Transactions D:
Computer Science & Engineering and
Electrical Engineering
Vol. 16, No. 2, pp. 104–118
© Sharif University of Technology, December 2009

SCIENTIA
IRANICA

On-Chip Interconnection Network with an Efficient Parallel Buffer Structure and Generic Traffic Model

J.H. Bahn[1] and N. Bagherzadeh[2,*]

Abstract. *In this paper, we present two important topics indirectly related to the design and simulated analysis of Network-on-Chip (NoC) architectures. In order to enhance the performance of the baseline router to achieve maximum throughput,a new parallel buffer architecture and its management scheme are introduced. By adopting an adjustable architecture that integrates a parallel buffer with each incoming port, the design complexity and its utilization can be optimized. By utilizing simulation-based performance evaluation and comparison with previous NoC architectures, its efficiency and superiority are proven. One of the key areas of research addressed in this work is to find more realistic traffic models in order to properly test the buffer management schemes proposed in this work. Therefore, we introduce a generic traffic model for on-chip interconnection networks that is superior to previous techniques for NoC architectural performance evaluation. Our traffic model is based on three empirically-derived statistical characteristics using temporal and spatial distributions. With captured parameters, accurate traffic patterns can be generated recursively to show similar statistical characteristics of the observed on-chip networks.*

Keywords: *Network-on-Chip; Virtual channel; Parallel buffer; Router; Generic traffic mode.*

INTRODUCTION

In designing Network-on-Chip (NoC) systems, there are several issues to be considered, such as topology, routing algorithm, performance, latency and complexity. All these factors are taken into account when the design of a NoC have developed better performance routing techniques using oblivious/deterministic or adaptive routing algorithms [1-6]. In addition, adoption of the Virtual Channel (VC) has been prevailing because of its versatility. By adding virtual channels and proper utilization of their channels, deadlock-freedom can be easily accomplished. Network throughput can be increased by dividing the buffer storage associated with each network channel into several virtual channels [4]. By proper control of virtual channels, network flow control can be easily implemented [7]. Also, to increase the fault tolerance in a network, the concept of a virtual channel has been utilized [8,9]. However, in order to maximize utilization, allocation of virtual channels is a critical issue in designing routing algorithms [10,11].

In order to evaluate the performance of either these routing algorithms or their routers, including implementations, many researchers have used conventional traffic patterns [1,12] or some limited number of real traffic traces. Even though these static traffic patterns exhibit similar patterns of some particular applications, there is a fundamental limit in covering complete traffic patterns of real applications. For this reason, some researchers have used real traffic patterns extracted from real applications to evaluate the performance of their proposed routing, algorithm or router based on more practical bench marks [13,14].

In this paper, we proposed a base NoC architecture adopting a minimal adaptive routing algorithm with near-optimal performance and feasible design complexity [6]. Based on this NoC architecture, a new routing-independent parallel buffer structure and its management scheme are discussed as a viable alternative to the VC approach. As a result, the channel utilization and maximum throughput in performance are improved. In order to utilize a realistic traffic

1. *Qualcomm CDMA Technologies, Qualcomm Inc., San Diego, CA, 92121-1714, USA.*
2. *Department of Electrical Engineering and Computer Science, University of California, Irvine, CA, 92697-2625, USA.*
*. *Corresponding author. E-mail: nader@uci.edu*

Received 30 January 2009; received in revised form 21 September 2009; accepted 3 November 2009

model for our proposed architectural entrenchment to the NoC router, a generic traffic model for NoC environments is described. The proposed model is based on the spatial/temporal profile of traffic using three statistical parameters.

The organization of this paper is as follows. In the next section, a brief introduction of base NoC architecture, adopting a minimal adaptive routing algorithm, is provided. While explaining the proposed Parallel Buffer (PB) structure and its management scheme, the enhanced NoC architecture including these parallel buffers will be introduced. In order to prove its benefit, several simulation-based evaluation results and a comparison with the base NoC architecture will be provided. In the following section, an overview of our traffic model with three different statistical components for NoC is provided and the details of each component are presented. The overall procedure of generating traffic patterns with the given parameters is described and the accuracy of the proposed traffic model is validated by comparing it with real traffic traces. Finally, some conclusions will be drawn.

BASE NETWORK-ON-CHIP (NOC) ARCHITECTURE

We propose an adaptive routing algorithm and the baseline architecture for a flexible on-chip interconnection [6]. It adopts a wormhole switching technique and its routing algorithm is livelock-/deadlock- free in a 2-D mesh. Specifically, to eliminate the deadlock situation and simplify the routing algorithm, two disjoint vertical channels are provided instead of using virtual channels. The use of a vertical channel is limited by the direction of delivered data. That is, each vertical channel is denoted by *N1/S1* for east-bounded and *N2/S2* for west-bounded packets, respectively. Also, the data from the internal Processing Element (PE) or Execution Unit (EU) connected with the router uses separate injection ports, *IntL-in* and *IntR-in*, depending on its direction of target node. As a result, available routing ports are grouped as {*W-in, N1, E-out, S1, IntR-in*} and {*E-in, N2, W-out, S2, IntL-in*} where *N1/N2* or *N2/S2* represents incoming/outgoing ports simultaneously (-*in* an incoming port, and -*out* an outgoing port for the given channel, respectively).

For each set of ports, the routing decision is independently performed. For instance, in the set of east-bounded ports, i.e. {*W-in, N1, E-out, S1, IntR-in*}, incoming ports are routed to each output port depending on port priority as shown in Table 1. There are two different levels of priority on incoming ports and outgoing ports, respectively. The priority on outgoing ports is given in the order of *N1-out*, *E-out*, *S1-out*, and *Int-out*. Thus, it is organized by starting from the north in a clockwise direction to the port connected to EU, *Int-out*, having the lowest priority. The other priority on incoming ports is differently assigned, depending on the given outgoing port. For the given outgoing port, the priority on incoming ports is given in the order of a clockwise direction, starting from the incoming port next to the given outgoing port, if the incoming port has a deliverable flit to the given outgoing port. If any incoming port is routed to an outgoing port with higher priority, that port is not considered in the routing decision for outgoing ports with lower priority. Algorithm 1 summarizes a detailed procedure of the routing decision. Based on these operations, the micro-architecture of either a *Right* or *Left* router is designed as Figure 1a.

ENHANCED NETWORK-ON-CHIP (NOC) ARCHITECTURE WITH PARALLEL BUFFERS

In order to enhance the performance of base NoC architecture, an approach similar to the parallel buffer technique of virtual channels is selected as shown in Figure 1b. Instead of using dedicated buffers for each port, parallel buffers with a small depth queue or FIFO are added in front of each incoming port. The difference from previous approaches with virtual channels is a routing-independent parallel buffer structure and its

Table 1. Priority assignment on incoming/outgoing ports.

Outgoing Ports	Incoming Ports
N1-out	*S1-in*, *W-in*, *IntR-in*
E-out	*S1-in*, *W-in*, *N1-in*, *IntR-in*
S1-out	*W-in*, *N1-in*, *IntR-in*
N2-out	*E-in*, *S2-in*, *IntL-in*
S2-out	*N2-in*, *E-in*, *IntL-in*
W-out	*N2-in*, *E-in*, *S2-in*, *IntL-in*
Int-out	*N1-in*, *N2-in*, *E-in*, *S1-in*, *S2-in*, *W-in*

```
local variable: MappediPortFlag[ ] contains whether iPort[ ] is routed or not
initialize MappediPortFlag[ ] to 0
/* searching output ports from N1-out to Int-out clockwise */
for oIndex $\leftarrow$ N1-out to Int-out do
    if oPort[oIndex].Status is active then
        /* if the selected output port is occupied for transmission, then maintain the routing until the last flit is delivered */
        traverse data from iPort[oPort[oIndex].RoutediPort] to oPort[oIndex]
        decrease oPort[oIndex].RestFlits by 1
        MappediPortFlag[oPort[oIndex].RoutediPort] set to 1
        if oPort[oIndex].RestFlits is zero then
            oPort[oIndex].Status set to inactive
        end if
    else
        /* if the selected output port is free, then search the available input port routable to the output port */
        for iIndex $\leftarrow$ value from left to right at the row of the corresponding outgoing port in Table 1 do
            if MappediPortFlag[iIndex] is 0 then
                if iPort[iIndex].HeaderReady is true then
                    traverse data from iPort[iIndex] to oPort[oIndex]
                    /* register the routing information for incoming port such as length of flits, mapped incoming port, and
                    active status to outgoing port */
                    extract the length of flits and set oPort[oIndex].RestFlits
                    oPort[oIndex].RoutediPort $\leftarrow$ iIndex
                    MappediPortFlag[iIndex] $\leftarrow$ 1
                end if
            end if
        end for
    end if
end for
```

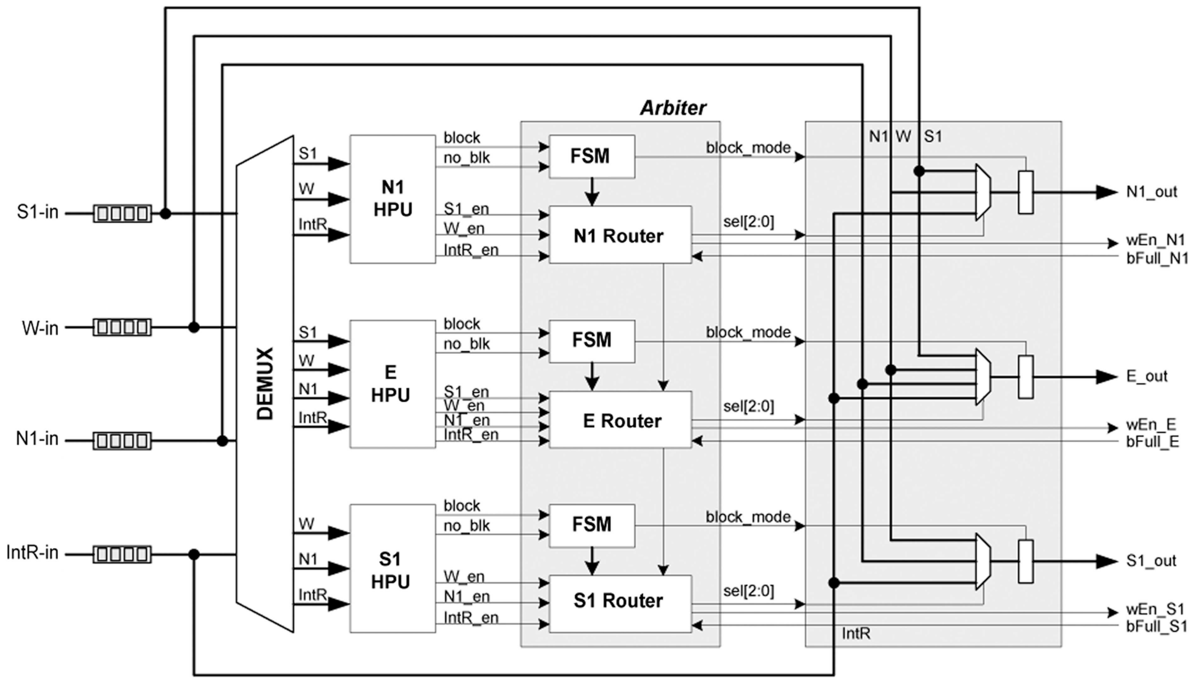
Algorithm 1. Pseudo routing algorithm for base router.

efficient management scheme, which will be described in detail.

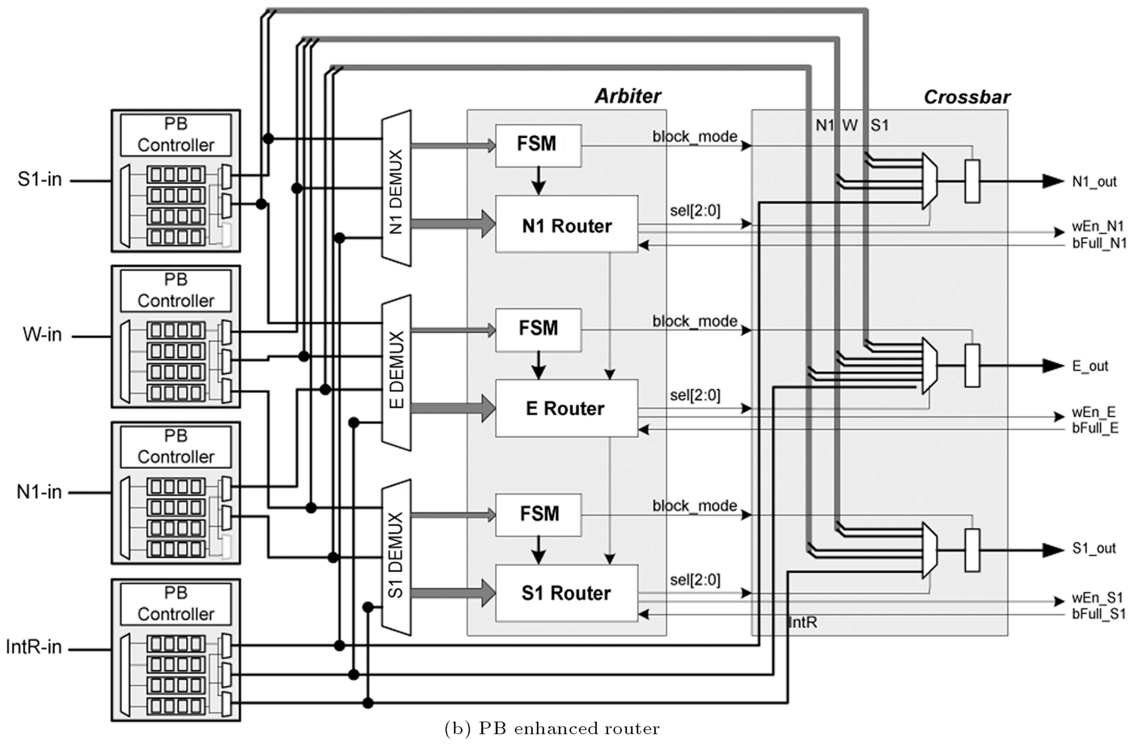
Figure 2 shows a detailed parallel buffer structure applied in the enhanced NoC architecture. To maximize the utilization of channels, multiple outputs from a parallel buffer for each forwarded direction are provided to the routers. By virtue of 2-D mesh topology, the maximum number of forwarded directions is 3. For each forwarded direction, a maximum of 2 outputs from a parallel buffer are provided. The following example explains how to extract the maximum number of outputs from a parallel buffer for each output port.

Let's assume that a parallel buffer with 8 FIFOs at each incoming port is allocated and all FIFOs that are only in the parallel buffer at the incoming port *W-in* contain packets. Also, the packets occupying FIFOs in the parallel buffer at *W-in* port arrived at different times. The destination of each packet occupying each FIFO in the parallel buffer is *E*, *NE*, *N*, *SE*, *SE*, *S*, *NE*, and *S* in the order of $\{PB_0$, PB_1, PB_2, PB_3, PB_4, PB_5, PB_6, $PB_7\}$ where PB_i represents the ith FIFO in the given parallel buffer. Let's assume the order of arrival time for each packet is $\{PB_1$, PB_3, PB_0, PB_2, PB_4, PB_6, PB_5, $PB_7\}$. If each FIFO in the parallel buffer is grouped in the order of arrival time and its available routing direction, the resultant groups of FIFOs are $\{PB_1$, PB_2, $PB_6\}$ for *N-out*, $\{PB_1$, PB_3, PB_0, PB_4, $PB_6\}$ for *E-out*, and $\{PB_3$, PB_4, PB_5, $PB_7\}$ for *S-out*. Because no incoming ports other than *W-in* have deliverable data, the routing decision is performed only on port *W-in*. According to the described priority in Table 1, the outgoing port *N-out* is the first one to be considered. For *N-out* outgoing port, the packet stored in PB_1 will be selected. For *E-out* outgoing port, the packet stored in PB_3 will be chosen because PB_1 is already occupied by the outgoing port *N-out* with a higher priority. Finally, to *S-out* outgoing port, the packet stored in PB_4 will be forwarded because the earlier packet in PB_3 is already reserved for *E-out*. Therefore, instead of searching all the entries in each group, the first 2 entries are sufficient for checking the available incoming packet for the routing decision. Algorithm 2 summarizes a detail routing procedure for the enhanced PB router.

Differing from the conventional VC approaches, the operation of the proposed parallel buffer is no longer dependent on neighboring routers. By autonomous management of a parallel buffer, depending on outgoing port read requests, the parallel buffer can be simply assumed as a single FIFO. That is, from the previous neighboring router, the parallel buffer is recognized as a single FIFO with ordinary interfaces such as `bFull` (buffer fullness) or `wEn` (write enable). Therefore, the only task of this parallel buffer is to store the incoming packets and manage their occupancy with respect to their destination and read operations depending on routing decision. In order to manage the empty FIFOs in the given parallel buffer, as illustrated in Figure 2, simple logic circuits are added in *PB_in Control Block*. With given empty signals from all input FIFOs, one of the empty FIFO indices is chosen, which controls the path of storing incoming *flit* into



(a) Base router



(b) PB enhanced router

Figure 1. Micro-architecture of base and PB enhanced router.

the corresponding FIFO. Simultaneously, to control the incoming packet in *flit* the Header Parsing Unit (HPU) and the associated control unit (IOPCF) are needed.

In the proposed parallel buffer structure, every two outputs among the allocated FIFOs in the parallel buffer are chosen and forwarded to the inputs of routing decision logic for the corresponding outgoing port. The parallel buffer controller manages the history of arrival packets and their FIFO locations in the parallel buffer, and groups of in-use FIFOs based on its outgoing direction; HPU is utilized for this purpose. By moving the location of HPU, which

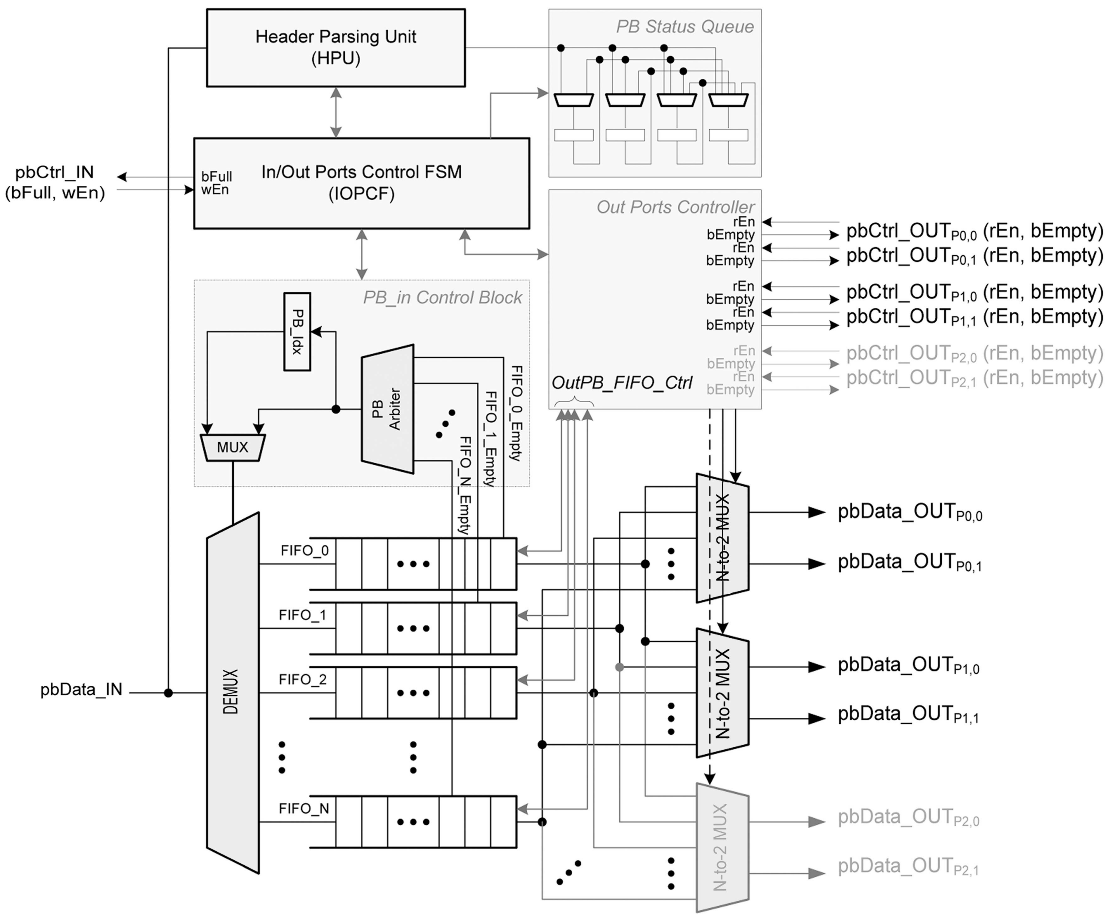


Figure 2. Proposed parallel buffer structure.

```
1:  local variable: $MappediPortFlag[\,][\,]$ contains whether $iPort[\,][\,]$ is routed or not
2:  initialize $MappediPortFlag[\,]$ to 0
3:  /* searching output ports from $N1\text{-}out$ to $Int\text{-}out$ clockwise similarly with base router */
4:  **for** $oIndex \leftarrow N1\text{-}out$ to $Int\text{-}out$ **do**
5:      **if** $oPort[oIndex].Status$ is active **then**
6:          traverse data from $iPort[oPort[oIndex].\ RoutediPort][oPort[oIndex].RoutediPortPB]$ to $oPort[oIndex]$
7:          decrease $oPort[oIndex].RestFlits$ by 1
8:          $MappediPortFlag[oPort[oIndex].\ RoutediPort][oPort[oIndex].RoutediPortPB]$ set to 1
9:          **if** $oPort[oIndex].RestFlits$ is zero **then**
10:             $oPort[oIndex].Status$ set to inactive
11:         **end if**
12:     **else**
13:         /* if the selected output port is free, then search the available input port routable to the output port */
14:         **for** $iIndex \leftarrow$ value from left to right at the row of the corresponding outgoing port in Table 1 **do**
15:             /* searching the assorted PB queue for the given output port */
16:             **for** $iPBIndex \leftarrow$ value from the order of PB queue **do**
17:                 **if** $MappediPortFlag[iIndex][iPBIndex]$ is 0 **then**
18:                     **if** $iPort[iIndex][iPBIndex].HeaderReady$ is true **then**
19:                         traverse data from $iPort[iIndex][iPBIndex]$ to $oPort[oIndex]$
20:                         /* register the routing information for incoming port such as length of *flits*, mapped incoming port,
                            mapped index of PB queue, and active status to outgoing port */
21:                         extract the length of *flits* and set $oPort[oIndex].RestFlits$
22:                         $oPort[oIndex].RoutediPort \leftarrow iIndex$
23:                         $oPort[oIndex].RoutediPortPB \leftarrow iPBIndex$
24:                         $MappediPortFlag[iIndex][iPBIndex] \leftarrow 1$
25:                     **end if**
26:                 **end if**
27:             **end for**
28:         **end for**
29:     **end if**
30: **end for**
```

Algorithm 2. Pseudo routing algorithm for enhanced PB router.

is originally placed in the router logic as Figure 1a, the critical path for the enhanced router is reduced. Because the performance of FIFOs is much faster than that of the router logic with HPU [6], it results in balancing the workload of each block, with respect to the timing. Therefore, the enhanced NoC architecture obtains a better performance for real implementations.

EVALUATION OF THE PERFORMANCE IN THE ENHANCED NOC ARCHITECTURE

Evaluation Environment

In order to evaluate the performance of the base NoC architecture, a time-accurate simulation model was implemented in SystemC. By comparing with different routing algorithms, its competitive performance has been evaluated. Parallel buffers with an efficient management scheme expect an overall performance increment. The parallel buffers with the proposed management scheme were included in this model, replacing the previous FIFO module approach.

All the network simulations were performed using 100,000 cycles with 4 commonly used traffic patterns, such as uniform random, bit-complement, matrix-transpose traffic and bit-reverse traffic. Two different sizes of 2-D mesh topology based on 4×4 and 8×8 were studied, and the number of FIFOs in the parallel buffer per incoming port was varied. However, the depth of FIFO in the parallel buffer was fixed as 4, and 4-flit long packets were used. For the measurement of throughput and adjusting incoming traffic, the standard interconnection network measurement technique [1] was adopted.

Simulation Results and Their Analysis

SystemC simulations with various traffic patterns and two different network topologies resulted in experimental results of PB enhanced NoC architecture. With these collected data, the plots of average latency vs. offered traffic are drawn in Figures 3 and 4 for 4×4 and 8×8 mesh, respectively. For a 4×4 mesh network, both uniform random and bit-reverse traffic patterns show a notable increase in maximum throughput, approximately 25% and 19%, respectively. In the 8×8 mesh network, uniform random, matrix-transpose and bit-reverse traffic patterns show a noticeable improvement of around 28%, 28% and 18%, respectively. However,

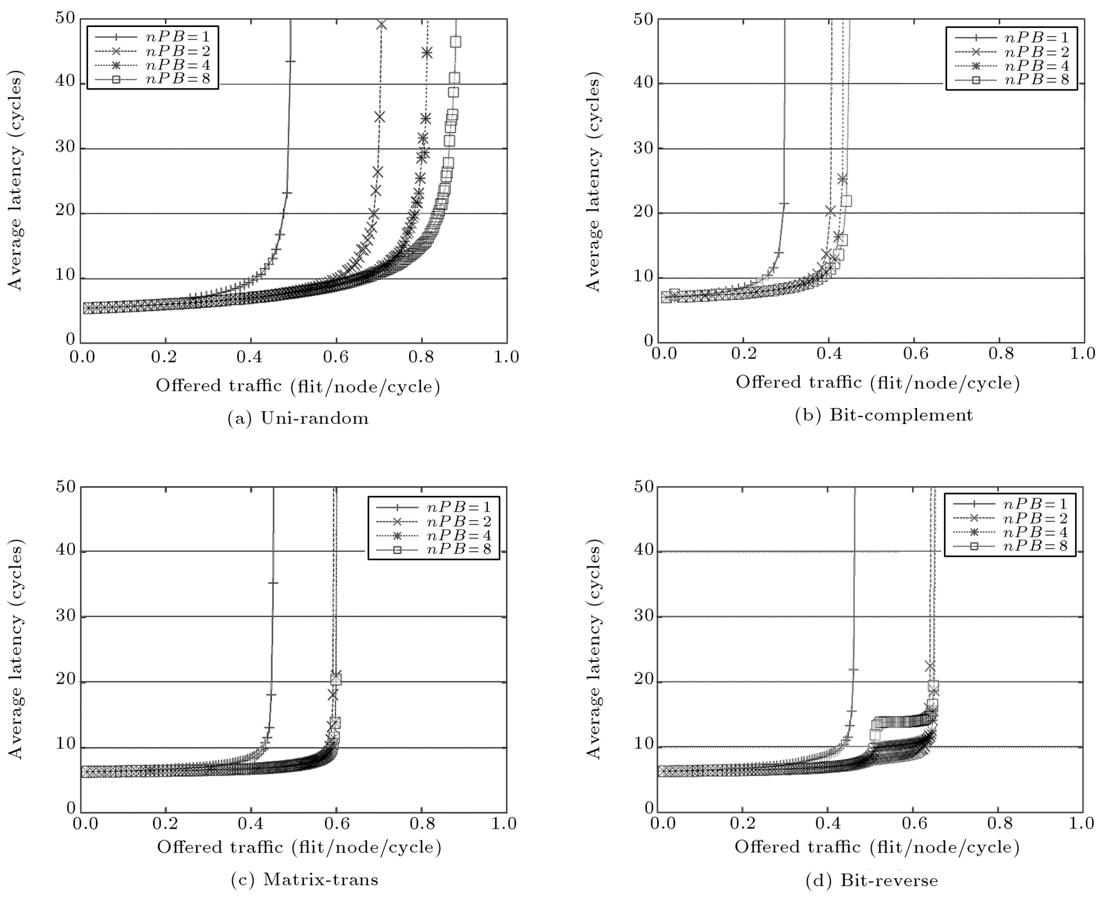


Figure 3. Performance in 4×4 mesh.

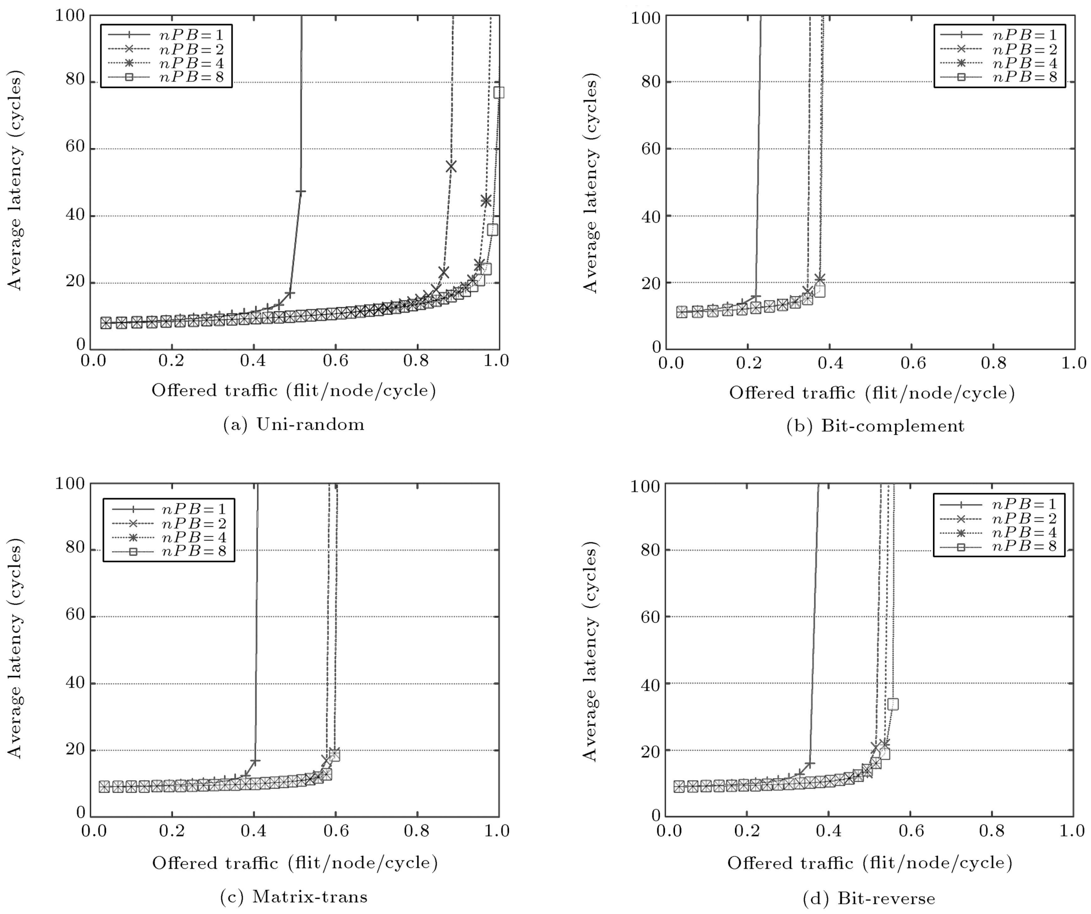


Figure 4. Performance in 8 × 8 mesh.

for bit-complement traffic patterns in 4 × 4 and 8 × 8, the improvement of performance seems to be minor. The bit-complement traffic pattern anomaly is because it has relatively lower flexibility in choosing routing paths from source to destination and most of the traffic patterns concentrate on the central region for a given mesh, resulting in severe routing contention and blocking similar to the analysis in [11]. As the size of 2-D mesh topology increases, the effect of a parallel buffer in improving performance grows, because the size increase provides a much higher degree of flexibility in routing paths.

As shown in [6], the previous base NoC architecture reaches up to 0.4 offered traffic with a uniform random traffic pattern for the 8 × 8 mesh network, even when infinite buffers are allocated between links. However, in the new parallel buffers adopted NoC architecture, the performance already outperforms, even when two-FIFO parallel buffer per incoming port are applied, as shown in Figure 4a. Furthermore, by applying a four-FIFO parallel buffer per incoming port, the maximum throughput for 8 × 8 mesh reaches up to 0.45 (about 13% improvement). With comparison to the base NoC architecture, four-FIFO parallel buffers per incoming port achieve an optimal performance benefit. Also, compared with the general virtual channel approach [3] where at least 8 virtual channels per physical channel are required to get nominal performance and resolve the deadlock problem, the proposed NoC architecture with parallel buffers has its own benefit.

OVERVIEW OF THE PROPOSED TRAFFIC MODEL

As part of enhancing NoC modeling and simulation improvements, we propose a generic traffic model for NoC based on traffic traces obtained from a full system simulation or real system devices. This model combines the spatio-temporal characteristics of traffic with three independent components, (H_s, λ_s, $\delta_{(s,d)}$) where s and d represent the indices of the source node and the destination node, respectively. With three independent components, the given traffic can be analyzed and characterized in a statistical manner. Different from the approach used in [15], each statistical component is derived per node. To define the burstiness of each node, the Hurst exponent, H_s, for source node, s, is adopted. As a component of the characteristics of self-similarity, H_s determines the temporal burstiness of traffic at each node, that is, the peak size of injection packets and their injection patterns of arrival time. To define one of the spatial properties in NoC traffic traces, distribution of the average injection rate at every node, denoted by λ_s, is captured. Finally, $\delta_{(s,d)}$, representing distribution of the traffic ratio from s node to d node in the given injection rate, λ_s, is extracted.

For each component of our (H_s, λ_s, $\delta_{(s,d)}$) traffic model, we analyze and extract the proposed statistical distribution against 8 traffic traces used in [16]. Those are SPLASH-2 [17] traces gathered by running the benchmarks on Bochs [18], a multiprocessor simulator with an embedded Linux 2.4 kernel. Each benchmark was run in Bochs with 49 ($= 7 \times 7$) concurrent threads, and the memory trace is captured. This memory trace is then applied to a memory system simulator that models the classic MSI (Modified, Shared, Invalid) directory-based cache coherence protocol, with the home directory nodes statically assigned, based on the least significant bits of the tag, distributed across all processors in the entire chip.

TRAFFIC MODELING

In this Section, we explain the details of our (H_s, λ_s, $\delta_{(s,d)}$) traffic model. Based on the extracted parameters, the procedure for generating a synthetic traffic trace will be provided as well. Traffic modeling is an important component of our parallel buffer evaluation, since lack of good traffic loads could potentially skew the outcome of our architectural evaluation for NoC.

Temporal Burstiness: H_s

In classic networks, self-similarity is one of the key features to characterize burstiness, as well as Long-Range Dependence (LRD) of traffic, in the temporal sense. To measure such burstiness and LRD, the Hurst parameter, H, is used, where $H \in (1/2, 1)$ indicates the presence of LRD. As many communication traffics are proven to be statistically self-similar, some researchers already showed that the traffic in NoC also has a self-similar characteristic [15,19]. Thus, we parameterize such a degree of burstiness or LRD using H. Furthermore, in order to be accurate, this parameter, indicating burstiness is analyzed on every injection node.

Because the definitions of self-similarity are well described in the literature, in this section, a brief description of self-similarity will be introduced. For additional details, refer to [20-23].

Considering a cumulative process $Y(t)$ with stationary increments, let X_t be its corresponding incremental process:

$$X_t = Y(t) - Y(t-1). \tag{1}$$

Process $X_s^{(m)}$ is defined as an aggregated process of X_t if:

$$X_s^{(m)} = [X_{sm-m+1} + X_{sm-m+2} + ... + X_{sm}]/m. \tag{2}$$

Process X_t is *self-similar* if X_t is indistinguishable from $X_s^{(m)}$. Because this is a very restrictive definition, usually *second-order self-similarity* is considered for traffic analysis, i.e. *auto-covariance* of the original and aggregated processes should be the same:

$$\gamma^{(m)}(k) = \gamma(k), \tag{3}$$

$$\lim_{m\to\infty} \gamma^{(m)}(k) = \gamma(k), \tag{4}$$

where $\gamma(k) = E[(X_t - \mu)(X_{t+k} - \mu)]$ and $\gamma^{(m)}(k) = E[(X_s^{(m)} - \mu)(X_{s+k}^{(m)} - \mu)]$. Process X_t is *exactly* second-order self-similar or *asymptotically* second-order self-similar if Equation 3 or Equation 4 is satisfied, respectively.

In order to measure the degree of self-similarity, the Hurst parameter, H, is used, where a process is self-similar with parameter $H (0 < H < 1)$ if:

$$Y(t) = k^H Y(kt), \qquad \forall k > 0, \quad t \geq 0, \tag{5}$$

which means that the original and normalized aggregated processes should have the same distribution. In other words, the self-similarity can be understood as the ability of an aggregated process to preserve the burstiness of the original process, i.e. the property of *slowly decaying variance*:

$$var(X^{(m)}) \sim m^{2H-2}. \tag{6}$$

In this paper, Equation 6 is computed to measure the Hurst parameter, H. Table 2 shows the measured H value per node for eight different traces.

Injection Rate: λ_s

As one of the spatial parameters in our traffic the model, traffic injection rate determines the distribution of the injection load per node. In [15], this spatial injection distribution is parameterized by the standard deviation, σ, of the injection distribution with an actual coordinate assignment. In that approach, it assumes that the actual results possess Gaussian-type distributions. Even though that approach can help the injection distribution be quantified using a single σ value, the mapping to Gaussian-like distribution is not always accurate in a real NoC traffic situation. Also, it requires large amounts of computation to

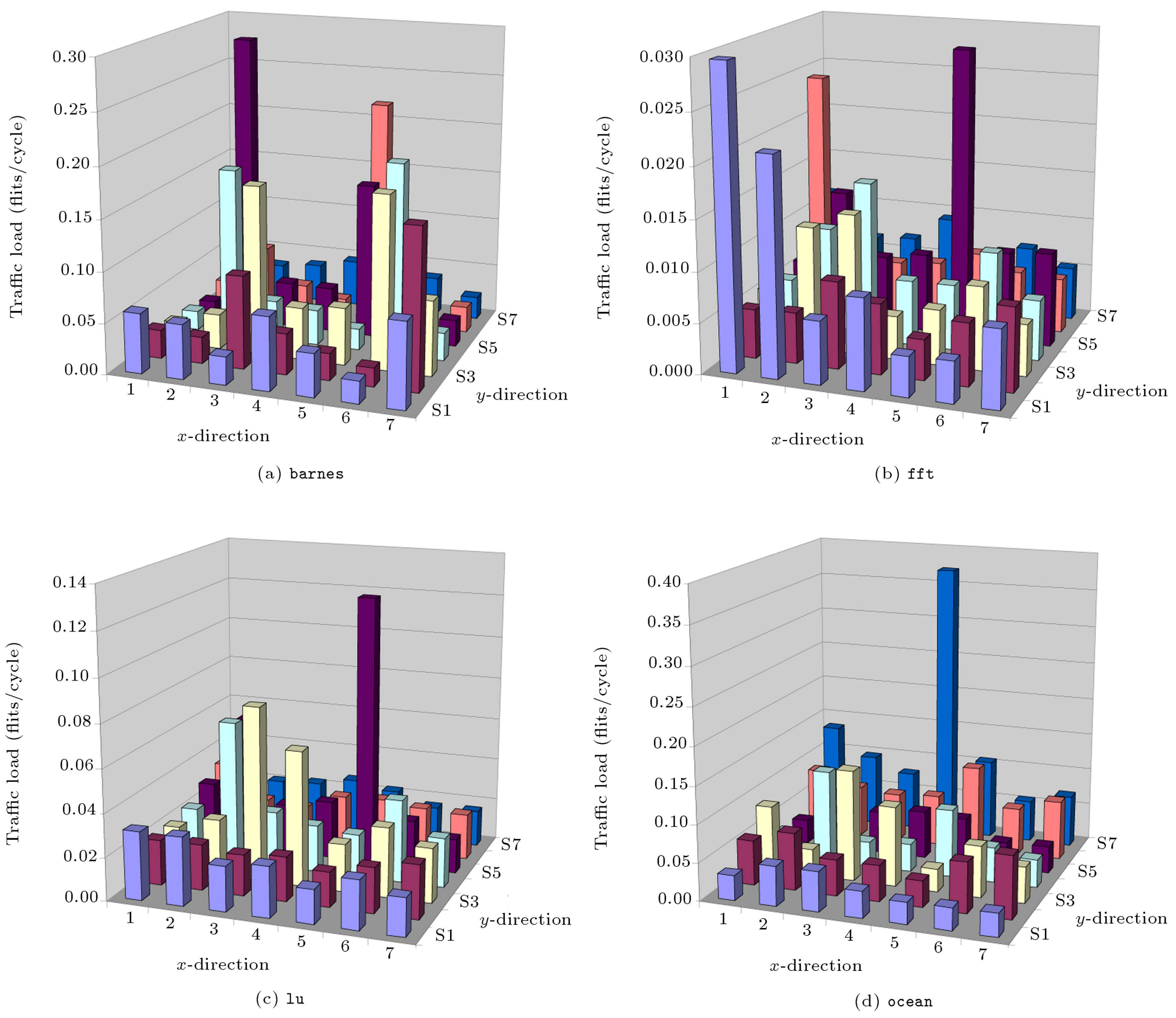
Table 2. Measured Hurst parameter for traffic traces in 7 × 7 mesh.

`barnes`						
0.95322	0.930023	0.976069	0.976115	0.931223	0.958229	0.967662
0.953165	0.958007	0.906468	0.927599	0.932508	0.952065	0.938917
0.961447	0.950251	0.960211	0.976686	0.923968	0.9375	0.864216
0.958195	0.890648	0.918889	0.93756	0.958898	0.927886	0.904769
0.956649	0.985442	0.939089	0.920304	0.904582	0.938582	0.885551
0.961257	0.88265	0.938839	0.95921	0.96127	0.917485	0.928582
0.869166	0.965638	0.895383	0.906786	0.909061	0.883905	0.88992
`fft`						
0.975095	0.941273	0.948791	0.964113	0.951594	0.956773	0.961565
0.95084	0.951059	0.966805	0.966816	0.960732	0.952544	0.963932
0.95335	0.978645	0.977989	0.962314	0.958788	0.968619	0.963367
0.961431	0.973687	0.992332	0.959887	0.96177	0.968292	0.964245
0.964755	0.972209	0.972016	0.97097	0.990788	0.968756	0.974325
0.993044	0.972676	0.971063	0.975506	0.976879	0.973728	0.970238
0.975961	0.980018	0.977654	0.973719	0.968751	0.964043	0.960366
`lu`						
0.924199	0.953505	0.956322	0.953416	0.955267	0.961355	0.949458
0.958744	0.95534	0.953154	0.954404	0.954908	0.952774	0.956247
0.954391	0.967936	0.985967	0.980213	0.953937	0.95738	0.956906
0.959172	0.982929	0.961082	0.954406	0.95408	0.967851	0.956651
0.963854	0.9776	0.95748	0.965053	0.992639	0.962783	0.954991
0.967356	0.950398	0.952623	0.964804	0.954611	0.961498	0.962472
0.956253	0.960129	0.957397	0.95746	0.957518	0.95611	0.954621
`ocean`						
0.880499	0.808062	0.794881	0.845817	0.799616	0.816938	0.832904
0.7324	0.784654	0.830542	0.826276	0.77571	0.753008	0.795864
0.901085	0.865756	0.723628	0.749781	0.781642	0.752834	0.742008
0.802515	0.757628	0.798828	0.75681	0.760965	0.790886	0.750362
0.82159	0.810766	0.726558	0.748242	0.777665	0.78399	0.78296
0.818765	0.770188	0.809174	0.785652	0.803659	0.762584	0.782661
0.837008	0.761783	0.87089	0.763205	0.77494	0.78582	0.778135

find out the exact coordinate assignment. Hence, in this paper, the original distribution of injection rate on every node is kept as it is. This enables more accurate synthetic traffic generation than σ-based Gaussian-like distribution. Figure 5 shows injection rate distributions for traffic traces in a 7 × 7 mesh.

Spatial Distribution: $\delta_{(s,d)}$

Another spatial distribution, $\delta_{(s,d)}$, represents the traffic ratio from source node s to destination node d, based on injection rate λ_s. In [15], spatial hop distribution p is adopted. In order to formulate the hop count distribution model, the mechanism was applied so that the mapping should not choose a receiver whose distance is d hops from the sender unless it cannot find one. Also, in that model, there is no concern about the geometry of destination nodes. In other words, all nodes with the same d-hop distance from the source node are considered to have the same statistical characteristics. This assumption is somewhat different from typical NoC traffic, regardless of the optimal communication mapping. However, our model considers the difference of location of destination nodes within the same distance of hops when the traffic ratio between the source and the destination node is analyzed. Moreover, the matrix of traffic ratio from each source node is constructed



(a) `barnes` (b) `fft` (c) `lu` (d) `ocean`

Figure 5. Injection rate distributions for traffic traces in 7×7.

in order to characterize the spatial distribution of source/destination pairs. Figure 6 illustrates the distribution of traffic ratio for each node in the `barnes` application.

Synthetic Traffic Generation

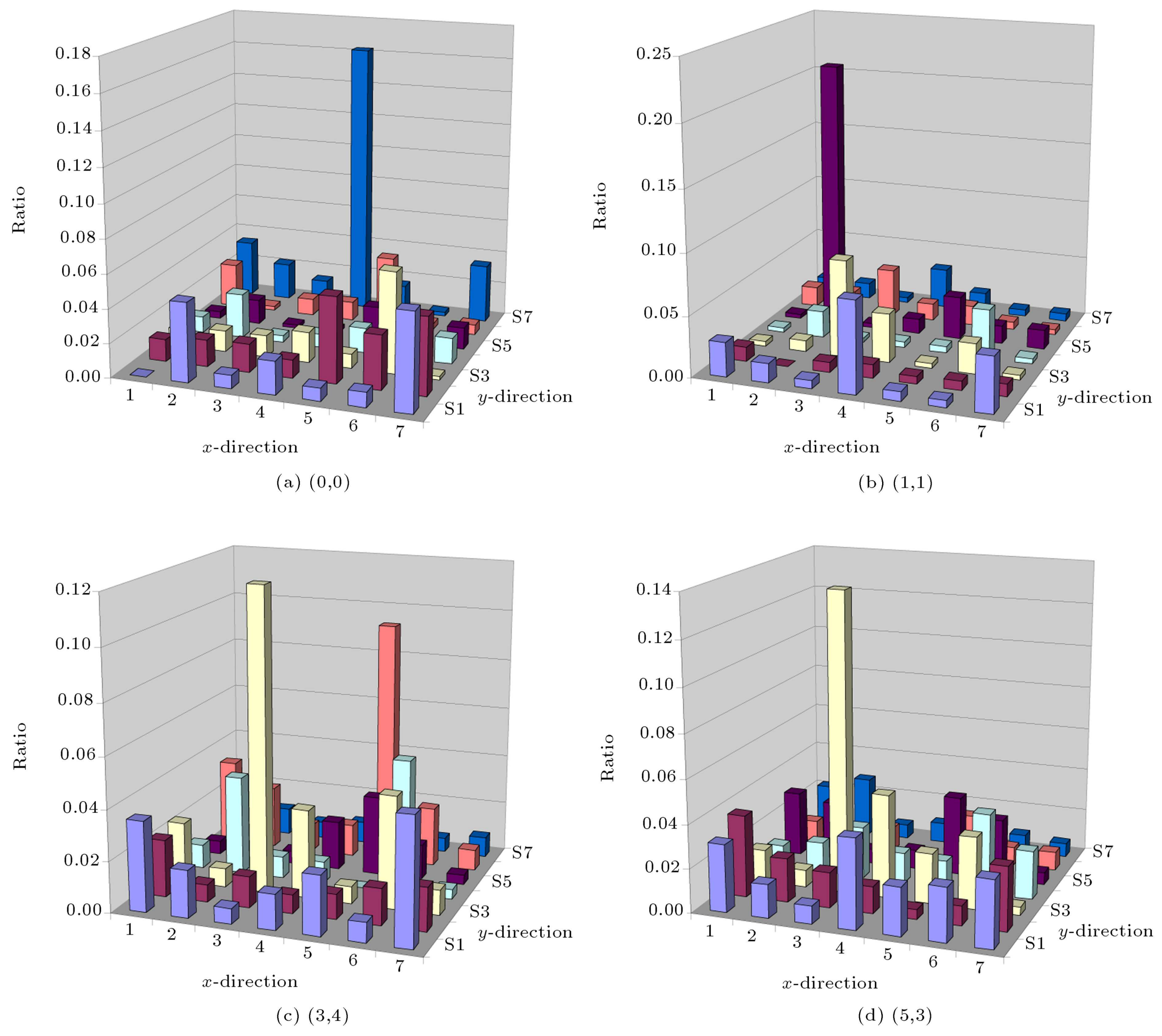
To describe how our (H_s, λ_s, $\delta_{(s,d)}$) model can generate synthetic network traffic, we implemented *tgNePA*, a tool that automatically generates NoC traffic of the given network topology from the configured (H_s, λ_s, $\delta_{(s,d)}$) traffic model. Figure 7 shows the traffic generation flow in *tgNePA*.

tgSelfSimilar: Traffic Generation Based on ($\boldsymbol{H_s}$, $\boldsymbol{\lambda_s}$)

To generate self-similar NoC traces, *tgNePA* uses the method described in [24]. For this method, the synthetic self-similar traffic is obtained by aggregating multiple sub-streams, each consisting of alternating Pareto-distributed on/off periods. Pareto distribution is defined by a heavy-tailed distribution with the probability-density function $f(x) = ab^{\alpha}/x^{\alpha+1}$, $x \geq b$, where α is a shape parameter and b is a location parameter. Pareto distribution with $1 < \alpha < 2$ has a finite mean and an infinite variance. To generate Pareto-distributed values, the following formula is used: $X_{Pareto} = b/[U^{1/\alpha}]$ where U is a uniform random variable ($0 \leq U \leq 1$). The Hurst parameter, H, of the self-similar trace generated by this method can be derived by $H = (3 - \alpha)/2$ [24,25].

Additionally, while generating Pareto-distributed values, the injection rate for each sub-stream is controlled. Therefore, by applying λ_s to each generation of the self-similar stream for corresponding node s, the (H_s, λ_s) configured self-similar traffic is obtained.

Depending on the method of self-similar traffic generation, its accuracy may be varied. To minimize



(a) (0,0) (b) (1,1) (c) (3,4) (d) (5,3)

Figure 6. Distributions of traffic ratio on selected nodes for `barnes` traffic trace in 7×7.

error between the expected (H_s, λ_s) and the measured value from the generated traffic, a recursion is applied, as shown in the first phase of *tgSelfSimilar* (see Figure 7). Along with generating self-similar traffic with the expected (H_s, λ_s) configuration, (H'_s, λ'_s)-tuple components of the generated traffic are measured. If the error for the expected (H_s, λ_s), as compared to the measured (H'_s, λ'_s), is acceptable, then the generated self-similar traffic is delivered to the next step, *splitPE*. Otherwise, generation of self-similar traffic with a similar configuration is repeated.

splitPE: Traffic Generation Based on $\delta_{(s,d)}$

The second phase generates the destination node upon the generated self-similar traffic of each node. Because the ratio of traffic from each source node, s, to each destination node, d, is already provided by the distribution of $\delta_{(s,d)}$, generation of the destination node for each instance of traffic from corresponding source node s can be accurately controlled. Different from the Trident's approach [15], the ratio of traffic for each pair of source and destination is separately assigned. Therefore, the distribution of source/destination pairs can be more accurately emulated.

VALIDATION

Each synthetic traffic is generated using analyzed (H_s, λ_s, $\delta_{(s,d)}$) for each application mentioned in the previous Section. In order to control the recursion of *tgSelfSimilar*, we set the marginal error bound of H_s and λ_s to 5%. In the recursive generation of self-similar traffic for each node, Hurst parameter H_s can be easily matched with the given marginal percentage. However, in matching the lower injection rate, λ_s, it requires excessive computation time. For that reason, to reduce such a large computation time in matching the injection rate, a proportional margin value is applied as an alternative approach. That is, for a

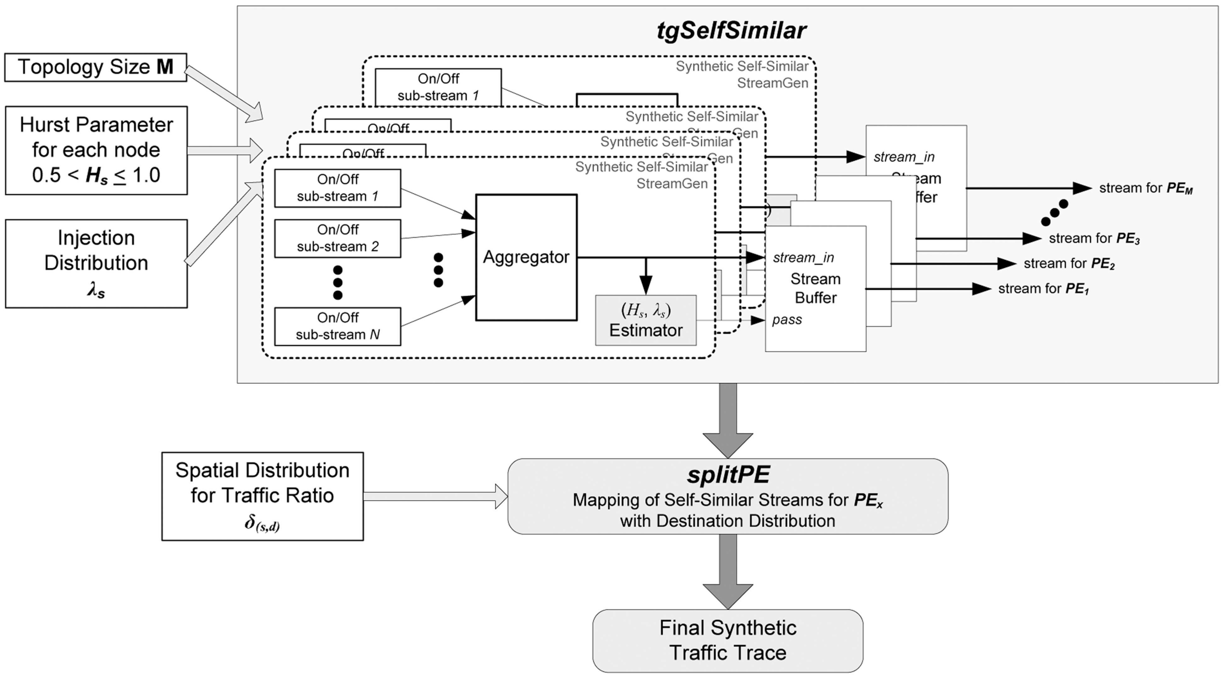

tgSelfSimilar
Topology Size M
Hurst Parameter for each node 0.5 < $H_s \leq$ 1.0
Injection Distribution λ_s
Synthetic Self-Similar StreamGen
On/Off sub-stream 1
On/Off sub-stream 2
On/Off sub-stream N
Aggregator
(H_s, λ_s) Estimator
stream_in
Stream Buffer
pass
stream for PE_M
stream for PE_3
stream for PE_2
stream for PE_1
Spatial Distribution for Traffic Ratio $\delta_{(s,d)}$
splitPE
Mapping of Self-Similar Streams for PE_x with Destination Distribution
Final Synthetic Traffic Trace

Figure 7. *tgNePA* traffic generation flow diagram.

relatively higher injection rate, a tighter margin value is applied. Reversely, in a relatively lower injection rate, a lower margin value is applied. For instance, by using the logarithmic scale of injection rate, the marginal value can be scaled by multiplying it with $|\log_{10}(\lambda_s)|^{|\log_{10}(\lambda_s)|}$. Because the higher injection rate is dominant, the effect of larger error at source nodes with lower injection rates can be minimized. Table 3 shows the measured Hurst parameter of synthetic traffic according to the analyzed traffic model of each application. The H_s parameter for synthetic traffic has accuracy in the range of a 2.7% to 4.3% average error bound. However, the accuracy of λ_s is varied depending on the level of injection rates of applications, because the proportional margin value to the level of injection rates is applied in matching the injection rate during the first phase of traffic generation. For instance, in `barnes` application, the average of injection rates of original traffic is 0.065 and the ratio of average error in injection rates is 6.8%. On the other hand, for `fft` application, the average of injection rates of original traffic is 0.0089 and the ratio of average error is 26%. In this case, the level of injection rates is relatively low, i.e. the scale factor to apply a proportional margin is 27 (=3^3) during the recursion. Therefore, the resultant synthetic traffic has relatively large error compared to the original injection rates.

For source/destination distribution $\delta_{(s,d)}$ of synthetic traffic, its accuracy is almost 100%, as shown in Figure 8.

Finally, throughout the cycle of NoC simulation, using original traffic traces as well as synthetic traffic traces, the accuracy of the overall network performance is observed (see Table 4).

CONCLUSION

We proposed a new parallel buffer structure and management scheme, as well as its optimal micro-architecture. By applying this proposed parallel buffer to the previously based NoC architecture, remarkable performance improvement was observed using simulations of various traffic patterns. Even though deadlock-freedom is realized by providing disjoint vertical channels instead of virtual channels, which is a general approach for this purpose, considerable performance benefit can be obtained by adding parallel buffers with a smaller number of FIFOs. Also by moving the header parseing unit into the parallel buffer controller, the timing balance between the parallel buffer and router logic can be obtained at the micro-architecture level. In order to enhance our simulation-based architectural validation approach, a generic traffic model for on-chip interconnection networks was described. To keep the temporal and spatial distribution of traffic traces, all statistical information is measured per node. In order to characterize the burstiness of injection nodes, Hurst parameter H_s is chosen. For specifying temporal statistics, the distribution of injection rates λ_s and the ratio of source/destination pairs $\delta_{(s,d)}$ on the given source node are used. With the proposed traffic model, we also introduced a recursive traffic generation method to minimize the error of statistical components, and allow synthetic traffic traces with

Table 3. Measured Hurst parameter for synthetic traffic traces in 7×7 mesh.

`barnes`						
0.922604	0.900864	0.929619	0.932654	0.887087	0.924292	0.924685
0.910217	0.92359	0.894023	0.892669	0.900064	0.922199	0.916118
0.913906	0.924976	0.918239	0.93298	0.878349	0.890662	0.87693
0.933494	0.873282	0.90397	0.912061	0.924956	0.891938	0.887996
0.926644	0.949166	0.894207	0.888905	0.879505	0.917606	0.898869
0.940055	0.898484	0.894776	0.911445	0.925797	0.888928	0.922339
0.831892	0.917855	0.917632	0.892862	0.879119	0.886606	0.862894
Average error ratio = 0.032						
`fft`						
0.930469	0.901261	0.904385	0.919863	0.905215	0.920039	0.927163
0.918589	0.930192	0.932514	0.93347	0.921322	0.913654	0.953305
0.907441	0.929731	0.931737	0.91638	0.911257	0.935479	0.91756
0.927311	0.927169	0.950335	0.933219	0.934989	0.923255	0.925058
0.930802	0.92372	0.95463	0.924153	0.942931	0.920807	0.942047
0.950361	0.932204	0.937753	0.935888	0.929717	0.925567	0.928409
0.944464	0.931497	0.933646	0.927396	0.928349	0.932712	0.920281
Average error ratio = 0.041						
`lu`						
0.920638	0.913734	0.926406	0.914993	0.912014	0.929957	0.930625
0.915871	0.913395	0.923574	0.93993	0.914495	0.925637	0.910338
0.919504	0.95263	0.937157	0.935734	0.908091	0.915819	0.916393
0.917747	0.936518	0.91593	0.915539	0.912704	0.94054	0.92322
0.919705	0.933109	0.91106	0.918234	0.944434	0.923524	0.913384
0.926757	0.904367	0.90686	0.927362	0.912072	0.916251	0.91677
0.917779	0.929412	0.920856	0.918243	0.920673	0.918731	0.924098
Average error ratio = 0.039						
`ocean`						
0.8483	0.783373	0.770411	0.875487	0.766814	0.828058	0.824162
0.739303	0.77524	0.798285	0.8164	0.80788	0.719764	0.780451
0.895681	0.849855	0.709589	0.728271	0.758329	0.756766	0.705397
0.841761	0.745178	0.807255	0.721676	0.736948	0.787291	0.71473
0.837015	0.799498	0.699795	0.717817	0.815538	0.754053	0.804242
0.805689	0.780835	0.790708	0.818814	0.764959	0.742495	0.753796
0.799607	0.76778	0.853409	0.795997	0.752484	0.785473	0.752427
Average error ratio = 0.027						

Table 4. Comparison of cycle accurate NoC simulation between original and synthetic traffic traces in 7 × 7 mesh.

	Original Traffic		**Synthetic Traffic**		**Error Ratio (%)**	
Application	**Offered load**	**Avg. Latency**	**Offered Load**	**Avg. Latency**	**Offered Load**	**Avg. Latency**
`barnes`	0.06522	7.26	0.06951	7.32	6.58	0.86
`fft`	0.00889	8.20	0.01125	7.95	26.63	2.94
`lu`	0.03560	7.67	0.03240	7.52	8.99	2.17
`ocean`	0.06881	7.65	0.07345	7.78	6.75	1.70
`radix`	0.03690	7.96	0.04177	8.00	13.18	0.42
`raytrace`	0.00636	7.99	0.00987	7.95	55.21	0.53
`water-nsquared`	0.01649	7.93	0.01939	7.77	17.60	2.22
`water-spatial`	0.02221	7.99	0.02529	7.76	13.83	2.89

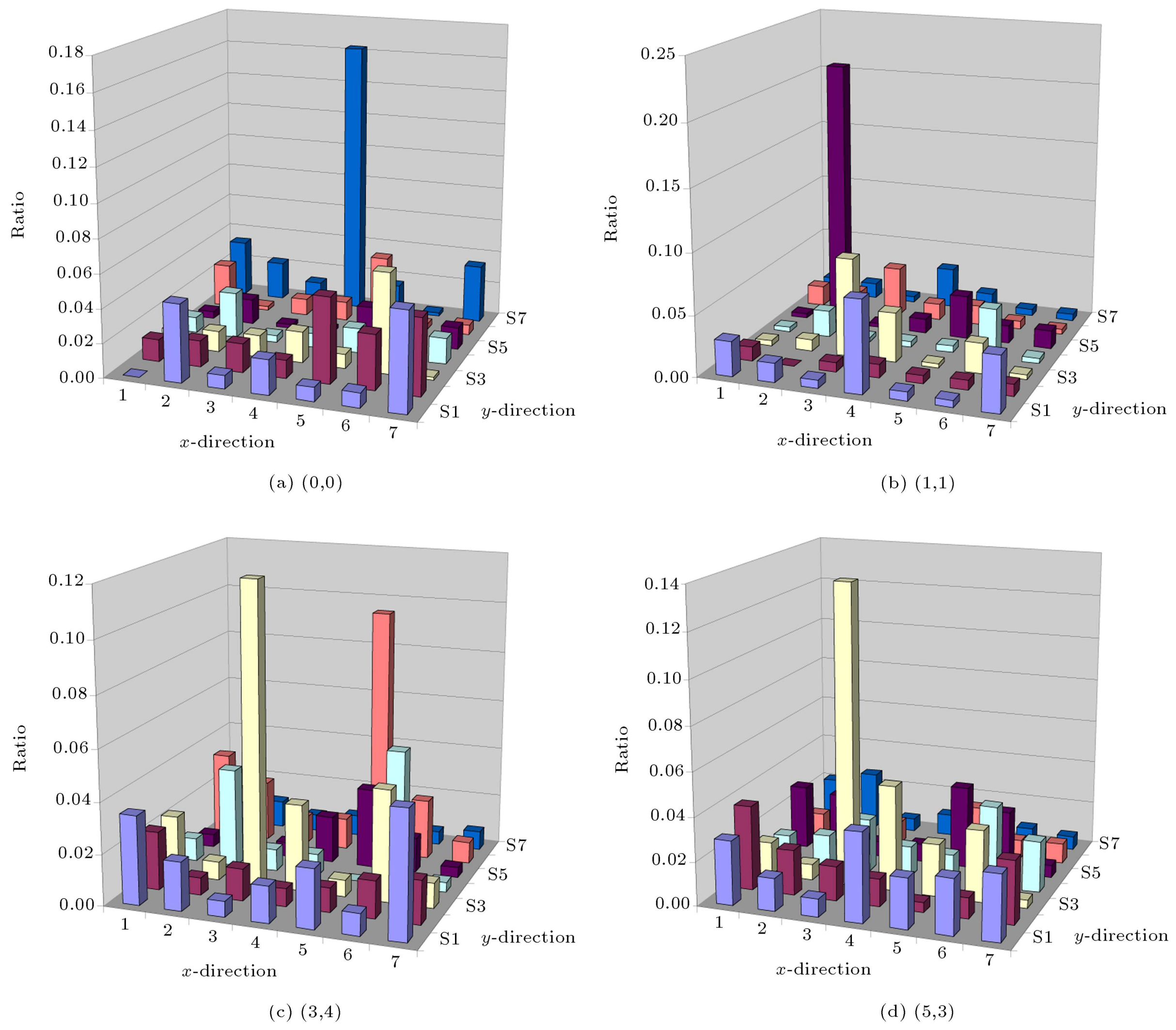


Figure 8. Distributions of traffic ratio on selected nodes for synthetic traffic trace of `barnes` application in 7 × 7.

similar temporal and spatial statistics to be generated. Throughout detailed comparison of each component and performance simulation, our proposed traffic model can reconstruct traffic patterns with a similar tendency of real NoC traffic and provide insights into NoC traffic.

REFERENCES

1. Dally, W.J. and Towles, B. "Principles and practices of interconnection networks", Morgan Kaufmann, San Francisco (2004).
2. Sullivan, H. et al. "A large scale, homogeneous, fully distributed parallel machine", in *ISCA'77*, pp. 105-117, ACM Press, New York (1977).
3. Seo, D. et al. "Near-optimal worst-case throughput routing for two-dimensional mesh networks", in *ISCA'05*, pp. 432-443, ACM Press, New York (2005).
4. Dally, W.J. and Seitz, C.L. "Deadlock-free message routing in multiprocessor interconnection networks", *IEEE Trans. Computer*, **C-36**(5), pp. 547-553, IEEE Computer Society, Washington (1987).
5. Glass, C.J. and Ni, L.M. "The turn model for adaptive routing", *J. ACM*, **31**(5), pp. 874-902, ACM Press, New York (1994).
6. Bahn, J.H. et al. "On design and analysis of a feasible network-on-chip (NoC) architecture", in *ITNG'07*, pp. 1033-1038, IEEE Computer Society, Washington (2007).
7. Dally, W.J. "Virtual-channel flow control", *IEEE Trans. Parallel and Distributed Systems*, **3**(2), pp. 194-205, IEEE Press, Piscataway (1992).
8. Boppana, R.V. and Chalasani, S. "Fault-tolerant wormhole routing algorithms for mesh networks", *IEEE Trans. Computers*, **44**(7), pp. 846-864, IEEE Computer Society, Washington (1995).
9. Zhou, J. and Lau, F.C.M. "Adaptive fault-tolerant wormhole routing with two virtual channels in 2D meshes", in *ISPAN'04*, pp. 142-148, IEEE Computer Society, Los Alamitos (2004).

10. Vaidya, A.S. et al. "Impact of virtual channels and adaptive routing on application performance", *IEEE Trans. Parallel Distributed Systems*, **12**(2), pp. 223-237, IEEE Press, Piscataway (2001).
11. Rezazad, M. and Sarbazi-azad, H. "The effect of virtual channel organization on the performance of interconnection networks", in *IPDPS'05*, pp. 264.1, IEEE Computer Society, Washington (2005).
12. Lahiri, K. et al. "Evaluation of the traffic-performance characteristics of system-on-chip communication architectures", in *VLSID'01*, pp. 29-35, IEEE Computer Society, Washington (2001).
13. Hu, J. and Marculescu, R. "DyAD - smart routing for network-on-chip", in *DAC'04*, pp. 260-263, ACM Press (2004).
14. Palesi, M. et al. "A methodology for design of application specific deadlock-free routing algorithms for NoC systems", in *CODES+ISSS'06*, pp. 142-147, ACM, Seoul (2006).
15. Soteriou, V. et al. "A statistical traffic model for on-chip interconnection networks", in *MASCOTS'06*, pp. 104-116, IEEE Computer Society, Washington (2006).
16. Kumar, A. et al. "Express virtual channels: Towards the ideal interconnection fabric", in *ISCA'07*, pp. 150-161, ACM, San Diego (2007).
17. SPLASH-2, http: //www-flash.stanford.edu/apps/SPLASH
18. Lawton, K.P. "Bochs: A portable PC emulator for unix/X", *Linux Journal*, **1996**(29es), Specialized Systems Consultants, Inc., Seattle (1996).
19. Varatkar, G.V. and Marculescu, R. "On-chip traffic modeling and synthesis for MPEG-2 video applications", *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, **12**(1), pp. 108-119, IEEE Educational Activities Department, Piscataway (2004).
20. Doukhan, P. et al., *Theory and Applications of Long-Range Dependence*, in Birkhäuser Boston (2002).
21. Park, K. and Willinger, W., *Self-Similar Network Traffic and Performance Evaluation*, in John Wiley & Sons, Inc., New York (2000).
22. Leland, W.E. et al. "On the self-similar nature of Ethernet traffic (extended version)", in *IEEE/ACM Trans. Network*, **2**(1), pp. 1-15, IEEE Press, Piscataway (1994).
23. Tsybakov, B. and Georganas, N.D. "Self-similar processes in communications networks", in *IEEE Trans. Information Theory*, **44**(5), pp. 1713-1725, IEEE Press, Piscataway (1998).
24. Taqqu, M.S. et al. "Proof of a fundamental result in self-similar traffic modeling", *SIGCOMM Computer Communication Review*, **27**(2), pp. 5-23, ACM, New York (1997).
25. Willinger, W. et al. "Self-similar through high-variability: Statistical analysis of Ethernet LAN traffic at the source level", in *SIGCOMM Computer Communication Review*, **25**(4), pp. 100-113, ACM, New York (1995).