

SNAPP Technologies Corporation's

ROUTER ARCHITECTURE USING COMBINATORIAL DESIGNS

Abstract

The paper presents a new type of parallel router architecture based on a combinatorial arrangement of routing processing elements. The suggested router architecture is based on a mathematical approach of combinatorial designs that lead to a novel type of multiprocessor communication networks. The so-called SNAPP architecture (Shared Network Acceleration Parallel Processor) is build up on this development. Unlike ATM-based routers, the proposed schema allows for easier multicast implementation. In comparison with the existing architectures, the proposed construction allows to create a router with a lesser number of interconnections between processing elements and reduced operational load. With this construction the performance of information transmission in large data communication networks, like the Internet, can be improved in terms of speed and reliability.

Keywords: Computer communication networks, router architecture, Internet, multiprocessor systems, combinatorial designs

Introduction Routing Problem

A Router is a special purpose computer or a special purpose computer program, which handles the communication of packet-switched networks.

Routers play a decisive role in computer network, particularly Internet, sometimes called a network of routers. Packet-switched networks employ the idea of splitting big data blocks to be sent from one node of the network to another onto smaller chunks named packets and handle these packets each individually. The nodes need not be connected directly; it is presumed that there will be one or more intermediate nodes on the way from the source to the destination. Packets travel independently through the chain of intermediate nodes performing routing, until they reach the destination.

The operations of a typical router are split onto three independent processes:

1. Path determination
2. Packet switching
3. Administration and control

The process encounters local problems inside the routers, such as next hop determination, as well as global problems of optimal channel assignments for information flows between the nodes. Some of the problems - not related to the routing itself, such as packet queues, network security, and routing table(s) management. The Basic Reference Model of the Open Systems Interconnection (OSI) specifies seven layers of network communication organization. From the OSI Model point of view, a router is a network layer device that uses certain metrics to determine the optimal path to dispatch the traffic.

To illustrate a different approach to solve routing problems in this work we will analyze three existing routing architectures and propose a new one.

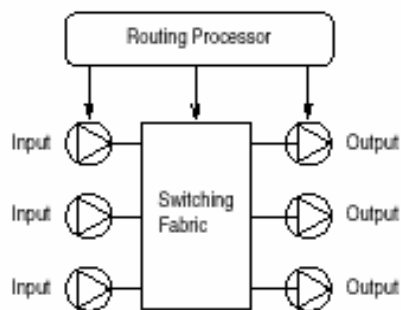


Figure 1: Architecture of a router/ Generic router

A generic router has four architectural components (Fig.1]): input ports receiving data, output ports transmitting it, switching fabric that interconnects input ports with output ports and routing processor which performs coordination and control [KeshlavSharima98]. Although actions of these components are to be coordinated, their operations can run independently. The main function of a router is to move a packet from an input channel to one of the output channels, which requires intensive processing activities or application-specific integral

circuits for switching. The switching in its turn needs control and it necessitates extensive computation performed practically in real time. Routers have to maintain a substantially large table that reflects the state of the routes in the whole Internet, neighbor routers and channels between them. This look-up table is searched to determine an optimal destination line for packet transmission. While the best algorithms can perform as fast as $O(\log_2 N)$, where N is the number of bits in the address [Kumar98], it requires significant computation to create hash tables necessary for these algorithms to work. In some circumstances [BGPstorm] routing table updates can stimulate excessive computation so additional resources must be reserved to retain functionality in a hostile environment like modern Internet.

The other-than-routing processing may involve blocking certain packets or modifying their transport headers and information contents. As the number of services, which system administrators expect to find at the routers, grows the more powerful router hardware needs to be deployed. Thus, the continuous growth and evolution of the Internet heavily overloads traffic handling facilities and demands for router devices of higher performance.

The shared memory architecture utilizes common RAM (Random Access Memory) as switching fabric (Fig.1). Among other advantages, it can be easily implemented with a universal computer allowing having the lowest possible cost per port. Overall routing performance is very limited by the memory bandwidth used by all the input/output operations and the CPU (Central Processing Unit) constantly accessing data from the program. We can also use the shared memory architecture as a basic element for more complex designs and will use the name Processing Element (PE) for it. A simplest PE consists of one input port, one output port, a memory module and a CPU controlling it.

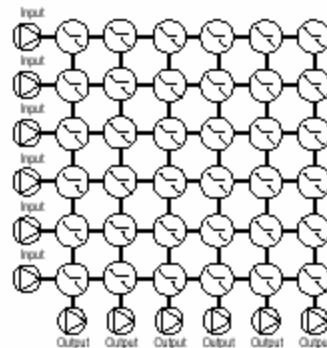


Figure 2: Crossbar Router

To increase the router performance beyond the RAM bandwidth we can equip every pair of input/output channels with an independent PE (Fig.2 Crossbar-router). This architecture allows to route data streams at the speed up to PE bus bandwidth.

To achieve fast throughput routers have been made using shared parallel processors [Asthana92] (see Fig. [fig: Shared-processors].) While a single PE cannot perform routing fast enough, a pool of PEs can process at any desired speed. An arriving packet is stored in the input interface. The interface transmits the packet header to the next available PE through the interconnection circuitry. The PE responds with the appropriate output interface number so the input interface can transfer the whole packet to it. Thus the slower process of

routing can be dynamically spread across multiple processing units and actual packet forwarding is performed by a fast switch. Currently it is custom to use as the core switch an ATM (Asynchronous Transfer Mode) based device.

Because of extensive development and availability of ATM hardware components, they are widely used in routers [KeshlavSharima98] and it creates a problem with multicast traffic. A multicast packet may be designated to several output interfaces at once, so the input interface should arrange multiple transmissions, which delays other input packets and processing until all transmissions, are over. It also increases the latency as the packet for the output channel, which happen to be the last in queue will be delayed until all other ports have receive it.

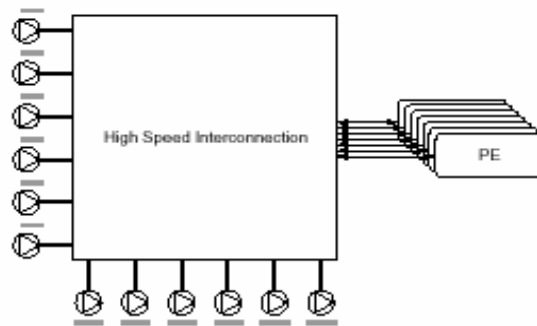


Figure 3: Router with Shared Parallel Processors

In this paper, we suggest a new type of routing architecture. It naturally brings together increased processing power and effective commutation facilities. This architecture also can easily support multicast transmission. It employs the technique of combinatorial designs as described in the next section.

Combinatorial Designs

Combinatorial design is a popular mathematical construction, which presents a collection of subsets of a given set, selected in accordance with some prescribed condition. Among different types of combinatorial designs, one presents a particular point of interest in computer applications – the Balanced Incomplete Block Designs (BIBD)[1]. Every subset in a non-trivial BIBD contains fewer elements than the whole set.

A BIBD is characterized by five parameters:

- b - the number of blocks (subsets),
- v - the number of elements in the whole set,
- r - the number of blocks containing each element,
- k - the number of elements in each block, and
- λ - the number of blocks in which every pair of elements appears

So a BIBD is also called a (b, v, r, k, λ) configuration

These parameters satisfy relations [1]:

$$b \cdot k = v \cdot r$$

$$\lambda \cdot (v-1) = r \cdot (k-1)$$

$$b \geq v$$

A BIBD exists only if all conditions are satisfied. A BIBD, for which $v = b$ and $r = k$ is called a symmetric BIBD. For $\lambda = 1$, a symmetric BIBD should satisfy the following criteria:

$$v = b = s^2 + s + 1$$

$$r = k = s + 1$$

$$\lambda = 1$$

$$s = p^m \quad \text{where } p \text{ is a prime and } n \text{ is a positive number}$$

So far no general necessary conditions for existence of symmetric BIBDs have been proven for higher λ [1].

For the sake of illustration, let us consider a set of 7 elements $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$.

A BIBD constitutes a collection of subsets $b_i = 1, 2, \dots, 7$ as shown by the columns in Table 1.

	b_1	b_2	b_3	b_4	b_5	b_6	b_7
x_1	1	0	0	0	1	0	1
x_2	1	1	0	0	0	1	0
x_3	0	1	1	0	0	0	1
x_4	1	0	1	1	0	0	0
x_5	0	1	0	1	1	0	0
x_6	0	0	1	0	1	1	0
x_7	0	0	0	1	0	1	1

Table 1: A symmetric BIBD

The pairwise balanced distribution of the elements with $\lambda = 1$ assumes that any pair of elements of X , say x_1 and x_3 can be found in exactly one block (which in our case is b_7).

In a symmetric design, the number of elements satisfies $4 \cdot n - 1 \leq v \leq n^2 + n + 1$, where $n = k - \lambda$ is the order of the design [2]. The upper bound corresponds to a finite projective plane of order n , in which symmetric BIBDs will have $(n^2 + n + 1, n^2 + n + 1, n + 1, n + 1, 1)$ – configurations. Some well-known projective plane configurations are shown in Table 2.

n	<i>BIBD</i>
2	(7,7,3,3,1)
3	(13,13,4,4,1)
4	(21,21,5,5,1)
5	(31,31,6,6,1)
6	<i>does not exist</i>
7	(55,55,8,8,1)
8	(73,73,9,9,1)
...	...

Table 2: Some symmetric BIBDs

While we need the whole set of BIBD blocks to cover any possible pair, all elements of a given set can be found in a smaller subset of blocks. Such a subset for a symmetric BIBD contains r blocks.

For example, consider element x_1 . This element is contained in blocks b_1 , b_5 , and b_7 . The full covering implies that $b_1 \cup b_5 \cup b_7 = X$. Thus, $b_1 = \{x_1, x_2, x_4\}$, $b_5 = \{x_1, x_5, x_6\}$, $b_7 = \{x_1, x_3, x_7\}$, so the union of these sets apparently presents the whole set $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$. This statement follows directly from the property of pairwise balance.

Indeed, the block with a given element contains its pairings with all other elements. Therefore, the union of these blocks must contain all the elements of the basic set X .

While the problem of design generation for symmetric designs is considered trivial, asymmetric design generation has no solution other than direct computation.

The pairwise balanced distribution of the elements with $\lambda = 1$ assumes that any pair of elements of X , say x_1 and x_3 can be found in exactly one block (which in our case is b_7).

In a symmetric design, the number of elements satisfies $4 \cdot n - 1 \leq v \leq n^2 + n + 1$, where $n = k - \lambda$ is the order of the design [2]. The upper bound corresponds to a finite projective plane of order n , in which symmetric BIBDs will have $(n^2 + n + 1, n^2 + n + 1, n + 1, n + 1, 1)$ – configurations. Some well-known projective plane configurations are shown in Table 2.

Router Architecture Using Combinatorial Designs

An approach, similar to the presented solution of the problem of parallel execution of sequential code, may be applied to the routing or switching problem, especially in packet-switching networks, where routing requires intense real-time computations. In a network router, a packet arrived through an input channel should be transmitted to one of the output channels according to the routing table. Thus routing in general can be reduced to the problem of pairwise coupling of input and output channels.

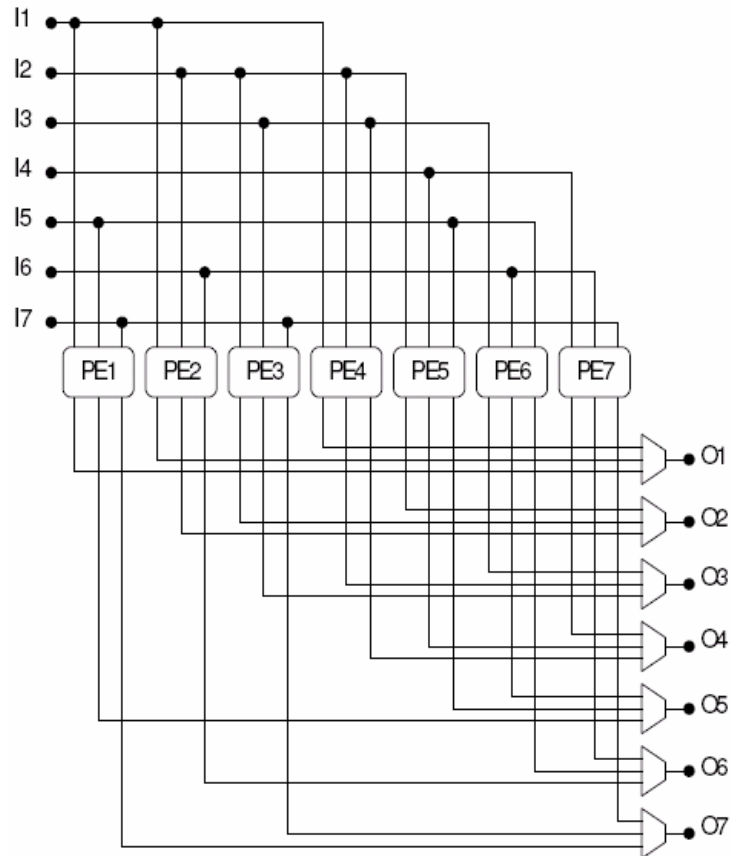


Figure 4 : (7,7,3,3,1) Combinatorial Router

The routing organization using the suggested scheme is shown in Fig. 4. In the case of a (7, 7, 3, 3, 1) router, each PE has three input channels, a memory module, a CPU and three output channels. Initially, all the PEs should be loaded with the routing information. Each of the PEs contains a part of the routing table, corresponding to channels directly connected to the PE. In the given example, output contention resolution is made with output buffers O1 - O7, although it could be done by other means. Assume a packet came from the input interface I2 and should be routed to the interface O5. The interface I2 is connected to the processing elements PE2, PE3 and PE5 with the line 2. All of the elements listed start to search the route to the destination in their routing tables. Processing elements PE2 and PE3 fail as they have no route to O5. PE5 has the route listed in its routing table and it forwards the packet received to O5. There is an ambiguity if a packet must be routed to a destination channel with the same sequence number as the source channel. This may happen, for example, if there is a loop in the routing table, a mis-configured downstream router or an address spoofing attack occurred. In this case the packet should be dropped, with possible notification of the sender. Due to the distributed nature of processing, the suggested design has better resistance to overloads while Denial of Service (DoS) attacks or global routing instability has happened. While overload of any of the channels is fatal for a traditional single-processor, shared-bus and shared memory router, a combinatorial router remains mostly operational even some of the channels and PEs get clogged. This makes the combinatorial router a good candidate for Internet routers.

Pre-processing Scheduling

One of the problems with parallel processing in general, and routing in particular, is scheduling. Given a problem (a packet for a router), some mechanism should allocate processing resources and properly distribute the data among them. Due to its complexity, the problem requires substantial resources by itself. Thus, the scheduling operation could easily be a bottleneck for the whole system. Unlike existing solutions, a combinatorial router has no such scheduling overhead. There is a predetermined subset of processing nodes, which could perform the routing for a packet, arrived through an input channel. It is worth noting, that bare wires with no computational resources involved doing such scheduling. In the previous example, the packet (arriving through I2) will be processed by a fully covering subset of processing nodes, which consists of {PE2, PE3, PE5}. Such a subset is sufficient for routing to any combination of output interfaces.

Routing Table Reduction

As you can see in Fig. 4, every processing element has access to a subset of output channels. Thus, the element needs only a part of the routing table, which corresponds to the connected outputs only. It is expected that some overhead is introduced by the "longest prefix match algorithm". The routing program in a PE needs to consider whether there could be other PEs which could route a packet. In the worst-case scenario, when prefixes of every length have different destinations, every processing element should hold the entire routing table.

Multicast Implementation

It is obvious that combinatorial router can easily route multicast traffic. As it was noted before, the minimum necessary number of processing elements, which can route a packet to any number of output channels is equal to the number of blocks in a fully covering subset, which is r . With virtual output queues, multicast implementation may be not different from a unicast transmission. A single copy of a packet is queued for transmission in multiple output queues. Although this is a practical and economical approach, it does not guarantee the lowest possible delay variation.

Route Caching

In general, Internet traffic can be characterized as non-uniform. Over time, users' interests are shifting from one host to another, generating funnel flow towards the attraction source. These bursts can create an excessive load on the routing elements, but fortunately, the number of source-destination pairs is limited, and so any caching technique could be very beneficial. Once a destination route has been determined, the router can put the destination address along with destination port index into cache. Any subsequent packet should first be verified against what has been cached to find if a similar packet has been routed. To make use of caching, a parallel router should possess a property of persistence. If a processing element had performed routing for a packet, the successor packets should be processed by the

same processing element. As the presented architecture has a predetermined route for every input/output combination, the property of persistence successfully holds.

Performance Evaluation

To evaluate performance of the proposed architecture, a simulator program had been developed using Discrete Event Simulation technique [13]. The model is using the following assumptions:

1. Routers are implemented with a switching architecture
2. No constraint on delay variation is imposed, thus multicast transmissions can be performed asynchronously
3. Routers implement virtual queuing
4. The traffic is Poisson distributed
5. Channel load is 10% of the bandwidth
6. Packets are equal size of 352 bytes (average in the Internet)

The following statistics is collected:

- ? Total throughput
- ? Latency
- ? Drop rate

The routing element of the combinatorial router and the single processor router have similar architecture, shown in Fig. 5.

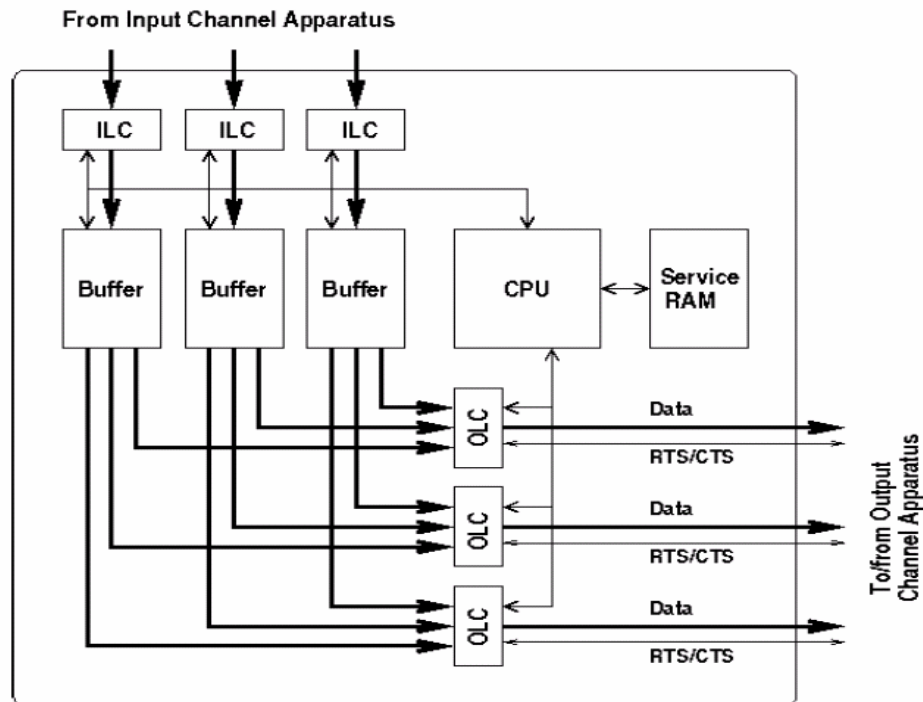


Figure 5: Model Router Architecture

A single processor router or a combinatorial Processing Element (PE), consists of input channels, Input Line Controllers (ILC), memory Buffers, Output Line Controllers (OLC), output channel data and signaling buses, Central Processing Unit (CPU) and Service RAM. Service RAM holds CPU programs, routing tables and other software components, necessary for the router operation. During the initiation sequence, CPU is allocating Buffer memory blocks for packets and programs ILC with the next available packet block addresses. To avoid input blocking, each ILC has more than one packet block address slots (input queue). Upon completion of one packet, ILC can immediately start receiving the next packet into the next available packet memory block. ILC stores data received from the input channel, using Direct Memory Access or Bus Mastering or a similar technique without CPU involvement. Upon receiving of pre-programmed number of data elements (a packet header), ILC generates an interrupt signal, which goes to CPU to inform that a packet header is ready for the processing. Upon receiving the signal, CPU is accessing the packet header data and makes a decision on the further packet route. If the packet destination is known, CPU programs OLC with the packet memory block address, packet length and possible delay before the transmission (a packet reference). If the output channel speed is less or equal than the input channel speed, the delay is equal to zero, otherwise it is determined by the speed ratio and the length of remaining portion of the packet, arriving to the Buffer. To avoid output blocking and reduce the number of interrupts, each OLC has more than one packet memory block address slots. These slots can be named the output queue. After receiving the whole packet, ILC removes the packet memory block address from the input queue. If the number of elements in the input queue is less than pre-programmed "low mark", ILC

generates an interrupt signal, requesting CPU for more free packet memory block addresses. CPU gets the required number of addresses from the list of available packet memory blocks and places them into the input queue. If OLC has a ready packet reference in the output queue, it sends Request To Send (RTS) signal to the output channel apparatus. Once the output channel is free, it responds with Clear To Send (CTS) signal back to the OLC. OLC transmits packet data from Buffer to the channel, using Direct Memory Access or Bus Mastering or a similar technique without CPU involvement. Upon completion or failure of the transmission, OLC removes packet reference from the queue. If the number of elements in the output queue is less than pre-programmed "high mark", OLC sends an interrupt signal to CPU, requesting for more packets to send. CPU places the packet memory block addresses into the list of available packet memory blocks and performs other actions like notifying the sender about the failure. To avoid head of line blocking, any OLC can access any Buffer, so if one of the channels is busy, others can successfully transmit.

Single Processor Router

The single processor router is implemented with the following parameters:

1. Memory bus speed is 400MHz
2. 8 bits per byte

The results of simulation are shown in Fig. 6 and Fig. 7.

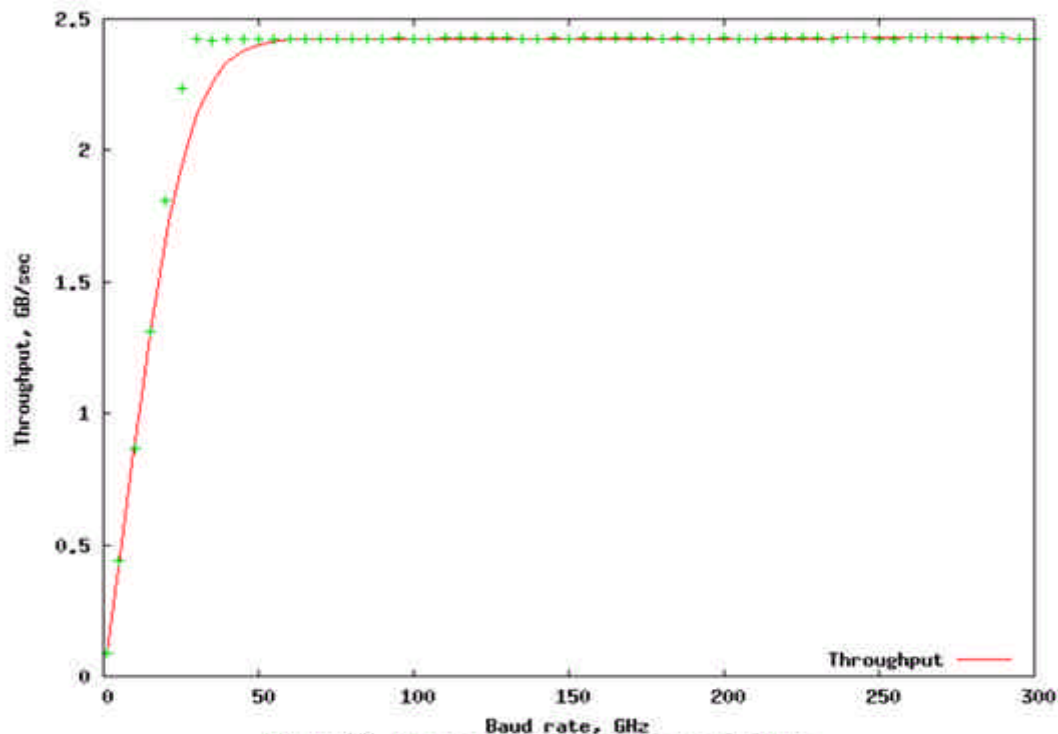


Figure 6 : Throughput of Uniprocessor Router

The maximum throughput reached is about 2.4GB/sec.

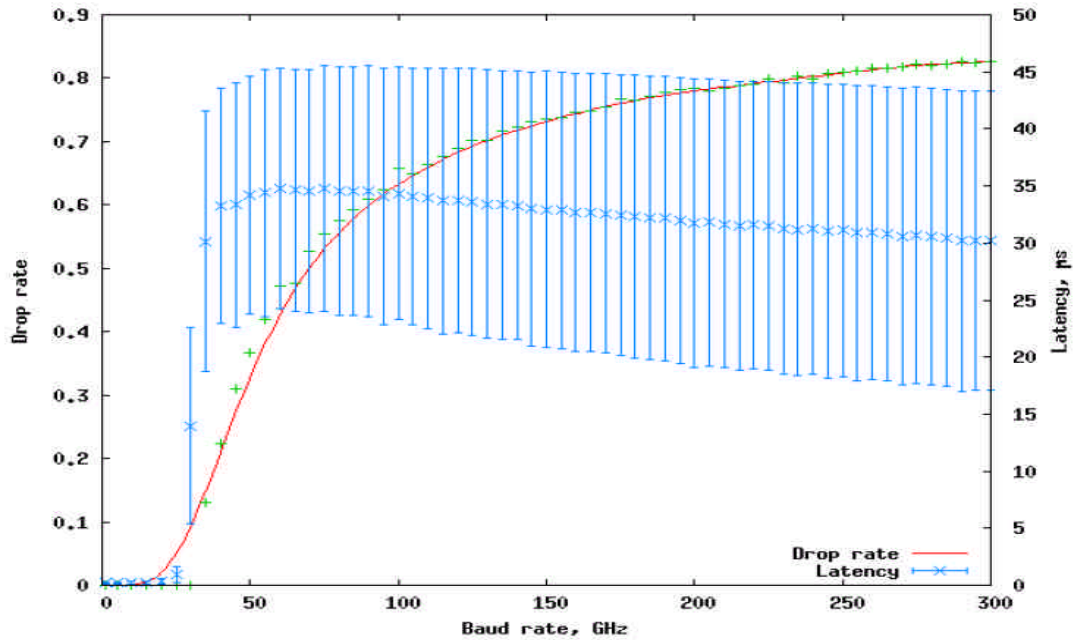


Figure 7 : Latency and Drop Rate for Uniprocessor Router

Once the router has reached the maximum throughput, the queues get overflown and packet drops occur. At the same time the latency increases as well as latency variance. At the higher baud rates the router starts failing to place the packet into input queue as its CPU can not handle interrupts at required rate. This reduces the number of processed packets and the observed latency slightly decreases.

Combinatorial Router

The combinatorial 7,7,3,3,1 router is implemented with the same parameters:

1. Memory bus speed is 400MHz
2. 8 bits per byte

The results of simulation are shown in Fig. 8 and Fig. 9.

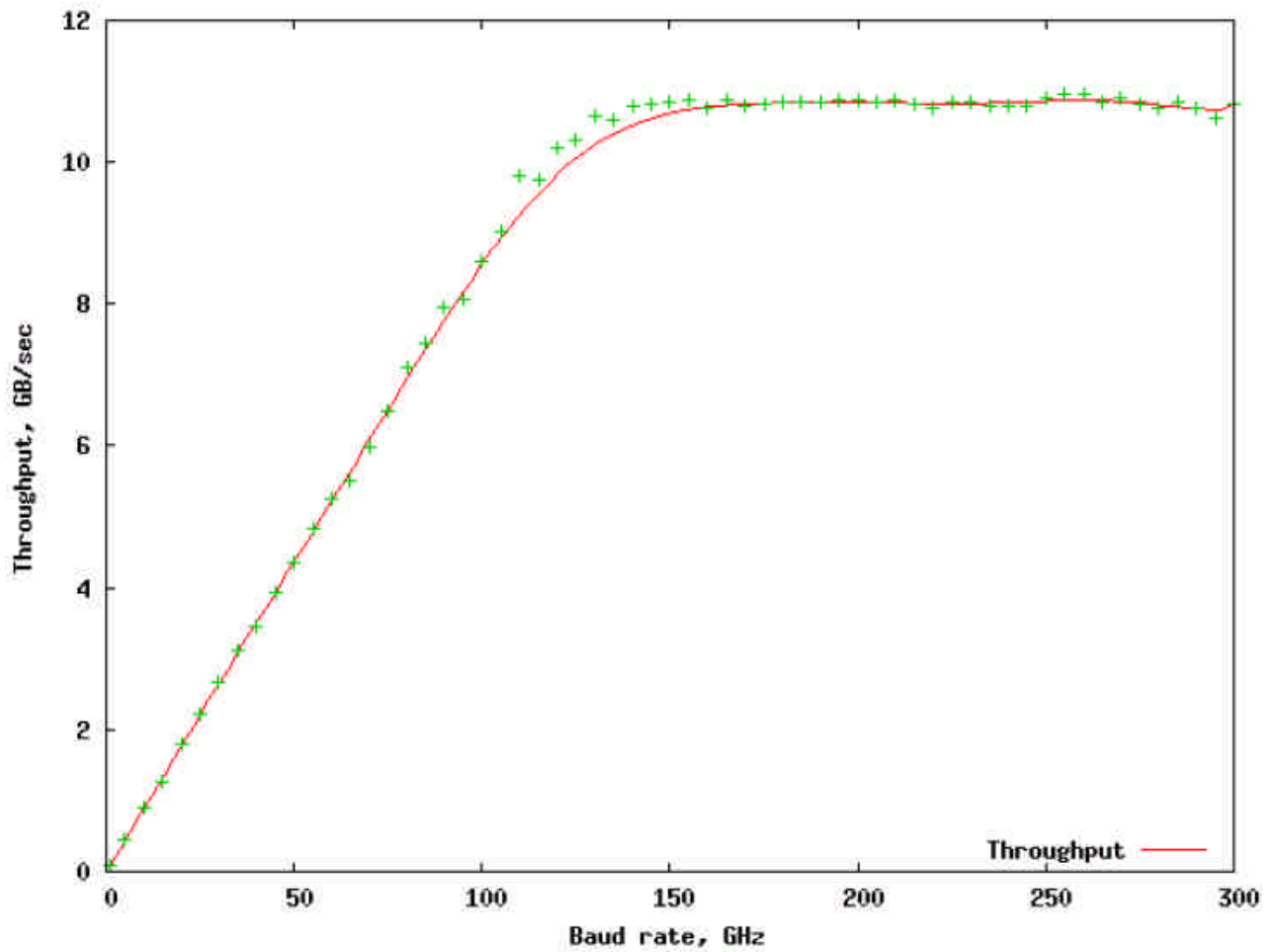


Figure 8 : Throughput of Combinatorial Router

The maximum throughput reached is 10.8 GB/sec, which presents 450% throughput increase using the same type of memory modules.

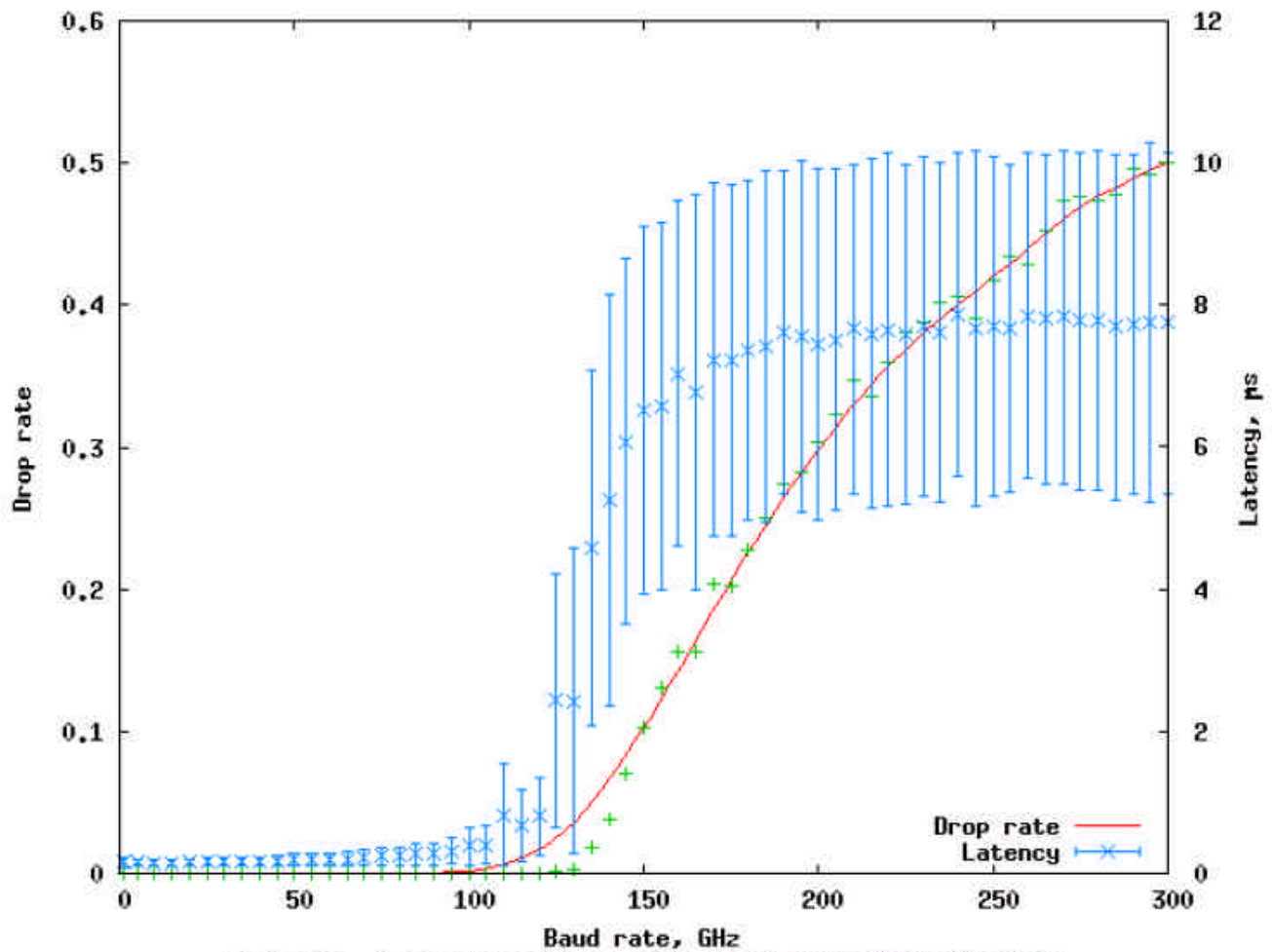


Figure 9 : Latency and Drop Rate for Combinatorial Router

At 300 Gbit/sec we have not reached yet the limit of computational capabilities of combinatorial router as the latency still growing.

Increasing Reliability and Performance

In order to provide basic routing or switching capabilities, it is enough that the pairwise property of combinatorial design holds. To obtain special functionality we may impose additional requirements to the design. Assume that we want to have a more redundant or a faster device. Higher values of λ could serve that purpose.

Let us create a router, which should:

1. Handle heavy traffic, which exceeds internal PE bus bandwidth
2. Sustain one PE failure per channel with no performance degradation

Both properties can be satisfied with a BIBD having $\lambda=3$

Let us consider (15, 15, 7, 7, 3) BIBD. Such a design may consist of the following set of blocks:

- $b_1 = \{ 1, 2, 3, 4, 5, 6, 7 \}$
- $b_2 = \{ 1, 2, 3, 8, 9, 10, 11 \}$
- $b_3 = \{ 1, 2, 3, 12, 13, 14, 15 \}$
- $b_4 = \{ 1, 4, 5, 8, 9, 12, 13 \}$
- $b_5 = \{ 1, 4, 5, 10, 11, 14, 15 \}$
- $b_6 = \{ 1, 6, 7, 8, 9, 14, 15 \}$
- $b_7 = \{ 1, 6, 7, 10, 11, 12, 13 \}$
- $b_8 = \{ 2, 4, 6, 8, 10, 12, 14 \}$
- $b_9 = \{ 2, 4, 7, 8, 11, 13, 15 \}$
- $b_{10} = \{ 2, 5, 6, 9, 11, 12, 15 \}$
- $b_{11} = \{ 2, 5, 7, 9, 10, 13, 14 \}$
- $b_{12} = \{ 3, 4, 6, 9, 11, 13, 14 \}$
- $b_{13} = \{ 3, 4, 7, 9, 10, 12, 15 \}$
- $b_{14} = \{ 3, 5, 6, 8, 10, 13, 15 \}$
- $b_{15} = \{ 3, 5, 7, 8, 11, 12, 14 \}$

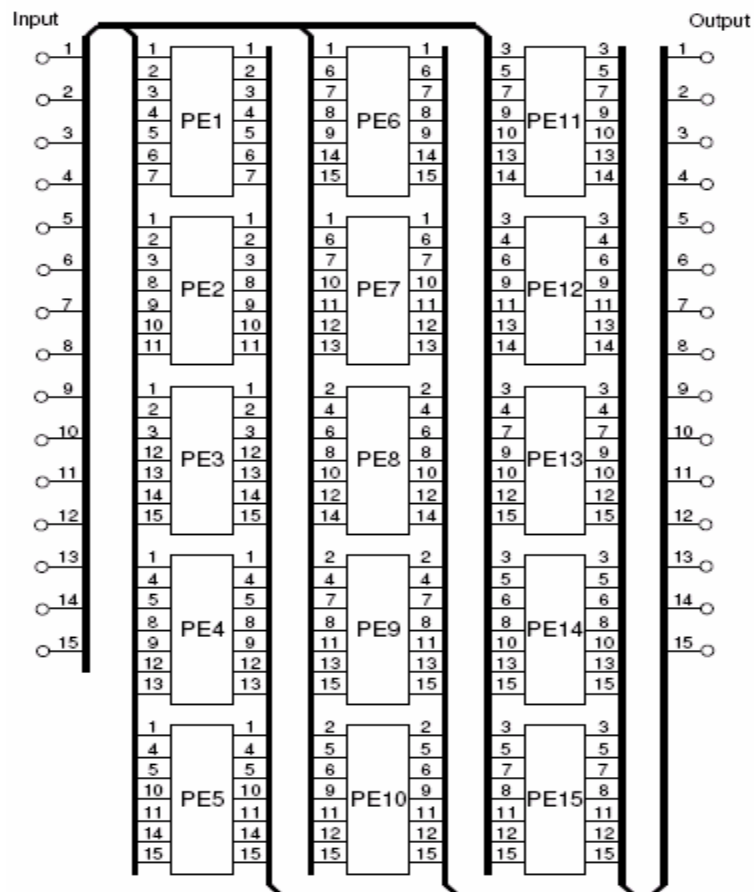


Figure 10 : (15, 15, 7, 7, 3) Combinatorial router architecture