ISCA-21 -- International Symposium on Computer Architecture
April 18-21, 1994, Chicago, Illinois

METRO: A Router Architecture for High-Performance, Short-Haul Routing Networks

André DeHon
<andre@ai.mit.edu>
Frederic Chong <ftchong@ai.mit.edu>
Matthew Becker <beefe@ai.mit.edu>
Eran Egozy <eran@media.mit.edu>
Henry Minsky <hqm@ai.mit.edu>
Samuel Peretz <sam@monitor.com>
Thomas F. Knight, Jr. <tk@ai.mit.edu>
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139

Abstract

The Multipath Enhanced Transit Router Organization *(*METRO*) is a flexible routing architecture for high-performance, tightly-coupled, multiprocessors and routing hubs. A* METRO *router is a dilated crossbar routing component supporting half-duplex bidirectional, pipelined, circuit-switched connections. Each* METRO *router is self-routing and supports dynamic message traffic. The routers works in conjunction with source-responsible network interfaces to achieve reliable end-to-end data transmission in the presence of heavy network congestion and dynamic faults.* METRO *separates the fundamental architectural characteristics from implementation parameters. Simplicity of routing function coupled with freedom in the implementation parameters allows* METRO *implementations to fully exploit available technology to achieve low-latency and high-bandwidth. We illustrate the effects of this implementation freedom by summarizing the performance which various* METRO *configurations can extract from some modern CMOS technologies. Included in our illustrations is* METROJR-ORBIT*, a minimal instance of the* METRO *architecture we constructed in a* 1.2μ *gate-array technology.*

Acknowledgments: This research is supported in part by the Advanced Research Projects Agency under contracts N00014-87-K-0825 and N00014-91-J-1698. This material is based upon work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

The METRO architecture is optimized for low-latency, fault-tolerant node-to-node communications in tightly-coupled computing environments such as high-performance multiprocessors and network routing hubs. The basic routing function employed is simple yet powerful. Simplicity is achieved by pushing the responsibility for message buffering, congestion handling and fault handling outside of the network. The simplicity of function in METRO is analogous to contemporary trends in processor design: Functional complexity is reduced to speed primitive operation. While each router's behavior is simple, an assembled network behaves in a sophisticated manner, requiring little assistance from the source node to efficiently handle most cases of congestion control and fault avoidance. The METRO architecture is defined in a general manner which separates the fundamental characteristics of the architecture from implementation parameters. Freedom in choosing implementation parameters provides flexibility to customize particular METRO implementations to suit target applications and available technologies. Combining simple routing with freedom to optimize to the target technology, METRO implementations can achieve very high-performance supporting both low-latency and high-bandwidth communications.

Each METRO router is organized as a dilated crossbar routing component supporting half-duplex bidirectional, pipelined, circuit-switch connections. METRO routers may serve as the principal building blocks for a wide range of indirect, multistage routing networks. METRO routers feature:

- High-Bandwidth, Pipelined Circuit-Switched Operation
- Self Routing
- Low-Latency Connection Establishment and Data Transmission
- (Unlimited) Variable Length Message Support
- Stochastic, On-line Fault Avoidance Support
- Flexible Routing Configuration Options
- Fast Path Reclamation
- Cascade Support for Building Wide Routers from Narrow Routers
- Multiple-TAP Scan Architecture
- Integrated Support for Fault-Localization and Fault-Avoidance

Section 2 describes the communications domain where METRO is most suitable. In Sections 3 through 5 we review the terminology used in describing METRO-style routing components and describes the basic operation and architecture of METRO routing components. Section 6 describes sample implementations of the METRO architecture. Section 7 reviews several contemporary routing networks and switches for comparison with METRO. In Section 8, we review the key benefits of the METRO architecture.

2 Application Domain

Low Latency Communications In many applications, including general-purpose MIMD multiprocessing, cross network latency is the critical factor limiting application performance. In applications where we can only extract limited parallelism, cross network latency acts to limit the speedup achievable using parallel processing. In particular, an application with p operations which may proceed in parallel on each cycle, running on a machine with latency l can, on average, execute $\frac{p}{l+1}$ operations per cycle. Only in cases where parallelism is much larger than the number of nodes (n) employed to speedup the application ($p > (n \cdot l)$), is application performance decoupled from network latency. In cases where application parallelism is limited compared to the number of available processors, the achievable speedup due to parallel processing is directly limited by cross network latency.

Short-Haul Networking We refer to METRO as a building block for *short-haul* networks to distinguish it from the networking technologies in common use for Local- and Wide-Area Networking (LAN/WAN). LAN/WAN technologies allow the construction of long-haul networks for interconnecting loosely coupled nodes. Long-haul networks are generally used to link together computers in distributed computing environments.

The interconnection distances between computers or routing hubs in long-haul networks are necessarily large due to the physical separation of machines. In these cases, the latency between nodes is dominated by the latency traversing the physical interconnect between nodes. This has two effects: (1) the relative importance of low-latency switching is small since the node-to-node travel time is dominated by the latency required to traverse the long, physical interconnection media; (2) since the time required to inject an entire message into the network can be small compared to the time required to deliver it, interconnection bandwidth is utilized most efficiently by packet switching where an interconnection channel is allocated to a message for only long enough for the message to be injected into the interconnect. Consequently, long-haul networks employ packet switching and are not significantly impacted by the delay through switching elements.

In tightly-coupled networks, the interconnection delays are much smaller and routing latency has a much larger proportional impact on end-to-end message latency. METRO is optimized for these configurations and applications where low end-to-end message latency is critical. In these tightly-coupled situations, the time required to inject a message is often large compared to the end-to-end interconnect latency. We use the term *short-haul* networking to denote network technologies optimized for the case where end-to-end interconnection latency is comparable to or smaller than the message injection time.

Since message injection time dominates interconnect transit latency in short-haul networks, there is little negative impact on usable network bandwidth if we dedicate resources to a single message for the duration of its communication. Consequently, we can employ circuit switching interconnect techniques without sacrificing significant performance. In these short-haul network environments circuit switching offers several advantages:

1. Simpler management which allows fast operation – The lack of buffering in the network simplifies the operation performed by each router allowing faster implementations.
2. Fast, deterministic acknowledgements – The dedicated network connection allows low-overhead, fast and efficient end-to-end verification of message delivery.
3. Stateless network – No messages ever exist solely

in the the network. Consequently, it is possible to stop network operation at any point in time without losing or duplicating messages. This feature is useful in gang-scheduled, time-shared, multiprocessors, allowing context switches to occur without incurring overhead to snapshot network state.

METRO employs circuit switching to simplify and speed switching through the network and allow efficient implementation of reliable message protocols.

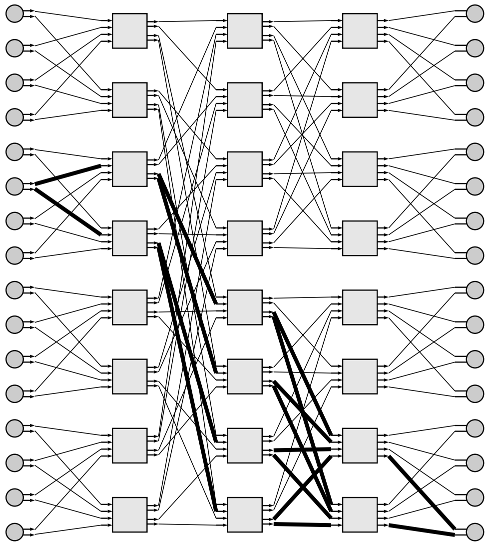
Note that these two classes of networking technologies optimize for bandwidth utilization and latency differently based on their domains of application. There is clearly a region of operation between the extremes of short- and long-haul networking. In this region there is a tradeoff between usable bandwidth and end-to-end latency. The choice of an optimization target in this region is often application dependent.

Network Organization A collection of METRO routers is typically used to construct a *multipath*, multistage network. Multiple stages allow the routers to route data between a source and a destination through a logarithmic number of routing components. In a multibutterfly-style network, each stage in the routing network subdivides the set of possible destinations into a number of distinct classes determined by the radix of the routing components. Successive stages recursively subdivide the destination class until each output node from the network is uniquely identified. Dilated routing components give rise to multiple independent paths through the network. The multiple paths in the network increase available bandwidth, decrease congestion, and provide tolerance to link and router faults. Figure 1 shows a small network of this genre.

[16] and [23] present some of the basic theory on multibutterflies, a well-understood example of multipath networks. Network construction, utilization, and performance issues are further explored in [10] [15] [2] [3]. Fat-Tree networks [17] [14] are another class of multistage, multipath networks which can be built using METRO routing components. [7] details issues involved in constructing such networks.

3 Terminology and Overview

A METRO router is a dilated crossbar routing component which supports half-duplex bidirectional, pipelined, circuit-switched connections. Each METRO router is self-routing and supports dynamic message traffic. METRO works in conjunction with a source-responsible network interface to achieve reliable end-to-end data transmission in the presence of heavy network congestion and dynamic faults. Fault-localization and reconfiguration capabilities provided through the scan interface allow the router to perform efficiently in the presence of static faults.



A multibutterfly style network constructed from 4×2 (inputs$\times$radix), dilation-2 METRO routers and 4×4 dilation-1 routers. Each of the 16 endpoints has two inputs and outputs for fault tolerance. Similarly, the routers each have two outputs in each of their two logical output directions. As a result, there are many paths between each pair of network endpoints. Paths between endpoint 6 and endpoint 16 are shown in bold.

Figure 1: 16×16 Multipath Network

General Terminology A *crossbar* has a set of i inputs and a set of o outputs and can connect any of the inputs to any of the outputs with the restriction that only one input can be connected to each output at any point in time. A *dilated crossbar* has groups of outputs which are considered equivalent. We refer to the number of outputs which are equivalent in a particular logical direction as the crossbar's *dilation*, d. We refer to the number of logically distinct outputs which the crossbar can switch among as its *radix*, r.

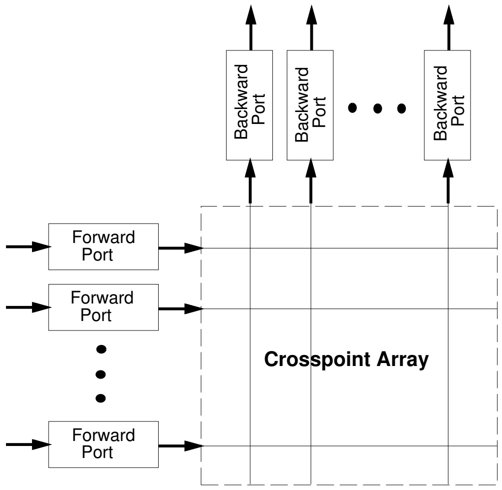
A *circuit-switched* routing component establishes connections between its input and output ports and forwards the data between inputs and outputs in a deterministic amount of time. Notably, there is no storage of the transmitted data inside the routing component. In a network of circuit-switched routing components, a path from the source to the destination is locked down during the connection: the resources along the established path are not available for other connections during the time the connection is established. In a *pipelined, circuit-switched* routing component, all the routing components in a network run synchronously from a central clock and data takes a small, constant number of clock cycles to pass through each routing component.

A crossbar is said to be *self routing* if it can establish connections through itself based on signalling on its input channels. That is, rather than some external entity setting the crosspoint configuration, the router configures itself in response to requests which arrive via the input channels. A router is said to handle *dynamic message traffic* when it can open and close connections as messages arrive independently from one another at the input ports.

METRO **Terminology** When connections are requested through a METRO router, there is no guarantee that the connections can be made. As long as the dilation of the router is smaller than the number of input channels into a router (*i.e.* $d < i$), it is possible that more connections will want to connect in a given logical direction than there are logically equivalent outputs. When this happens, some of the connections must be denied. A connection request rejected for this reason is said to be *blocked*. The data from a blocked connection is discarded and the source is informed that the connection was not established.

Once a connection is established through a METRO router, it can be *turned*. That is, the direction of data transmission can be reversed so that data flows from the original destination to the original source. This capability is useful for providing rapid replies between two nodes and is important in effecting reliable communications. METRO provides *half-duplex*, bidirectional data transmission since it can send data in both directions, but only in one direction at a time.

Since connections can be turned around and data may flow in either direction through the crossbar router, it can be confusing to distinguish input and output ports since any port can serve as either an input or an output. Instead, we will consider a set of *forward ports* and a set of *backward ports*. A forward port initiates a route and is initially an input port while a backward port is initially an output port. The basic topology for a crossbar router assumed throughout this paper is shown in Figure 2.



The basic router has i forward ports and $o = r \cdot d$ backward ports. Any forward port can be connected through the crosspoint array to any backward port. The arrows indicate the initial direction of data flow.

Figure 2: Basic Router Configuration

4 Router Operation

Overview In operation, a network endpoint will feed a data stream of an arbitrary number of words into the network at the rate of one word per clock cycle. The first few data words are treated as a routing specification and are used for path selection. Subsequent words are pipelined through the connection, if any, opened in response to the leading words. When the data stream ends, the endpoint may signal a request for the open connection to be reversed or dropped. When each router receives a reversal request from the sender, the router returns status and checksum information about the open connection to the source node. Once all routers in the path are reversed, data may flow back from the destination to the source. The connection may be reversed as many times as the source and destination desire before being closed. End-to-end checksums and acknowledgments ensure that data arrives intact at the destination endpoint. When a connection is blocked due to contention or a data stream is corrupted, the source endpoint retries the connection.

Stochastic Path Selection When a connection is opened through a router, there may or may not be outputs available in the desired logical output direction.

If there is no available output, the router discards the remaining bits associated with the data stream. When the connection is later turned, the router STATUS word returned by the routing node informs the source that the message was blocked at the router. When exactly one output in the desired direction is available, the router switches the connection through that output. When multiple paths are available, the router switches the data to a logically appropriate backward port selected *randomly* from those available.

This random path selection is the key to making the protocol robust against dynamic faults while avoiding the need for centralized information about the network state and keeping the routing protocol simple. When faults or congested regions develop in the network, the source detects the occurrence of a failed or damaged connection by monitoring the router status and the acknowledgment, if any, from the destination. The source then knows to resend the data. Since the routing components select randomly among equivalent outputs at each stage, it is highly likely that the retry connection will take an alternate path through the network, avoiding the newly exposed fault or hot spot. Source-responsible retry coupled with randomization in path selection guarantees that the source can eventually find an uncongested, fault-free path through the network, provided one exists. The number of retries required, in practice, is small. The random selection also frees the source from knowing the actual details of the redundant paths provided by dilated components in the network. Random selection among equivalent available outputs is an extremely simple selection criterion to implement in silicon and can be implemented with little area and considerable speed. Random selection also requires no state information not already contained on the individual routing component.

5 Architecture

The METRO architecture contains many features which provide for flexible implementation and application. This section describes many of the key features including:

1 Pipelining options for achieving high bandwidth
2 Cascading options for building wide data paths
3 Configuration options for operational flexibility
4 Connection reversal for low-latency replies
5 Fast block recovery for fast stochastic path search
6 Scan support for fault localization and masking

5.1 Feature Descriptions

Configurable Dilation In many dilated network configurations, it is desirable to utilize routers with differing dilations in some stages of the network. For instance, in the network shown in Figure 1, the final stage uses dilation-1 METRO routers while the earlier stages use dilation-2 routers. The dilation-1 routers in the final stage allow the network shown to tolerate the complete loss of any router in the final stage without isolating any endpoints from the network. The dilation-2 routers in the earlier network stages give the network its multipath nature allowing the network to tolerate congestion and faults in the earlier network stages. To facilitate this kind of network construction from a single router implementation, the effective dilation of a METRO router may be configured to any power of two up to the implementation specified limit, max_d.

Data Idle There are several cases in system use or in basic router behavior where it is necessary to keep a connection open while no data is available for transmission. DATA-IDLE is a word passed through the router just like data but which is outside of the normal band of data word encodings. There are two major uses of this designated token:

1. The protocols layered on top of a network of METRO routing component may use DATA-IDLE when it is not possible to deterministically specify when data will be available to send with a message. *e.g.* In a low-latency, distributed-memory multiprocessor, the sending endpoint might turn the connection around to get a fast reply to a read request. The delay associated with preparing the read data for the reply may depend on whether the data item requested currently resides in the remote node's cache or in main memory. The remote node can send DATA-IDLE words to fill the variable delay associated with data retrieval.

2. The routing components themselves use DATA-IDLE in cases where the number of cycles between events can vary within a given implementation. This feature is employed by the various forms of pipelining in METRO (*i.e.* Variable Turn Delay and Data Pipelining) to make implementation and application pipelining details transparent to the source endpoint.

Connection Reversal After opening a connection through a series of routers, the source will generally want a reply back from the destination to verify that the message was received. Often the source will also want to get a response to the connection request. By sending a designated control word, TURN, the established path is reversed in a pipelined manner so that data can return from the original destination back to the original source. During the pipeline delays associated with reversing the connection, each routing component has the opportunity to inject information into the return data stream about the status of the connection and the integrity of the data transmitted; this information is useful to the source in identifying and localizing errors. This path reversal allows METRO to implement efficient, low-latency, request-reply operations such as distributed-memory read operations. Once the connection is established and the request is transmitted, the reply data can stream back along the path opened by the request without incurring any additional latency to acquire a new connection through the network.

To allow arbitrary protocols to be layered on top of the basic METRO router protocol, connections may be turned back to the forward direction. Any number of data transmission reversals may occur during a single connection. It is always the prerogative of the transmitting end of the connection to signal a connection reversal.

Pipelining Data Through Routers If we can clock data between routing components faster than we can route data through a routing component, we can often achieve higher bandwidth by allowing the data to take multiple clock cycles to traverse the routing component. The only place where pipelining affects the routing protocol is when connections are reversed. Following a TURN the number of delay cycles before return data is available will depend on the number of pipeline stages through the routing switch since it is necessary to flush the router's pipeline in the forward direction, then fill it in the reverse direction before reverse data can be forwarded. DATA-IDLE words serve to hold the connection open during the pipeline delays.

Pipelined Connection Setup The longest latency operation inside a routing component is often connection establishment when arbitration must occur for a backward port. If we require that the connection setup occur in the same amount of time, or the same number of pipeline cycles, as the following data is routed through the component, the latency associated with connection setup will be passed along to become the latency associated with data transfer through the router. Alternately, we can allow more time, generally more pipeline cycles, for connection setup than for data transmission. When we allow this, each router will consume a number of words equal to the difference between the setup pipeline latency and the transmission pipeline latency from the head of each data stream. When constructing the route header, we pad the header to account for the words which each router strips from the head of the data stream during connection setup.

For example, consider a router which can route data along an established path in a single cycle, but requires three cycles to establish a new connection. The router would consume the first two words from the head of each routing stream before it was able to establish a connection and forward the remainder of the data out the allocated backward port.

Variable Turn Delay For long interconnect, METRO routers pipeline data across the wires interconnecting routers. In some network systems, it is generally not possible or desirable to make all the connections between routers equally long. In particular, closer routers should be able to take advantage of the fact that interchip signalling may occur faster (*i.e.* less pipeline delays between routers). Since each port on a single router connects to a different router it may be the case that the length of the wires connected to each port differ. For this reason, METRO provides the ability to define the number of pipeline stages associated with each port connection separately. During the time the router is waiting for a response from the turn by the attached routing component, the waiting router will send DATA-IDLE words to hold the connection open until it has reverse data to forward.

One important assumption made here is that we can model the wire between two components as a number of pipeline registers. How one assures that this assumption is satisfied is implementation dependent. With a properly series-terminated point-to-point connection between routers, we do not have to worry about reflections and settling time on the wire. The wire will look, for the most part, like a time-delay. The necessary trick is to make the time-delay approximate an integral number of clock cycles so that it does look like a number of pipeline registers. A brute force way to achieve this is to carefully control the length and electrical characteristics of the wires between components. Alternately, we can use adjustable delay in the pad drivers themselves to adjust the chip-to-chip delay sufficiently to meet the assumption. See Chapter 6 of [9] for further discussion.

Path Reclamation – Fast and Detailed When a router is unable to route a connection due to the lack of availability of an appropriate backward port, the connection is blocked. There are two ways which a METRO router can handle this situation. The router could wait for a path reversal request and shut the connection down after returning information to the source about the blocked state of the connection. Alternately, the router could immediately begin propagation of a backward drop request back to the source using a designated backward control bit (BCB). The earlier behavior provides the source with sufficient information to determine at exactly which router the blocking occurred and whether or not any errors occurred prior to that point. The latter behavior releases the resources held by the blocked connection rapidly while only informing the source of the routing stage in which the blocking occurred.

METRO routers allows each forward port to independently select between these two modes. The tradeoff between fast path reclamation and detailed information gathering can be handled dynamically while the router is in use. Note that the mode of path reclamation is solely determined by the configuration of the forward port on the router at which the blocking occurred. If we were to disable fast path reclamation on a single router in the network, only those connections which block at that particular router would hold the connection for a detailed reply. Connections which blocked before or after the router would recover using the fast path reclamation. This allows the system to select portions of the network (*e.g.* a particular stage in the routing network or a particular routing component) for gathering detailed information while blocking in the remainder of the network is signalled via fast reclamation for efficiency.

Scan Support METRO integrates extensive scan support using an IEEE 1149-1.1990 [4] compliant Test Access Port (TAP) extended to support multiple TAPs on each component (MultiTAP) [8]. The multiTAP support allows METRO increased tolerance to faults in the scan paths. The TAPs provide a convenient mechanism for setting METRO's mostly static configuration options.

In addition to the normal boundary-scan facilities associated with an IEEE 1149 TAP, METRO provides fine-grain facilities for on-line fault-diagnosis. In particular, many of the system in which an METRO router might be used are large and it would be inconvenient or impractical to bring the entire machine down to run traditional boundary-scan style diagnostics. For this reason, METRO routers allow each port to be treated separately for purposes of testing and reconfiguration. Each port can be disabled, removing it from the set of resources in use by the system. The topology and redundancy provided by typical METRO networks allow the network to continue functioning with some resources disabled. Once a port is disabled, boundary and internal scan tests can be applied exclusively to the disabled port or ports while the rest of the router functions normally. This allows a forward-backward port pair, a routing component, or some region of a network, to be isolated from normal operation for testing. Once the fault region is identified, the faulty-region can be left disabled and the rest of the system returned to service. Disabled faults are *masked*, in that their faulty behavior can no longer corrupt message traffic.

Router Width Cascading To allow wide routers to be built from routing components with narrow datapaths, METRO provides features to facilitate *cascading* routers. A router with an effectively wider data path can be achieved by using multiple METRO routers in parallel. Routing components often tend to be pin-limited. Width cascading reduces the competition for pins between datapath width and the number of forward and backward ports supported on a single IC. For any fixed number of IC pins, this allows the IC to support more forward and backward ports without sacrificing network datapath width. For a target logical router size, this allows logical routers to be constructed from primitive router ICs with less pins, and hence less expense.

In width cascading, two or more routers need to behave identically in terms of how they handle connections. METRO provides two hooks to ensure identical connection handling: *shared randomness* and a wired-AND pull-up.

For *shared randomness*, the routers receive their random bits from off chip. These bits are used along with the connection requests to decide the assignment of connection requests to backward ports. As long as the connection requests and shared random bits are identical for the set of cascaded routers, the cascaded routers will allocate identically in the non-faulty case. Only in faulty cases should the connection requests, and hence allocations, ever differ.

To handle the faulty case where the routing header is corrupted or a router behaves abnormally, METRO enforces a simple consistency check on each backward port. Each backward port has pull-up signal ($\overline{\text{IN-USE}}$) that indicates when the port is not in use. By con-

Variable	Function
sp	Number of Scan Paths ($sp \geq 1$)
w	Bit Width of Data Channel ($w \geq \log_2(o)$)
max_d	Maximum Dilation ($max\_d = 2^n$, $max\_d \leq o$)
i	Number of Forward Ports ($i = 2^n$)
o	Number of Backward Ports ($o = 2^n$, $o \geq max\_d$)
ri	Number of Random Inputs ($ri \geq 1$)
hw	Number of Header Words Consumed Per Router ($hw \geq 0$)
dp	Number of Data Pipestages Inside Router ($dp \geq 1$)
max_vtd	Maximum Number of Delay Slots Available for Variable Turn Delay ($max\_vtd \geq 0$)

Table 1: METRO Architectural Parameters

necting this signal across the cascaded routers in a wired-AND configuration, the backward ports can detect when they disagree about an allocation. Since any disagreement is necessarily an error, as soon as the disagreement is noticed, the connection is shut down on all attached routers so that the fault is contained.

There is still a need for end-to-end checking to assure that messages arrive intact. Though highly improbable, there still exist some cases in which a faulty allocation may go unnoticed by the shared pull-up. The shared pull-up facility significantly limits the negative effects that can be caused by the faults which are most likely to occur.

To avoid the need for additional resources to generate the random bits, the METRO architecture requires each component to generate one random output bit stream. Each METRO router typically has multiple random input bits to support the stochastic path selection.

5.2 Architecture Parameters

The METRO architecture encompasses a large family of potential router implementations. All METRO implementations share the same basic function and protocol. Individual METRO routers may be specialized for application and implementation technology by the appropriate selection of architectural parameters. Many of these parameters were introduced in the previous section. Table 1 summarizes these parameters.

Option	Instances	Bits per Instance
Port On/Off	$i + o$	1/port
Off Port Drive Output	$i + o$	1/port
Turn Delay	$i + o$	$\lceil log_2(max\_vtd) \rceil$/port
Fast Reclaim	$i + o$	1/port
Swallow	i	1/forward port
Dilation (d)	1	$log_2(max\_d)$/router

Swallow is only relevant on components where $hw = 0$.

Table 2: METRO Configuration Parameters

5.3 Configuration Options

Each METRO router has many options which can be selected each time the component is used as well as while the component is in use. All of these options are configurable under scan control from a TAP. Table 2 summarizes the configurable options available for any METRO router. Variable turn delay, dilation, and swallow would typically remain constant throughout operation. Port enables and fast reclamation may be reconfigured during operation as discussed in Section 5.1.

6 Implementations and Prospects

6.1 Single Router Performance

RN1, a direct ancestor of the METRO architecture implemented in a 1.2μ CMOS process [20], ran with an internal latency under 15 ns. RN1 supported 8 forward and backward ports ($i = o = 8$), byte wide datapaths ($w = 8$) and both dilation-1 and dilation-2 routing. RN1 was designed so that each routing stage was only a single pipeline stage in the network. Unlike METRO it did not treat the interconnect as a separate set of pipeline stages from the internal logic. Consequently, RN1 was limited to about 50 MHz operation [19].

METROJR is a minimal implementation of the METRO architecture with $i = o = w = 4$, $hw = 0$, $dp = 1$, and $max\_d = 2$. To date METROJR has been described entirely in a high-level language and synthesized to a standard ASIC library using the Synopsys Design Compiler [22]. Without pipelining connection setup, the critical path between input and output datapath registers due to connection allocation contains 25 gate delays.

An initial implementation of METROJR was done through Orbit Semiconductor under their Encore! program. METROJR was implemented as a 15K gate array in a 1.2μ CMOS process. Orbit indicates the components will run at 40-50 MHz. Running at 40MHz with interconnect accounting for only a single pipeline stage, METROJR-ORBIT has a 50 ns router-to-router latency and a 25 ns nibble (4-bit) latency. Since METROJR supports width cascading, multiple METROJR-ORBIT components can be cascaded to achieve higher data bandwidth.

Based on the Synopsys results, we are encouraged that a CMOS standard-cell implementation of METROJR can easily achieve higher performance. Using a 0.8μ effective gate-length CMOS process with 300 to 400 ps gates, we expect a 100 MHz implementation of METROJR can be realized relying primarily on standard-cell synthesis and layout. Of course, to support these speeds, custom i/o pads and clocking will be required to augment the standard-cell logic. Suitable low-voltage swing, matched-impedance i/o pads have been demonstrated in this process technology [11] and exhibit sufficiently low latency and power consumption to make such an implementation feasible. By taking advantage of connection setup pipelining, even lower data transfer latency can be achieved without going to a full-custom design.

Simulation of full-custom METROJR implementations indicate that 200 MHz implementations in the same 0.8μ CMOS process are feasible without connection setup pipelining. By adding a single connection setup pipeline stage ($hw = 1$), 400-500MHz implementations also look promising.

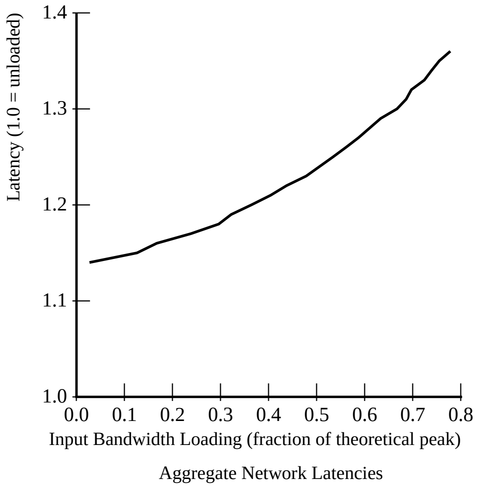
Table 3 summarizes a few potential implementations of the METRO architecture.

6.2 Aggregate Performance

As an example of how loading affects METRO networks, Figure 3 shows the effective latency of a 3-stage, multipath network under varying loads. Earlier work based around the routing protocol which evolved to become the METRO routing protocol shows that performance degrades robustly in the face of faults [2] [3].

7 Comparisons

Table 5 shows the performance of several contemporary routing networks in use for multiprocessor interconnect and network routing hubs. The DEC GIGAswitch is an FDDI routing hub and is representative of long-haul networking technologies. The KSR-1 and CM-5 are general-purpose, commercial, MIMD parallel processors. The MIT J-Machine is an academic research machine. The Caltech MRC is a full-custom, academic, mesh router employed by many mesh-based multiprocessors. Mercury/RACE is a commercial offering for real-time, parallel processing. The INMOS C104 is commercial, packet-switched crossbar switch intended for use in multiprocessors or routing hubs. Table 5 includes an estimate ($t_{20,32}$) for the unloaded network latency required deliver a 20-byte message across the network in a 32-node configuration for comparison with Table 3. As evidenced by the application example, $t_{20,32}$, even the minimal gate-array implementation of METRO compares favorably with the existing field of routing technologies.



The graph shows latency versus network loading for randomly distributed, 20-byte message traffic on a 3-stage network constructed from 8-bit wide, radix-4 METRO routers. This simulation models a parallelism limited case where processors stall waiting for message completion. The first two network stages are configured in dilation-2 mode, while the last stage is configured in dilation-1 mode. Like the network shown in Figure 1, each endpoint has two connections entering and leaving the network. Each endpoint was restricted to only use one of its entering network ports at a time, but can handle simultaneous traffic on both network output ports. The unloaded message latency is 28 clock cycles from message injection to acknowledgment receipt.

Figure 3: Aggregate Latency Performance Example

Architecture Instance	Technology	t_{clk}	t_{io}	t_{stg}	t_{bit}	Example $stages$	$t_{20,32}$
METROJR-ORBIT	1.2μ Gate Array	25 ns	10 ns	50 ns	25 ns/4 b	4	1250 ns
2-cascade		25 ns	10 ns	50 ns	25 ns/8 b	4	750 ns
4-cascade		25 ns	10 ns	50 ns	25 ns/16 b	4	500 ns
METROJR $w = 8$		25 ns	10 ns	50 ns	25 ns/8 b	4	725 ns
METROJR	0.8μ Std. Cell	10 ns	5 ns	20 ns	10 ns/4 b	4	500 ns
2-cascade		10 ns	5 ns	20 ns	10 ns/8 b	4	300 ns
4-cascade		10 ns	5 ns	20 ns	10 ns/16 b	4	200 ns
METRO $i = o = 8$							
$w = 4$		10 ns	5 ns	20 ns	10 ns/4 b	2	460 ns
METROJR	0.8μ Full Custom	5 ns	3 ns	15 ns	5 ns/4 b	4	270 ns
METRO $i = o = 8$							
$w = 4$		5 ns	3 ns	15 ns	5 ns/4 b	2	240 ns
METROJR $dp = 2$		2 ns	3 ns	10 ns	2 ns/4 b	4	124 ns
METROJR $hw = 1$		2 ns	3 ns	8 ns	2 ns/4 b	4	120 ns
2-cascade		2 ns	3 ns	8 ns	2 ns/8 b	4	80 ns
$w = 8$		2 ns	3 ns	8 ns	2 ns/8 b	4	80 ns
METRO $i = o = 8$							
$hw = 2$ $w = 4$		2 ns	3 ns	8 ns	2 ns/4 b	2	104 ns
4-cascade		2 ns	3 ns	8 ns	2 ns/16 b	2	44 ns

Shown here is a summary of METRO implementations and possible implementations. As an example of how these parameters impact performance, the last column shows $t_{20,32}$, the network latency required to deliver a 5-word (20-byte) message (*e.g.* a 4-word cache-line including checksum) across a 32-node multibutterfly network constructed like the one shown in Figure 1. Table 4 summarizes the assumptions and calculations used to derive $t_{20,32}$ for various METRO implementations.

Table 3: METRO Implementation Examples

t_{wire}		3 ns	assumed wire delay
vtd		$\left\lceil \frac{t_{io}+t_{wire}}{t_{clk}} \right\rceil$	interconnect delay in clock cycles
t_{on_chip}		$t_{clk} \cdot dps$	time data traverses chip
t_{stg}		$t_{on\_chip} + vtd \cdot t_{clk}$	chip-to-chip latency in the nework
$hbits$	$hw > 0$	$hw \cdot w \cdot c \cdot stages$	routing bits required
	$hw = 0$	$\left\lceil \frac{\left(\sum_{(\forall \text{ stages } i)} (\log_2 r_i)\right)}{w} \right\rceil \cdot w \cdot c$	c = number of cascade routers
$t_{20,32}$		$stages \cdot t_{stg} + (20 \cdot 8 + hbits) \cdot t_{bit}$	32-node, 5-word delivery time

This table summarizes the assumptions and relations used to calculate application latency for Table 3.

Table 4: Latency Equations for METRO

Router	Latency	t_{bit}	$t_{20,32}$	Reference
DEC/GIGAswitch	<15 μs/22-port xbar	10 ns/1 b	16μs	[5]
KSR/KSR-1	3μs/32-node ring	30 ns/8 b	3.5μs	[12]
TMC/CM-5 Router	250 ns/4-ary switch	25 ns/4 b	1.5μs $\rightarrow$ 3.5μs	[13]
INMOS/C104	<1μs/32-port xbar	10 ns/1 b	2.5μs	[18]
MIT/J-Machine	60 ns/ 3D router	30 ns/8 b	660 $\rightarrow$ 1020 ns	[6]
Caltech/MRC	50 $\rightarrow$100 ns/ 2D router	11 ns/8 b	300 $\rightarrow$ 800 ns	[21]
Mercury/Race	100 ns/6-port xbar	5 ns/8 b	500 ns	[1]

For sake of comparison, the last column estimates $t_{20,32}$, the network latency required to deliver a 5-word (20-byte) message (*e.g.* a 4-word cache-line including checksum) across an unloaded network supporting 32 processors.

Table 5: Contemporary Routing Technologies

8 Conclusions

The ensemble of features provided by the METRO architecture make it suitable for a wide range of applications and allow high-performance implementations both at the single chip level and at the network level.

METRO routers can be implemented to achieve high performance for two key reasons: (1) protocol/function simplicity and (2) architectural flexibility. The simple, deterministic routing protocol allow implementations of METRO routers with few gate-delays in the critical path. The latency required to setup a connection is minimized by leaving functions like buffering, packet manipulation, and adaptive routing out of the router. The flexible pipelining opportunities provided by the architecture allows a specific METRO implementation to be optimized to take full advantage of the implementation technology available. In technologies which allow high i/o bandwidth, the combination of variable turn delay and connection pipelining allows a METRO component to achieve very high bandwidth while conforming to the basic architecture and protocol. Connection setup pipelining allows data to stream through the router at a minimum latency once the connection is established in implementations where the latency to establish a connection is much larger than the data transfer latency.

Despite the fact that the METRO router is simple, the minimal hooks it does provide allow networks built from METRO routing components to achieve good aggregate performance and fault tolerance. Stochastic path selection allows the ensemble of routers to avoid network congestion and to tolerate even dynamic network faults. Fast path reclamation allows stochastic search for non-faulty, uncongested paths to proceed rapidly. With connection reversal, METRO routing components efficiently support end-to-end protocols which guarantee reliable message delivery.

The METRO architecture describes a large class of routers which can be implemented to serve applications with widely varying requirements. Cleanly parameterized as it is, routers with varying datapath size, input and output ports, and dilations can easily be specialized from the base architecture. METRO allows room for tradeoffs to be made between latency, throughput, i/o pins, and cost on an implementation and application basis.

A fully-synthesized, gate-array, ASIC implementation of the METRO architecture exhibits comparable or superior performance to many, more expensive and more custom implementations of other router architectures. Semi- or full-custom implementations of the METRO architecture hold promise for even higher performance at reasonable costs.

References

[1] Warren Andrews. RACE architecture brings flexibility to multiprocessing. *Computer Design*, 31(5):44–48, May 1993.

[2] Frederic Chong, Eran Egozy, and André DeHon. Fault Tolerance and Performance of Multipath Multistage Interconnection Networks. In Thomas F. Knight Jr. and John Savage, editors, *Advanced Research in VLSI and Parallel Systems 1992*, pages 227–242. MIT Press, March 1992.

[3] Frederic T. Chong and Thomas F. Knight, Jr. Design and Performance of Multipath MIN Architectures. In *Symposium on Parallel Architectures and Algorithms*, pages 286–295, San Diego, California, June 1992. ACM.

[4] IEEE Standards Committee. *IEEE Standard Test Access Port and Boundary-Scan Architec-*

ture. IEEE, 345 East 47th Street, New York, NY 10017-2394, July 1990. IEEE Std 1149.1-1990.

[5] Digital Equiment Corporation. GIGAswitch FDDI Crossbar Switch. Online, Internet-Accessible Product Information, April 1993.

[6] William J. Dally et al. The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms. *IEEE Micro*, pages 23–39, April 1992.

[7] André DeHon. Practical Schemes for Fat-Tree Network Construction. In Carlo H. Séquin, editor, *Advanced Research in VLSI: International Conference 1991*, pages 307–322. MIT Press, March 1991.

[8] André DeHon. Scan-Based Testability for Fault-tolerant Architectures. In Duncan M. Walker and Fabrizio Lombardi, editors, *Proceedings of the IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pages 90–99. IEEE, IEEE Computer Society Press, 1992.

[9] André DeHon. Robust, High-Speed Network Design for Large-Scale Multiprocessing. AI Technical Report 1445, MIT Artificial Intelligence Laboratory, 545 Technology Sq., Cambridge, MA 02139, February 1993.

[10] André DeHon, Thomas F. Knight Jr., and Henry Minsky. Fault-Tolerant Design for Multistage Routing Networks. In *International Symposium on Shared Memory Multiprocessing*, pages 60–71. Information Processing Society of Japan, April 1991.

[11] André DeHon, Thomas F. Knight Jr., and Thomas Simon. Automatic Impedance Control. In *ISSCC Digest of Technical Papers*, pages 164–165. IEEE, February 1993.

[12] Thomas H. Dunigan. Kendall Square Multiprocessor: Early Experiences and Performance. ORNL/TM 12065, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, March 1992.

[13] Thorsten von Eicken et al. Active Messages: a Mechanism for Integrated Communication and Computation. In *Proceedings of the 19th Annual Symposium on Computer Architecture*, Queensland, Australia, May 1992.

[14] Ronald I. Greenberg and Charles E. Leiserson. Randomized Routing on Fat-Trees. In *IEEE 26th Annual Symposium on the Foundations of Computer Science*. IEEE, November 1985.

[15] Tom Leighton, Derek Lisinski, and Bruce Maggs. Empirical Evaluation of Randomly-Wired Multistage Networks. In *International Conference on Computer Design: VLSI in Computers and Processors*. IEEE, 1990.

[16] Tom Leighton and Bruce Maggs. Expanders Might Be Practical: Fast Algorithms for Routing Around Faults on Multibutterflies. In *IEEE 30th Annual Symposium on Foundations of Computer Science*, 1989.

[17] Charles E. Leiserson. Fat-Trees: Universal Networks for Hardware Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[18] M. D. May, P. W. Thompson, and P. H. Welch, editors. *Networks, Routers and Transputers*. IOS Press, 1993.

[19] Henry Minsky, André DeHon, and Thomas F. Knight Jr. RN1: Low-Latency, Dilated, Crossbar Router. In *Hot Chips Symposium III*, 1991.

[20] Henry Q. Minsky. A Parallel Crossbar Routing Chip for a Shared Memory Multiprocessor. AI Technical Report 1284, MIT Artificial Intelligence Laboratory, 545 Technology Sq., Cambridge, MA 02139, 1991.

[21] Ravi Soundararajan. MRC Specifications. Alewife Systems Memo 25, MIT Artificial Intelligence Laboratory, 545 Technology Square, Cambridge MA 02139, February 1992.

[22] Synopsys. *Design Compiler Reference Manual*. Synopsys, Inc., version 3.0 edition, December 1992.

[23] E. Upfal. An $O(\log N)$ deterministic packet routing scheme. In *21st Annual ACM Symposium on Theory of Computing*, pages 241–250. ACM, May 1989.