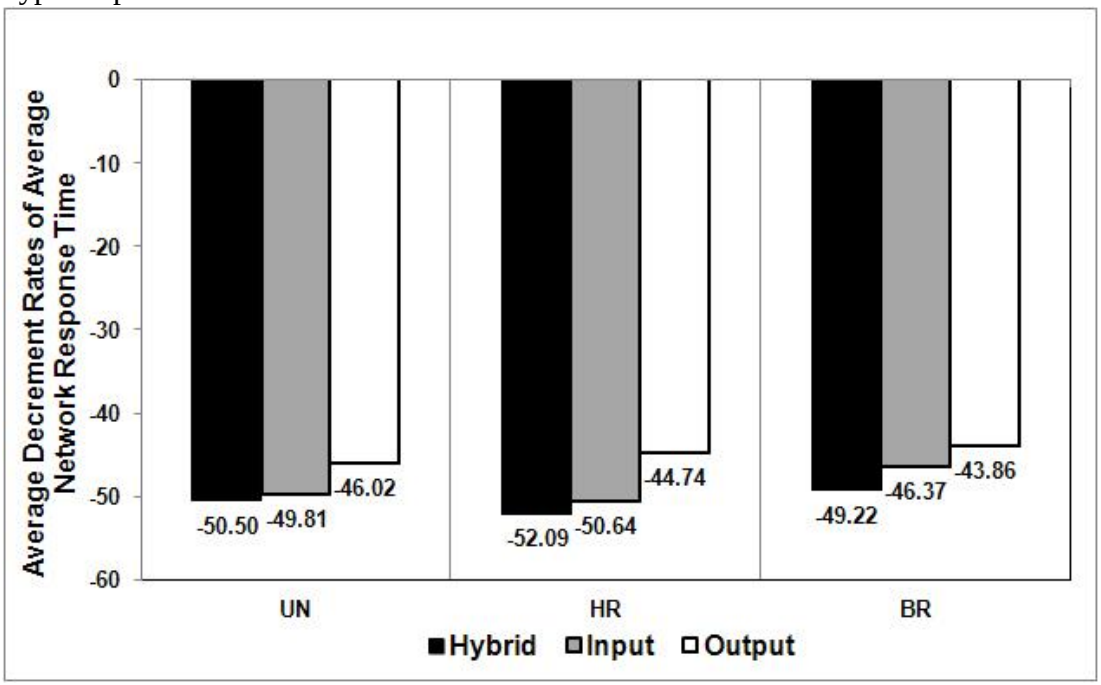

Figure 5.54.d. The average rate of network response time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 8 threads and bypass operation

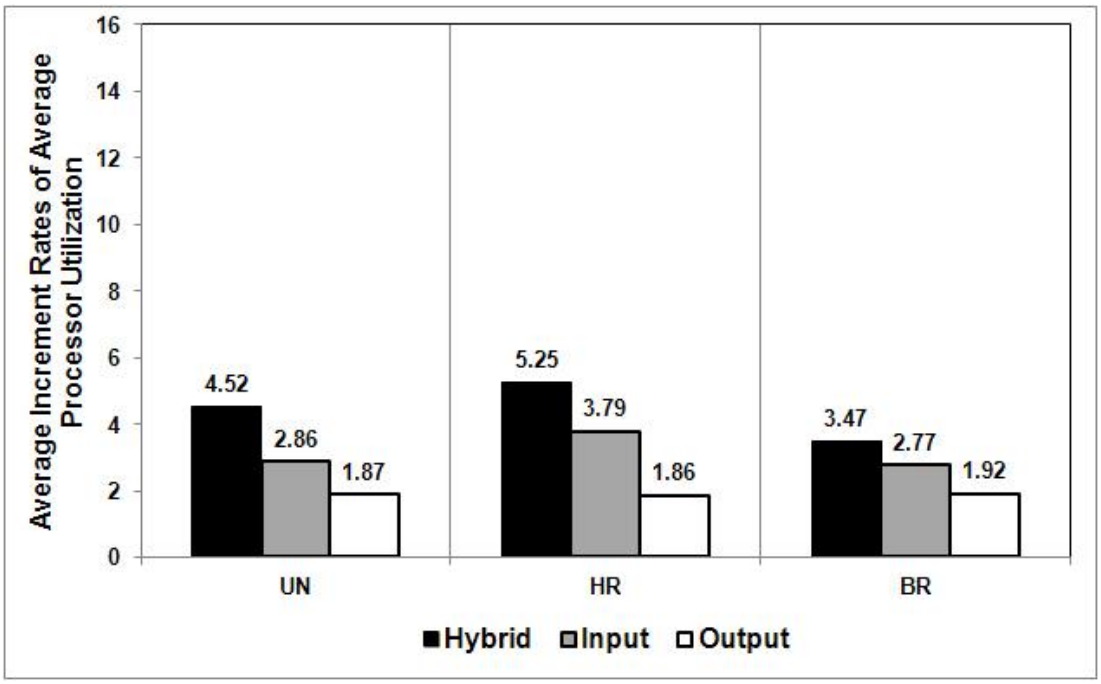

Figure 5.55.a. The average rate of processor utilization increment rates for all traffic patterns under Asynchronous Message Passing traffic model with 4 threads and no-bypass operation

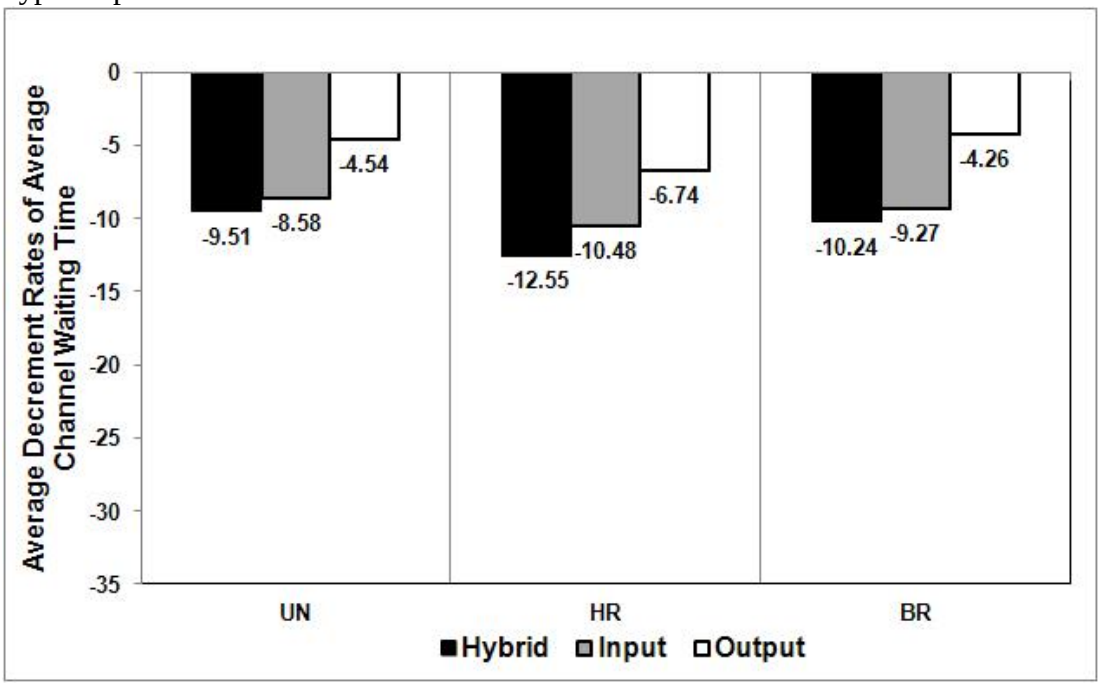

Figure 5.55.b. The average rate of channel waiting time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 4 threads and no-bypass operation

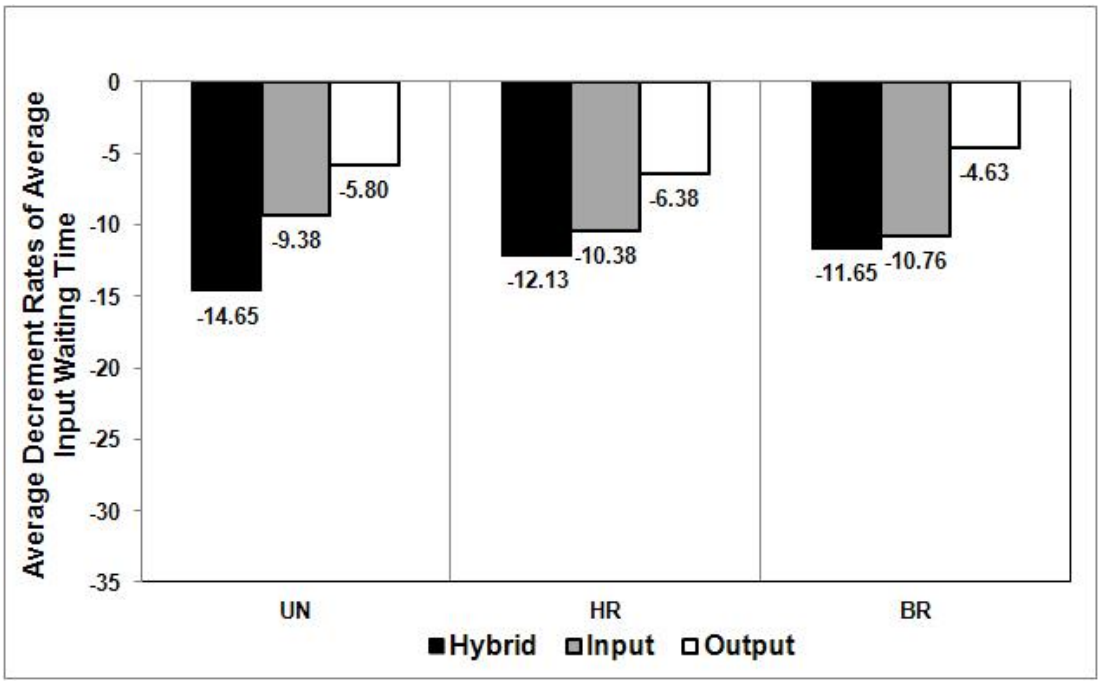

Figure 5.55.c. The average rate of input waiting time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 4 threads and no-bypass operation

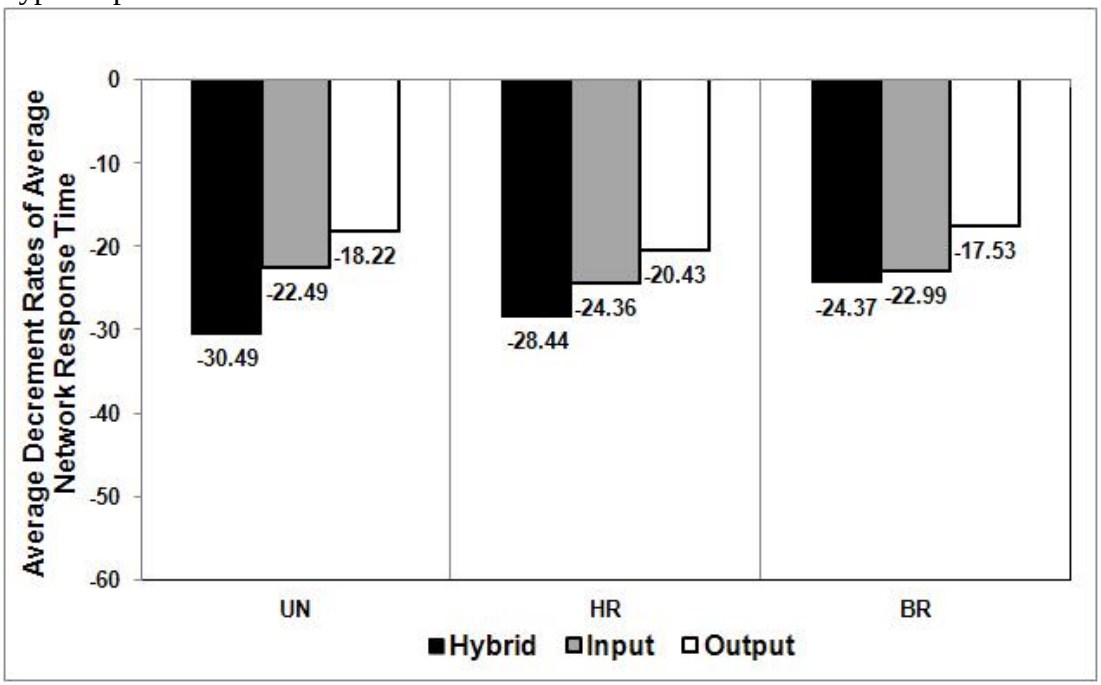

Figure 5.55.d. The average rate of network response time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 4 threads and no-bypass operation

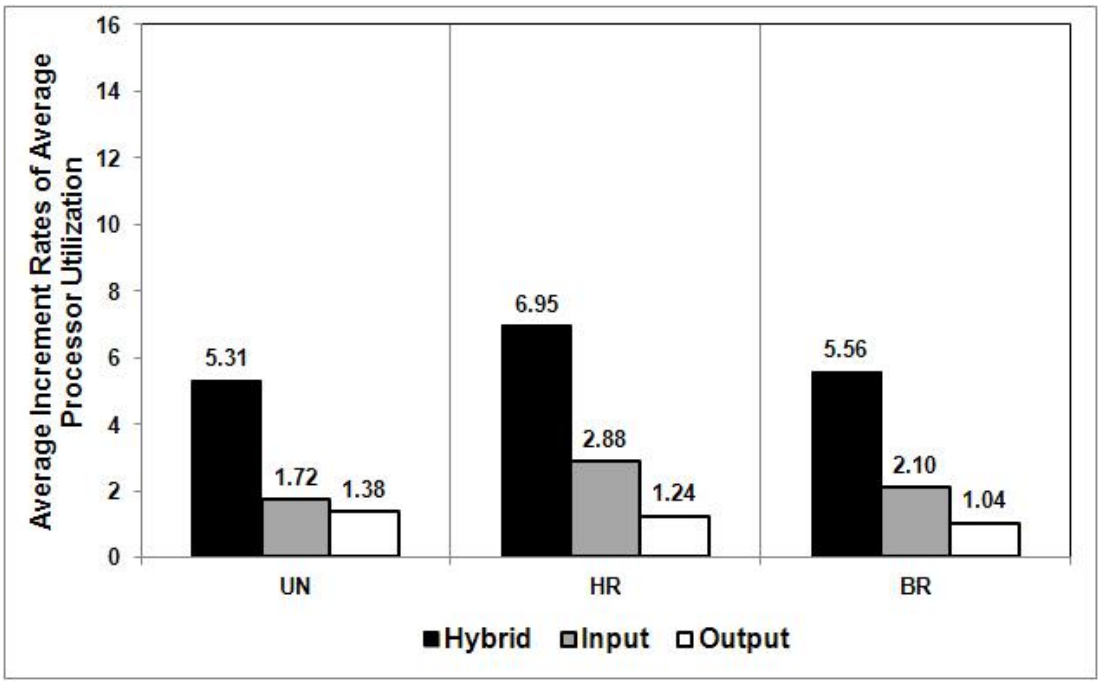

Figure 5.56.a. The average rate of processor utilization increment rates for all traffic patterns under Asynchronous Message Passing traffic model with 8 threads and no-bypass operation

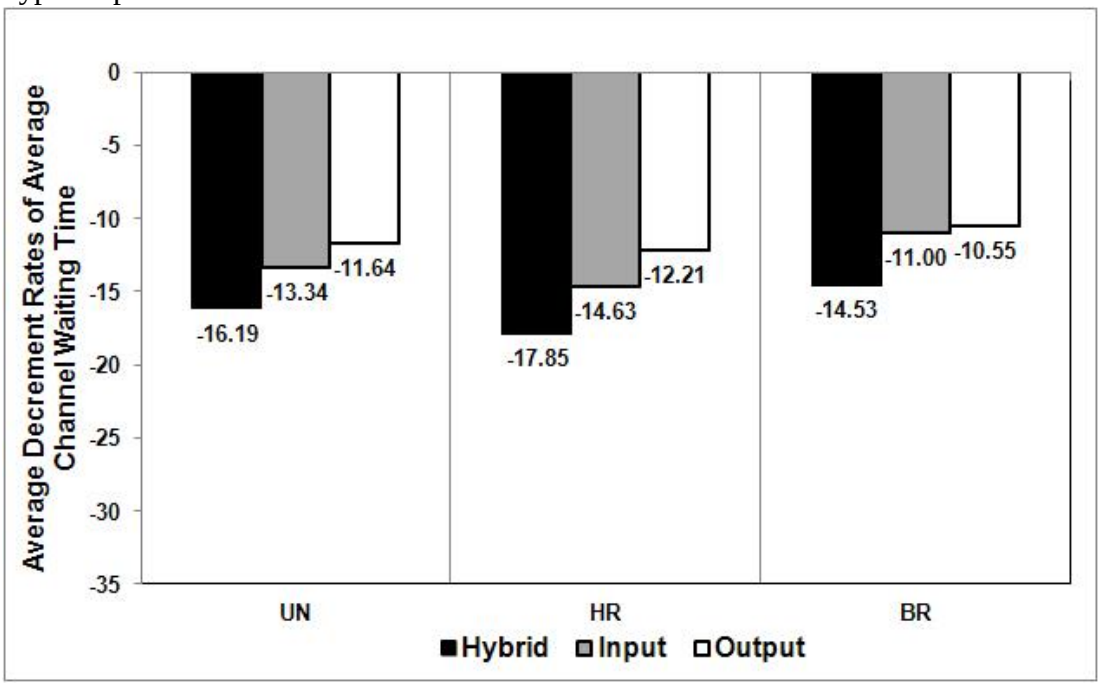

Figure 5.56.b. The average rate of channel waiting time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 8 threads and no-bypass operation

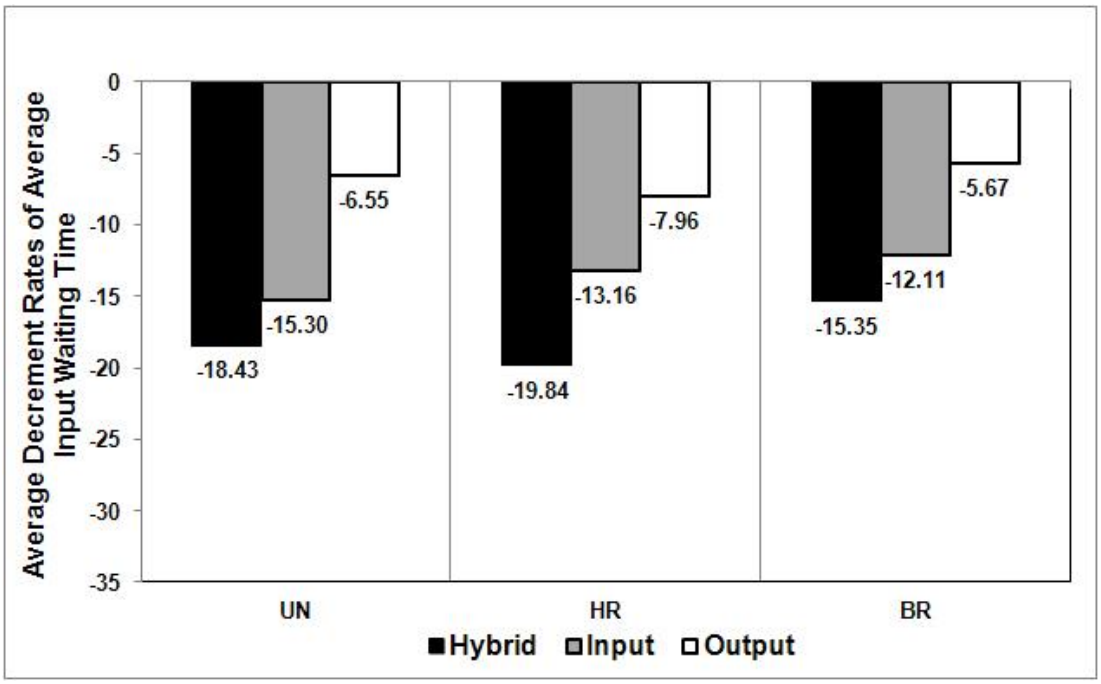

Figure 5.56.c. The average rate of input waiting time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 8 threads and no-bypass operation

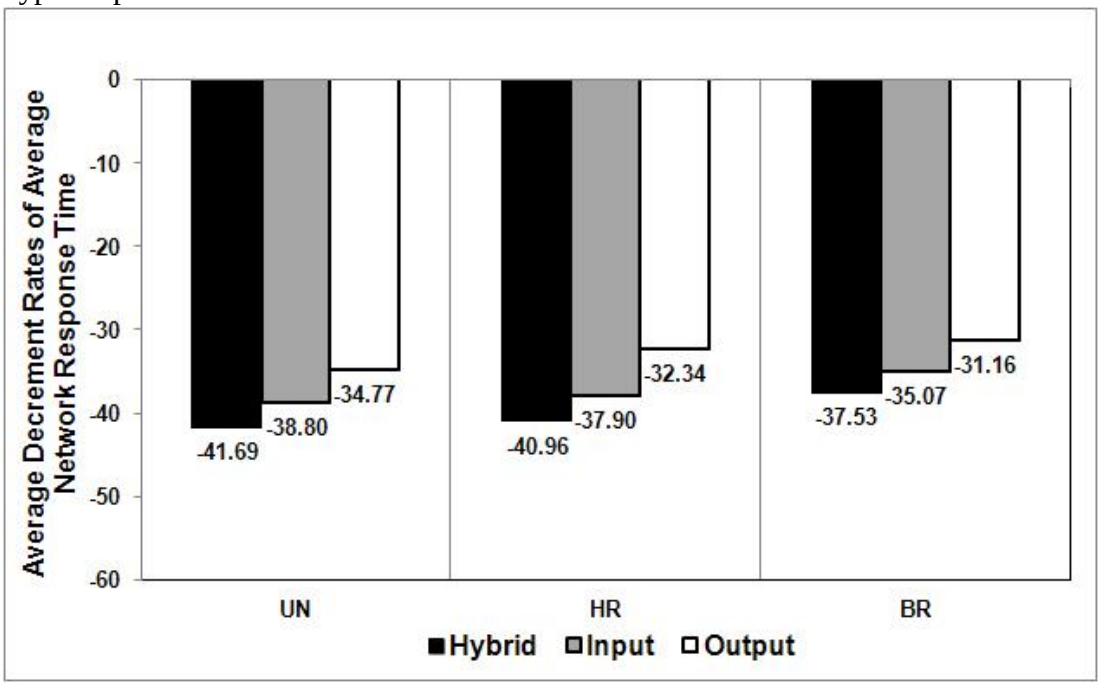

Figure 5.56.d. The average rate of network response time decrement rates for all traffic patterns under Asynchronous Message Passing traffic model with 8 threads and no-bypass operation