

POLITECNICO DI MILANO
MICROLAB
HPPS PROJECT PRESENTATION



CITiES
Reconfiguration Oriented Metrics

Tutor: V. Rana

Project Author:
Alessandro Meroni
711381

A.A. 2006/07

Contents

1	State of The Art	4
1.1	Communication Architectures	4
1.2	Performance Parameters	5
1.2.1	Point-to-point	5
1.2.2	Bus	5
1.2.3	NoC	7
1.3	Cost Factors	11
2	Analysis and Qualitative Evaluation	14
2.1	Applications and Scenarios Analysis	14
2.1.1	Applications and Scenarios Layers	14
2.1.2	Characteristics and Metrics Layer	17
2.2	Metrics Evaluation	17
2.3	Simulators Examination	19
3	Implementation and Results	22
3.1	Simulation Environment	22
3.1.1	Design Flow	22
3.2	Models Description	23
3.2.1	Assumptions and Considerations	25
3.3	ROME Implementation	29
3.4	Results	31
3.5	Future Works	32

Abstract

We are dealing with the creation of a new communication infrastructure on a multi processing elements SoC architecture.

Aim of this project is the definition of a set of novel metrics useful to evaluate different communication infrastructures to identify the best solution that have to be used into a given reconfigurable system. We need metrics that can validate and drive the choice of this new approach, and a simulator through which perform testing to discover the best configuration of the implementation that we want on-chip.

This project focuses on the analysis and study of these metrics. The most important metric that we will evaluate, concerns the analysis of Load Balancing. This analysis may deal with the comparison among different system parameters, such as the topology of the communication infrastructure and the number of processing elements and their topology. For instance with a good metric based on load balancing we can create or delete system nodes as the load change. With this approach we can avoid the formation of bottleneck and so, get the system performed.

Other metrics may evaluate the throughput, the number of the single nodes and the research of the saturation point of a particular configuration.

Through the creation of different network models, feature provided by a specific network simulator, we will evaluate the existing metrics with respect to their architecture and infrastructure in which are exploited. After several simulations, a vast amount of data has been stored; to allow faster search of the best-fitting solution we used a mysql-database.

*A program written in C/C++, named **rome**, permits the user to query the database with respect to a configuration file provided from the user, containing all the necessary information.*

1 State of The Art

A new set of metrics is need to define a collection of guidelines to drive the Communication Infrastructure reconfiguration and the dynamic plugins of Processing Elements.

Before creating this set, we need to evaluate the existing metrics with respect to their architecture and infrastructure in which are exploited.

Aim of this first project report is the presentation and the analysis of a well-known metrics set. There will be a first classification by the communication architecture and then another sub-classification relative to the topology used (w.r.t. NoC).

1.1 Communication Architectures

When dealing with Systems-on-Chip, the main important aspect about metrics that should be considered is the communication architecture implemented.

To better illustrate the taxonomy of different communication architectures, in Figure 1 are shown the most relevant ones that will be evaluated.

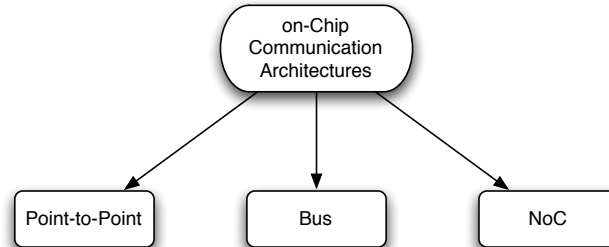


Figure 1: Communication Architectures Taxonomy

Since the introduction of the SoC concept in the 90s, the solutions for SoC communication structures have generally been characterized by custom designed ad hoc mixes of buses and point-to-point links. But due to some limitations about the bus and the point-to-point approach, a common concept for segmented SoC communication structures, based on networks, rose. This new concept is known as Network-on-Chip (NoC) [?].

There are many performance parameters and cost factors that characterize an architecture. In the bus and the NoC approach, the common performance parameters are: latency, bandwidth and throughput, and the common cost factors are: power consumption and area usage. But there are several parameters that are only relevant in a NoC implementation, such as the topology, the nodes degree and the link complexity.

1.2 Performance Parameters

In this section there will be the description of the common performance parameters of the point-to-point, bus and NoC architectures. All the parameters relevant for the bus-based architectures, are the same for the NoC ones, so they will be explained only in the Bus sub-section. Instead, in the sub-section 1.2.3 NoC there will be the presentation of all the parameters that are useful only for a NoC-based approach.

1.2.1 Point-to-point

A very simple communication architecture is the point-to-point one (Figure 2). In this configuration all the elements of the chip are interconnected each other with links.

As stated in [?], dedicated point-to-point links are optimal in terms of bandwidth availability, latency, and power usage as they are designed especially for this given purpose. Also, they are simple to design and verify and easy to model. But the number of links needed increases exponentially as the number of cores increases. Thus an area and possibly a routing problem develops.

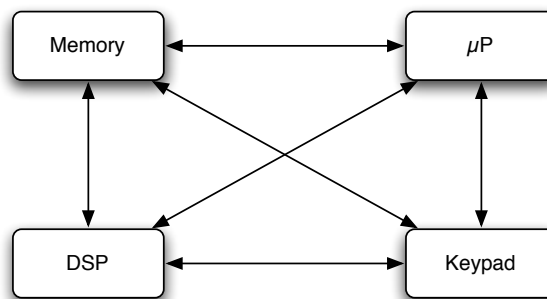


Figure 2: Point-to-point Communication Architecture

From the point of view of design-effort, one may argue that, in small systems of less than 20 cores, an ad hoc communication structure is viable. But, as the systems grow and the design cycle time requirements decrease, the need for more generalized solutions becomes pressing. For maximum flexibility and scalability, it is generally accepted that a move towards a shared, segmented global communication structure is needed.

1.2.2 Bus

The most commonly used communication structures are buses (Figure 3). Their main advantages are their flexibility, extensibility and their low design costs.

As recently stated in [1], bus-based communication architectures are mainly characterized by latency and bandwidth. Performance numbers may relate either to real time or depend on the maximum clock rate of a synchronous system.

For instance, such metrics as latency, bandwidth and throughput could be generally defined as follows:

- **Latency** $l_i \in \mathbb{N}_0$ of a network element i is the number of clock cycles by which a data transfer is delayed when passing through the network element. The *path latency* $l_P \in \mathbb{N}_0$ is the the number of clock cycles by which a data transfer is delayed by n network elements along the path from source to destination: $l_P = \sum_i^n l_i$.
- **Bandwidth** $b_L \in \mathbb{R}_0^+$ is the amount of data a network link L can physically deliver per time unit.
As bandwidth depends on the clock frequency and bitwidth of a link, these basic parameters are mainly considered in the following. Besides latency and bandwidth, throughput is also commonly used for characterizing the performance of communication architectures:
- **Throughput** $t \in \mathbb{R}_0^+$ of a networking is defined by $t = \sum_{i=1}^k b_{L_i}$, k being the number of links and b_{L_i} being the bandwidth of link L_i .

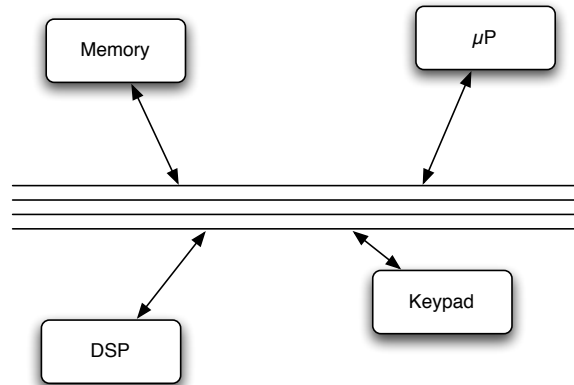


Figure 3: Bus-based Communication Architecture

In general, buses show a low latency when the bandwidth demands are low. They are well suited for communication patterns where most of the communication takes place between two components, for instance between a processor and memory. As soon as the communication pattern involves more than two modules, the main disadvantage of buses become apparent: their scalability is poor.

Since all traffic shares a single path, time-overlapping bus requests have to be serialized. This leads to a sharing of the effective bandwidth as the number of components increases. One way to deal with this issue is using *hierarchical* or *split* buses. The AMBA [?] and CoreConnect [?] bus architectures are examples of hierarchical bus systems. They consist of a low-speed peripheral bus connected to a high-speed system bus through a bridge. There are also extensions where the buses are split into *segments*, thus allowing locality of communication to be exploited. These extensions increase the scalability of buses at the cost of flexibility, as bridges may lead to bottlenecks between hardware modules on separated buses. Another disadvantage of buses are their long communication lines. In FPGA designs, those lines are costly to route, and in general lead to huge power consumption and a limited system clock speed.

New approaches have led to the definition of new performance parameters, such as the parallelism and the modularity that a system could achieve exploiting different communication architectures. As described in [?], it is also possible to define these new general parameters:

- **Parallelism** $dm_{max} \in \mathbb{N}$ supported by a network is the maximum number of independent data transfers that can occur simultaneously.
- **Flexibility** is the ability of a communication structure to support different communication patterns in a fixed design without loss of performance.
- **Scalability** describes the ability of a communication structure to be modified to provide a fixed set of performance parameters independently from the system size and characteristics.
Normally, the term scalability would only refer to the design time of a system. As for dynamical reconfigurable FPGA designs structural modifications are not limited to design time, we extend the definition of scalability also to the runtime of a system.
- **Extensibility** is the ability of an architecture to be extended to larger system designs.
Extensibility is closely related to scalability, but does not require the extended system to maintain the performance of the unextended system.
- **Modularity** The *modularity* of a communication architecture is defined by the extent up to which it can be divided into submodules.

1.2.3 NoC

Regarding NoC (Figure 4), the metrics analyzed are mainly those studied in classic real networks such as LANs or WANs [?].

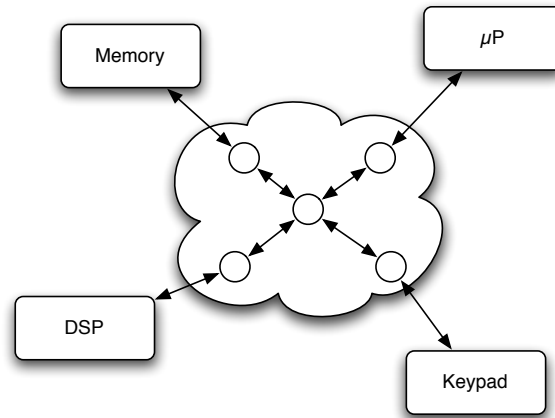


Figure 4: Network-on-Chip Communication Architecture

It is possible to make a sub-classification with respect to the different topology used in the Network-on-Chip. After that it will be possible to evaluate the factors that characterize a network, such as the degree and the link complexity. Furthermore it should be considered the routing protocol exploited to the routing of packets in the net.

Topology

One of the most relevant parameters that can make the difference, is the *topology* used in the Network-on-Chip. Topology concerns the layout and connectivity of the nodes and links on the chip.

As stated in [?], one simple way to distinguish different regular topologies is in terms of *k-ary n-cube* (grid-type), where k is the degree of each dimension and n is the number of dimensions. The *k-ary tree* and the *k-ary n-dimensional fat tree* are two alternate regular forms of networks explored for NoC. The network area and power consumption scales predictably for increasing size of regular forms of topology. Generally, mesh topology makes better use of links (utilization), while tree-based topologies are useful for exploiting locality of traffic.

Irregular forms of topologies are derived by mixing different forms in a hierarchical, hybrid, or asymmetric fashion. Irregular forms of topologies scale non-linearly with regards to area and power. These are usually based on the concept of clustering.

There are many properties that characterize different NoC's topology, [?] [?], such as:

- **Diameter:** Diameter is the maximum distance between two nodes.

Topology	Diameter	Link Complexity	Degree	Regular	Bisection Width
Ring	$\frac{n}{2}$	n	2	yes	2
Star	2	$n - 1$	$1, n - 1$	no	1
Square Mesh	$2(n^{1/2} - 1)$	$2(n - n^{1/2})$	2, 3, 4	no	$2\sqrt{n}$
Hypercube	$\log_2 n$	$\frac{n \log_2 n}{2}$	$\log_2 n$	yes	$\frac{n}{2}$
Fully Connected	1	$\frac{n(n-1)}{2}$	$n - 1$	yes	$\frac{n^2}{4}$

Table 1: Characteristics of different NoC's topologies

- **Link Complexity:** Link complexity is the number of links the topology requires.
- **Degree:** Node degree is the number of links from a node to its nearest neighbors.
- **Regularity:** A network is deemed to be regular when all the nodes have the same degree.
- **Bisection Width:** Bisection width is the number of links that must be cut when the network is divided into two equal set of nodes.

With these properties it is possible to create a table that summarize all the aspects of different NoC's topologies: Table 1.

Routing Protocols

The protocol concerns the strategy of moving data through the NoC. It is possible to define switching as the mere transport of data, while routing is the intelligence behind it, that is, it determines the path of the data transport.

In the following, (w.r.t. [?]) these and other aspects of protocol commonly addressed in NoC research, are discussed.

- **Circuit vs packet switching.** Circuit switching involves the circuit from source to destination that is setup and reserved until the transport of data is complete. Packet switched traffic, on the other hand, is forwarded on a per-hop basis, each packet containing routing information as well as data.
- **Connection-oriented vs connectionless.** Connection-oriented mechanisms involve a dedicated (logical) connection path established prior to data transport. The connection is then terminated upon completion of communication. In connectionless mechanisms, the communication occurs in a dy-

dynamic manner with no prior arrangement between the sender and the receiver. Thus circuit switched communication is always connection-oriented, whereas packet switched communication may be either connection-oriented or connectionless.

- **Deterministic vs adaptive routing.** In a deterministic routing strategy, the traversal path is determined by its source and destination alone. Popular deterministic routing schemes for NoC are source routing and X-Y routing (2D dimension order routing). In source routing, the source core specifies the route to the destination. In X-Y routing, the packet follows the rows first, then moves along the columns toward the destination or vice versa. In an adaptive routing strategy, the routing path is decided on a perhop basis. Adaptive schemes involve dynamic arbitration mechanisms, for example, based on local link congestion. This results in more complex node implementations but offers benefits like dynamic load balancing.
- **Minimal vs nonminimal routing.** A routing algorithm is minimal if it always chooses among shortest paths toward the destination; otherwise it is nonminimal.
- **Delay vs loss.** In the delay model, datagrams are never dropped. This means that the worst that can happen is that the data is delayed. In the loss model, datagrams can be dropped. In this case, the data needs to be retransmitted. The loss model introduces some overhead in that the state of the transmission, successful or failed, must somehow be communicated back to the source. There are, however, some advantages involved in dropping datagrams, for example, as a means of resolving network congestion.
- **Central vs distributed control.** In centralized control mechanisms, routing decisions are made globally, for example, bus arbitration. In distributed control, most common for segmented interconnection networks, the routing decisions are made locally.

Another very important performance parameter of a Network-on-Chip, is the routing *algorithm* used to delivery packets among different nodes.

In a system implementing a NoC, it could be possible that nodes (processing elements or routers or switches), or links become unaviable either temporarily or permanently. In such cases it is evident the need of a dynamic routing mechanism which can automatically calculate alternate paths in case of link/router failures on a chip.

As described in [?] there could be a huge difference between using a routing algorithm or not. These are some classic algorithms used in real networks and that can be applied in a simple way to a NoC.

- **Distance Vector Routing:** This is quite a simple routing mechanism where each router maintains a table about the known destinations available to it along with the link to get there, and it sends this information to its neighbors. In this way distance vector routing keeps information about neighbors and destinations only and not of the whole network, which makes it a simple mechanism. Distance vector routing is also known as Bellman-Ford algorithm because it is based on a shortest path computation algorithm which was first described by R.E. Bellman.
- **Link state Routing (exploiting Dijkstra’s “Shortest Path First”):** In contrast to distance vector routing which shares its routing table with immediate neighbours only, the link state routing algorithm tells every router on the network about its neighbour routers. The mechanism is based upon a *distributed map*, where each router has a copy of the map and it is regularly updated. The goal of link state routing is for each peer to have an identical picture of the state of the entire network; link state protocols require each router to know whether a link is up or down and which cost it has, and then calculate the total cost to reach a destination.

For instance, exploiting the Link State Routing in a simple network, Figure 5a, could lead to different performance results thanks to the changing of the *data rate*. In fact, as described in [?] and represented in Figure 5b-c, decrementing by the 45% the data rate, the packets drop ratio, that represents how many packets loss there are, is less in Figure 5c, than that one in Figure 5b.

1.3 Cost Factors

Usually to evaluate the feasibility of different interconnect schemes, it is important to think about the *power consumption* and the *area requirements*.

- **Power consumption:** The power consumption of the communication structure in large single-chip systems is a major concern, especially for mobile applications. The power used for global communication does not scale with technology scaling, leading to increased power use by communication relative to power use by processing. In calculating the power consumption of a system, there are two main terms: *power per communicated bit* and *idle power*. Depending on the traffic characteristics in the network, different implementation styles will be more beneficial with regards to power usage. Asynchronous implementation styles, are beneficial for low network usage since they have very limited power consumption when idle, but use more power per communicated bit due to local control overhead. Technology

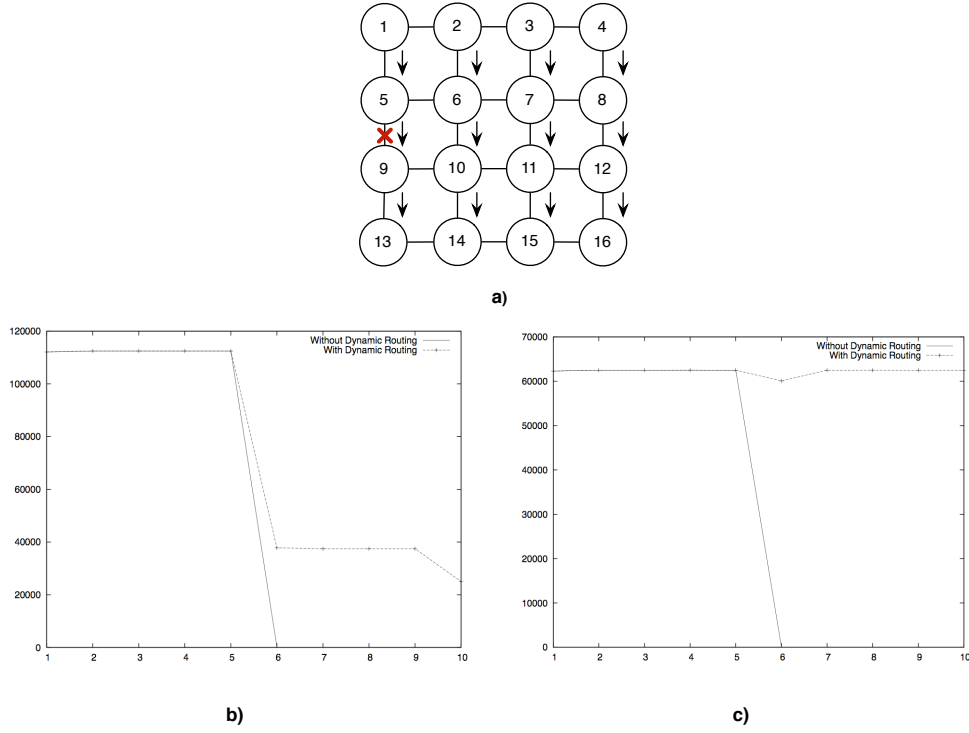


Figure 5: **a)** Simple NoC exploiting LS algorithm to find alternative path after the failure of 5-9 link. **b)** Throughput with an initial Data Rate. **c)** Throughput with only the 55% of the initial Data Rate.

scaling, however, leads to increased leakage current, resulting in an increasing static power use in transistors. Thus the benefit of low idle power in asynchronous circuits may dwindle.

- **Area usage:** As stated in [?], a design requires a certain number of LUTs and flip flops, and depending on how well the packing algorithm performs, the design will require more or less slices. Moreover, the place and route tools may not be able to utilize the two LUTs in every slice because of routing constraints and timing requirements. The number of slices better reflects the actual chip area required to implement the design. So the *area usage* can be defined as $A \in \mathbb{N}_0$, where A is the number of Slices of the chip.

Furthermore it is important to take care about the resources used by the system implementation, such as the *logic resources* and the *routing resources*:

- **Logic Resource:** The logic resource usage is measured in terms of the total number of LUTs required for a design and the number of LUTs related to

only the interconnection network, i.e., those used to implement the network interface modules. It is possible to think about the total number of LUTs used for each design, that is the *Logic Utilization*, and about the number of LUTs used for the network interfaces as a fraction of the total LUTs required for the complete system, that is the *Logic Overhead*.

- **Routing Resource:** The routing resource utilization is measured in terms of the total number of nets in the design and the number of nets used to implement only the network links and network interfaces. For instance, [?], the number of nets attributed to the network is found by counting the number of nets related to all the network interface modules in the design. It is possible to classify this cost factor, as the logic resource, into two measures: *Resource Utilization* and *Resource Overhead*.

2 Analysis and Qualitative Evaluation

After a complete study of the state of the art, we contextualized the different metrics with respect to real applications of different reconfigurable systems.

In order to evaluate the different trends of these metrics, we have analyzed different networks simulators to decide which one of these could be a useful tool to gain data results, such as throughput and latency, and that could be very fast in the data retrieval phase, for instance through shell execution.

2.1 Applications and Scenarios Analysis

In the CITiES group we are dealing with the reconfiguration of a communication infrastructure, so the first thing that we decided to do was to analyze the real world applications in which this reconfiguration take meaning.

We tried to answer questions such as “Which of these applications do really need a communication infrastructure reconfiguration ?”.

To answer this topology of questions we decided to start from the identification of different applications, also thanks to the *Market Solutions* provided from the Xilinx web-site [?].

After this identification, we analyzed the different scenarios in which each of these applications could be used for, and we obtained all the main characteristics of each scenario.

In the end, a classification has been made of the most relevant metrics with respect to the characteristics required from each scenario of the different applications.

2.1.1 Applications and Scenarios Layers

During the analysis of this layer we found different applications, Figure 6, that can be interesting to study, relative to their exploitation of the system reconfiguration.

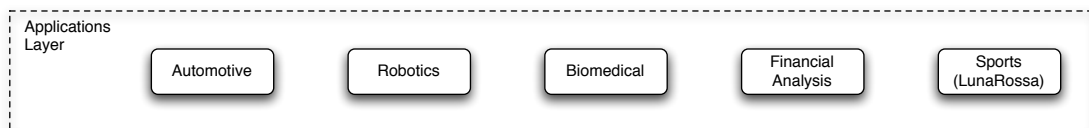


Figure 6: Applications Layer

The most relevant applications found are:

- **Automotive:** The automotive electronics market continues to grow on many levels but more specifically in the areas of in-car audio, infotainment sys-

tems and driver information systems. The market is primarily comprised of the following application segments:

- *Infotainment & Communication*: telematics, GPS navigation, multimedia systems, audio systems, rear-seat entertainment, DVD players, game consoles, integrated mobile phones, Internet access, digital radio and display systems.
 - *Driver Assistance Systems*: night vision & lane warning, tire pressure monitoring, park/reverse assist (back guide monitor), adaptive cruise control, and collision warning.
 - *Comfort & Convenience*: voice recognition, head up displays (HUDs), and reconfigurable instrument clusters.
- **Industrial**: A paradigm shift is taking place in the industrial market, moving from an "own it all" to "own the value add and outsource the rest." Furthermore industrial applications are requiring the ability to customize common hardware and software platforms. New technology demands higher performance DSP capabilities, connectivity and video for today's industrial applications.

Networking of industrial applications is also becoming more of an issue as demand for remote monitoring and wireless-enabled devices continues to grow.

Programmable logic is an ideal solution for these types of applications where multiple standards exist with no clear direction of which standard will prevail and where ASSPs are too costly to implement.

The industrial market covers a wide array of end applications that can be broadly categorized under:

- Security and safety system
 - Industrial automation
 - Control systems, field instrumentation and measurement, switch gear and control gear, sensing
 - Environment and building control
 - Motor control
- **Medical & Science**: The medical and scientific industry faces many issues and challenges in terms of offering high quality imagery, real-time digital processing, and providing high performance, high-speed A-D and D-A conversion.

The medical market covers a wide array of end applications. These end applications can be broadly categorized under:

- *Imaging*: Ultrasound, X-ray, MRI, CT, nuclear
 - *Cardiac rhythm management*: ICDs, external defibrillators
 - *Home health care*: Breath alcohol testers, cholesterol level monitors, emergency ventilators
 - *Hospital equipment*: Patient monitors, dialysis machines, infusion pumps, endoscopy, optometric, powered beds, physiotherapy equipment, etc.
- **Financial Analysis**: It could be very useful in market analysis the evaluation of different models to gain different information form statistics and other. Furthermore, for instance, the computation of different financial functions.
 - **SuperVision Security**: Another example may be a broadcast application used to security aims. These applications deliver:
 - Hardware acceleration to support 5x increase in bandwidth and processing required by high definition (HD) broadcasts
 - Migration support between standard definition (SD) and HD
 - Better compression techniques such as H.264
 - Migration support between MPEG-2 content and H.264
 - Adoption of existing LAN/WAN infrastructures for IP transmission
 - **Robotics**: This last application is the one that we considered the best fitting of our goals, due to its communication infrastructure reconfiguration portability.

This is one of the most significant scenario that includes not only *modules reconfiguration*, but also a *communication infrastructure* one. We have analyzed it and we reach this particular *best fitting* scenario:

 - *Actor*: Robot on a Planet
 - *Objective*: Learning and Exploration
 - *Instrumentation*: FPGAs, different acquisition modules and connection through satellite
 - *Capacities*: Image Analysis, Molecular Evaluation, Sensors Acquisition, Movements Control and Data Elaboration

- Needs: Low Power Consumption and Communication Infrastructure Reconfiguration

After the applications analysis, we made an identification of different scenarios for each applications, so that, for instance, the same scenario may be exploited from different ones.

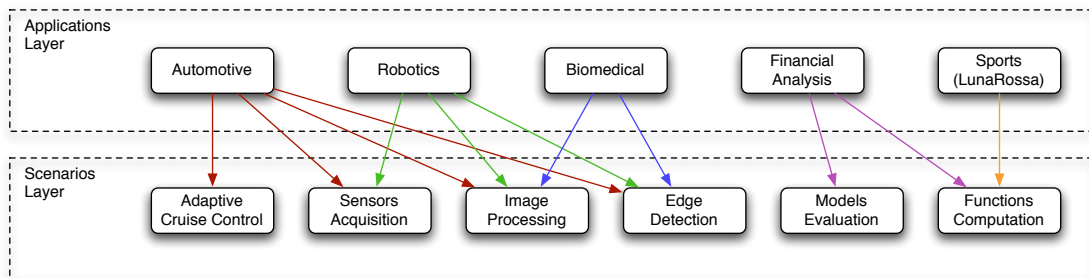


Figure 7: Applications and Scenarios Layers

The result that we reached is represented in the Figure 7, where it can be seen that three different scenarios *Sensors Acquisition*, *Image Processing* and *Edge Detection* belong to two different applications, *Automotive* and *Robotics*.

2.1.2 Characteristics and Metrics Layer

We have also identified in deep the characteristics of each scenarios, in *qualitative* terms, so to have some requirements to satisfy with metrics functions.

For each set of characteristics belonging to a particular scenario, we evaluated some important metrics such as *Throughput*, *Reconfiguration Time*, *Area Usage* and *Power Consumption*.

This *qualitative* analysis lead us to study the different trends that each of these metrics have.

2.2 Metrics Evaluation

In order to obtain some data to evaluate, it has been decided to develop a *light* program written in C/C++, that exploits functions coordinates of different trends, belonging to different communication infrastructures such as *Point-to-Point*, *Bus*, *Full Connected* and *Custom*.

The different trends evaluated, represent the growth or the decrease of *Throughput* and *Area Usage* with respect to the number of elements involved in the system, divided between *master* and *slave*.

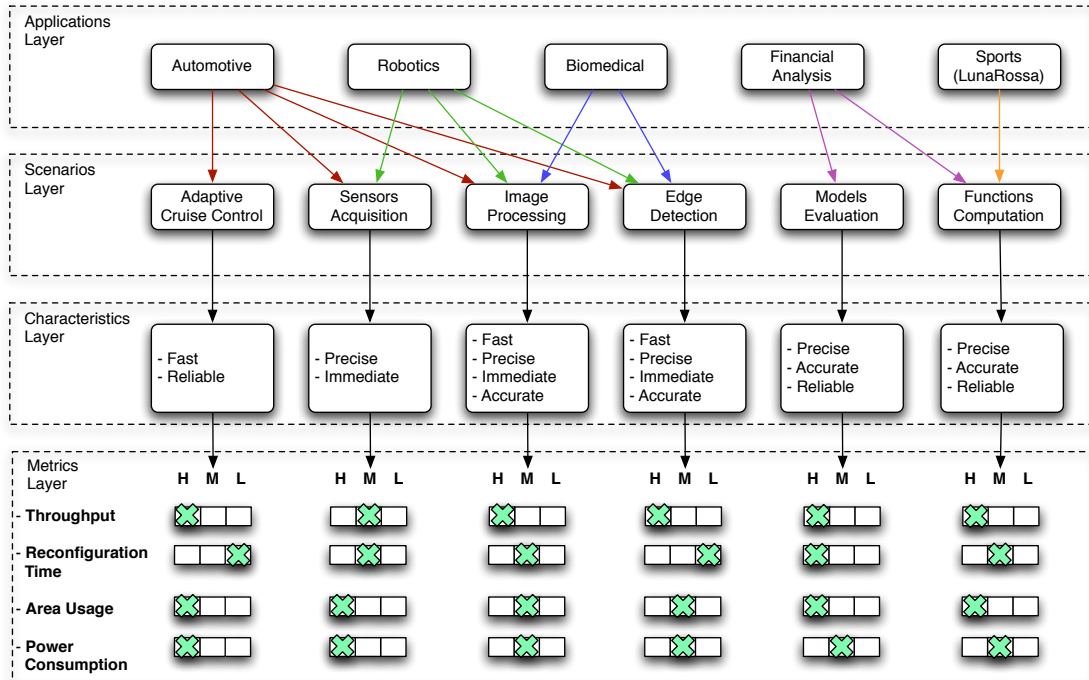


Figure 8: Characteristics and Metrics Layers

By now we considered only these factors that make this approach a bit *rough*, but it's still a work in progress that can lead us to improve step by step our results.

To do this very *qualitative* analysis we decided to divide each communication infrastructure trend into three different graphs, respectively: 20% master and 80% slave, 50% master and 50% slave, 80% master and 20% slave.

Our **prototype Analyzer** takes in input the number of masters and the number of slaves that a user has in his system, and gives as output the best fitting communication infrastructure that maximize *Throughput* and minimize *Area Usage*.

The data exploited to make this decision is something like what is represented in Figure 9.

In this graph, as said before, only a *qualitative* evaluation can be seen, which however is very important as it gave us some data and numbers to think about. By now we didn't take into consideration any type of FPGA used, or which particular configuration is exploited and other relevant factors.

What we are going to do next, is to exploit all the possible ways to achieve data and information of these metrics' trends, for instance through a Simulator (see next Section) we can obtain precise information relative to *Delivery Rate* and *Loss Rate*, and it should be also possible to emulate the *Reconfiguration Time*

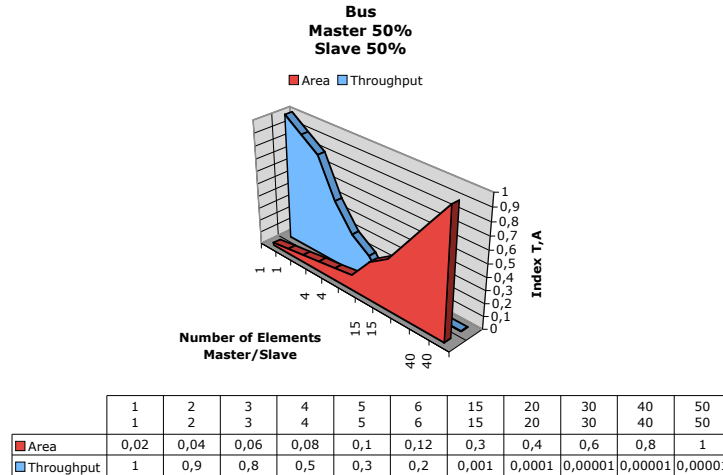


Figure 9: Bus Trends with 50% Master and 50% Slave

introducing some delays.

2.3 Simulators Examination

As stated before, a high-level network simulator is required to evaluate and compare the different communication architecture design decisions. As no general on-chip communication architecture simulator exists by now, we decided to use a network simulator suited for our goals. Once the best communication architecture candidates are identified, the simulation can be refined for that particular architecture with a more precise low-level communication architecture simulator. Different solutions can then be tested and evaluated in terms of power and performance.

The first thing to do was to find this good simulator, and so we needed to examine different network simulators.

Two particular tools, among a set of different simulators, were identified as possible candidate: *NS-2* [?] and *OMNeT++* [?].

- **NS-2:** is a discrete event simulator targeted at networking research that provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.
- **OMNeT++:** is a public-source, component-based, modular and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel.
Its primary application area is the simulation of communication networks

and because of its generic and flexible architecture, it has been successfully used in other areas like the simulation of IT systems, queueing networks, hardware architectures and business processes as well.

OMNeT++ is rapidly becoming a popular simulation platform in the scientific community as well as in industrial settings. Several open source simulation models have been published, in the field of internet simulations (IP, IPv6, MPLS, etc), mobility and ad-hoc simulations and other areas.

We analyzed in deep the differences between these two simulators, thanks to a *web-community* (see Tables 2 and 3, and also the website [?]), and this study lead us to the choice of using **OMNeT++** to reach all the possible information.

features	OMNeT++	NS-2
Flexibility	OMNeT++ is a flexible and generic simulation framework. One can simulate anything that can be mapped to active components that communicate by passing messages. For example, it can be used for simulating queueing networks, multiprocessor systems, hardware architectures or business processes.	NS-2 has been designed as a (TCP/IP) network simulator, and it's difficult to simulate things other than packet-switching networks and protocols with it. It has highly detailed and hardcoded concepts about nodes, agents, protocols, links, packet representation, and network addresses etc, which is good, but makes it very hard if you want to do things a little differently.
Programming Model	Object-oriented, event-driven simulator, written in C++. Topology descriptions are either written as text files (NED language), or can be dynamically created in run-time. There is also a graphical interface (GNED) for creating and editing the topologies, which automatically creates the topology file.	Mixed-mode: OTcl (Object-Tcl) with underlying C++ classes. OTcl is also used for creating and configuring networks, recording results etc.
Model Management	The OMNeT++ simulation kernel is a class library, i.e., models in OMNeT++ are independent of the simulation kernel. The researcher writes their components (simple modules) against the OMNeT++ simulation kernel API. OMNeT++ sources are never patched by models. Simple modules are then reusable, and can be freely combined like LEGO blocks to create simulations..	In ns-2, boundary between simulation core and models is blurred, without a clear API. Install instructions for 3rd party models usually begin like: "download ns2 2.xx.x, unpack it, then apply the following patch..."

Table 2: OMNeT++ Vs NS-2

features	OMNeT++	NS-2
Ability to Run Large Networks	OMNeT++ can simulate very large scale network topologies. The limit is the virtual memory capacity of the computer used.	ns-2 has scalability problems on simulating large network topologies (more details needed here).
Support for Parallel Simulation	Supports conservative parallel distributed simulation. The Null Message Algorithm (Chandy-Misra-Bryant) and Ideal Simulation Protocol (Bagrodia et al) are supported; others can be plugged in. Lookahead models for NMA can be plugged in. Communication layer is pluggable: currently implemented ones are MPI, named pipe, and file-based (for debugging). Unlike PADS, models do not need to be modified or instrumented for parallel simulation – it is just a matter of configuration.	The PADS research group at Georgia Tech. has developed extensions and enhancements to the ns-2 to allow a network simulation to be run in a parallel and distributed fashion on a network of workstations.
Experiment Design	Parameters of a simulation experiments are written in the omnetpp.ini, which enforces the concept of separating model from experiments.	Models and experiments are usually interwoven in ns-2: topology, parameters, model customizations, result collection etc usually in the same Tcl script, which makes "separation of concerns" difficult.
Support for Hierarchical Models	Hierarchical module structure in OMNeT++ facilitates dealing with complexity in a methodical manner. Model designer assembles a complex model from self-contained building blocks (i.e. simple modules and compound modules) which are reusable in other simulations as they are.	In ns-2, models are "flat": creating subnetworks, or implementing a complex protocol as a composition of several independent units (that appear as one unit) are not possible in ns-2.
Debugging and Tracing Support	OMNeT++ can show packet transmissions while a simulation is running. OMNeT++'s Tkenv is an interactive execution environment, which allows one to examine the progress of simulation and change parameters. There is also extensive library support for packet tracing etc.	?
Variety of Models Available	OMNeT++ has a good variety of models for simulating computer systems, queueing systems etc., but lags behind the ns-2 simulator on availability of communication protocol models.	ns-2 has a rich set of communication protocol models (since it has been designed as a network protocol simulator, this is not surprising).
Documentation	OMNeT++ has a well written and up-to-date manual (there are also tutorials for quick introduction). OMNeT++'s simulation API is more mature and much more powerful than ns-2's.	ns-2 documentation is fragmented (there is a good tutorial for quick introduction). There is no clear dividing line between the models and the ns-2 simulation library.

Table 3: OMNeT++ Vs NS-2

3 Implementation and Results

In this Section we will present how the different data have been achieved to populate our metrics database; so the simulation environment (OMNeT++) will be described and also will be presented all the assumptions used in the different simulations, with respect to delay, latency, routing protocols and others.

In the end there is the description about how the Prototype Analyzer, now named ROME, functions, and what kind of results can be obtained through it.

3.1 Simulation Environment

As stated in Section 2.3, we choose OMNeT++ as our simulation environment.

One of the most relevant features provided by this simulator, is the possibility to use the C/C++ language to give the simulation description, this gave us the possibility to use a set of C++ libraries, written by us, in support of the simulation.

OMNeT++ is a discrete event C++ simulation library specialized for the network domain. It offers a very flexible simulation environment and as it is an open source environment, it is possible to improve or add some features very easily.

Design of network models is done in a hierarchical way to deal more efficiently with complexity. Network models are built on a set of modules and sub-modules, each describing a specific behavior. Two types of implementations are possible for the C++ behavior model: threads or finite state machines.

OMNeT++ provides a high-level network topology description language (NED language). Topology description is completely independent of the implementation of the nodes behavior and therefore large topology exploration is easy and fast. OMNeT++ follows a modular approach: models are written into independent modules that can be easily reused in other network models.

OMNeT++ offers a graphical editor for network topology description that interprets text-based descriptions.

OMNeT++ has been used with success in a wide range of research areas, from various network simulations (optical, wireless, ad-hoc, IP, ...) to performance analysis of complex software systems (at Siemens) and hardware design (at networking equipment vendor companies). The main hot topics for which OMNeT++ is currently used are mainly IPv6 and wireless simulations.

3.1.1 Design Flow

The OMNeT++ simulation environment design flow is presented on Figure 10. First, network topology described in the NED format is translated in C++ by a preprocessor. Then, this code is compiled together with modules behavior description. Finally the object code is linked with OMNeT++ simulation libraries.

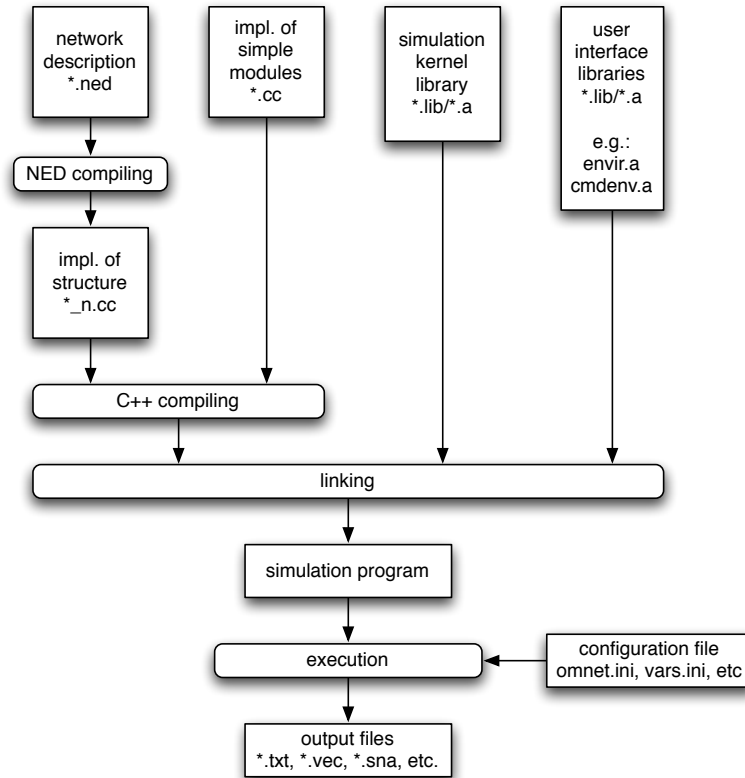


Figure 10: OMNeT++ Simulation Environment

The simulation program is therefore a fast executable object code. Parameters of the network model and of the simulation are written in a separate INI file.

It is thus not needed to recompile the program for each simulation; this permits a fast simulation procedures.

3.2 Models Description

We have a network simulator, and we want to evaluate different networks, so we built three different models, one for each communication architecture: *point to point*, *bus* and *network on-chip*.

For each model, a NED description has been written to design the topology and the modules interconnections. With respect to the NoC communication architecture, we decided to create two models with different topology, *star* and *square mesh*; with this choice it will be possible to compare the bus approach to the star one, and the mesh to all the others.

Due to the flexibility and hierarchical structure provided by the simulator, it

has been possible to reuse the same simple modules for each networks, changing only the C++ code by adding or removing functionalities, when necessary.

We basically created two different compound modules, in which different simple module may be instantiated:

- IP-Core:** it's the compound module that may represents a Master or a Slave, basically they are the same object, but with different functionalities. This compound module has into itself a **core**, that is the simple module which generates packets, of type *request* from a Master and *response* from a Slave. We created also two fifos connected to the core, one for the input flow and one for the output. These are still unused into the IP-Core module, but, if there could be the need, these two fifos are ready to use.

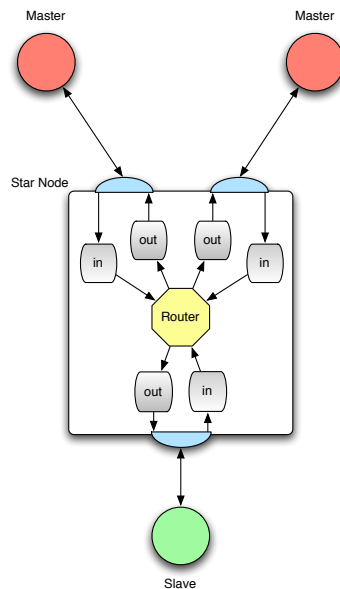


Figure 11: Star Central Node Model

- Node:** this one represents a delivery node of the network. It's the more complex component due to the fact that has the objective to route all the messages and packets to the other elements of the network, either ip-core or node (in a mesh topology). The simple modules contained by a router are the fifos and the router. We decided to model this node in a very simple, but also significative, way; just a fifo for the input, one for the output, and the router in the middle. Figure 11, for instance, represents the model of the central node of a star topology. In the next SubSection (3.2.1) there will be a detailed description about all the assumptions used with the fifo control and the routing protocols.

With the *point to point* approach, there is no need to use a delivery node such as that one previously described, because all the ip-cores are connected each others; only interconnections between slaves are exceptions.

The *bus* architecture has been modeled very similar to the *star* one; the only difference is that the bus doesn't have fifos,; when it's processing a packet, its state becomes *busy*, instead of *idle*, and so all the incoming packets during that state, will be dropped.

With respect to the *star* topology, the *square mesh* is completely different; a delivery node is connected only with a single ip-core, and with, at most, four other nodes, with exceptions for the corner nodes (two connections) and the perimetral ones (three connections). Inside each node, there are four output fifos, and four input fifos (one for each near node), and one local input fifo and a local output one (connected with its relative ip-core).

3.2.1 Assumptions and Considerations

Our goal is to obtain good estimations for each metric of these different communication infrastructures in the simplest way, but also with a meaningful result.

For what concern the simplest way, we made several assumptions w.r.t. to delays, traffic and processing time. It is important to say that we decided to model the communication between, for instance, a master and a slave with the transmission of different packets; this is only one of the several ways to model this communication, we decided this one because of the interesting results about the *Delivery Rate* and the *Loss Rate* of each infrastructure (better analyzed in Section 3.4).

The meaningful attribute can be found in the way we proceeded for each simulation; in particular we made several runs of each model for every possible master-slave configuration (w.r.t. the number of ip-cores) starting from 2 to 100 total number of elements; moreover we decided to make these simulations with, by now, three different level of traffic (packets inter-arrival time, generally speaking *generation time*): *Low*, *Medium* and *Hi*.

Assumptions

Now there is the description and the relative motivation, of each assumption:

- **Value Rules:** for each value in our simulations (packets length, inter-arrival time, latency and so on), we didn't consider it in its absolute format, because it doesn't matter how big is a packet or how fast is the communication channel. The important thing is their ratio; for instance if we have a packet of size 3Kb and a channel of 3Kb/s, in our model, it's the same to have a

packet of 3Mb and a channel of 3Mb/s. By the way it is also possible to select different measures for each value, at compile time.

- **Traffic Level:** to differentiate the simulations with respect to the intensity level of packets generation and to reduce the simulation time (about 100-400minutes for each traffic level simulation, depending on the topology complexity), we decided to create only three different level of traffic. By now they are only three, but only due a fact of time simulation; for instance if we wanted to have several different levels of traffic there hasn't any problem due to the flexibility of the simulation environment; all these parameters are stored in the *omnet.ini* file. The OMNeT++ Simulator provides a set of predefined distributions, continuous and discrete, such as *uniform()*, *exponential()*, *normal()*, *gamma_d()*, *erlang_k()*, *chi_square()*, *cauchy()*, *triang()*, *intuniform()*, *bernoulli()*, *geometric()* and *poisson()*. We decided to use the *exponential()* continuous distribution for our simulations with these three different values: *exponential(1000)* - low traffic -, *exponential(10)* - medium traffic - and *exponential(0.001)* - hi traffic -.
- **Fifo Congestion Control:** we decided to use queues that can contain up to 10 packets; obviously also these fifos are completely customizable. We assumed that when a packet arrives to a fifo, and that one is completely full, the packet will be dropped (basically deleted from the system); we also didn't assume any *timeout* or something else. It is possible to say that we are considering the *loss model* policy (see the *Routing Protocols* section of 1.2.3).
- **Communication Mode:** as stated before, we used a communication between two ip-cores based on packets transmission, because this model can lead to a very interesting results set, for instance the ratio between the packets *completed* and those one *processed* by each ip-core. When a Master sends a *request* packet to a Slave (or another Master), this one answers with a *response* packet and only when this one arrives to its destination we assume that the transfer is completed successfully. A packet is composed of different attributes such as the *destinationAddress*, the *sourceAddress*, its *startTime*, a flag identifying its *type* (request or response) and some booleans identifying different controls about priority queue and other.
- **Routing Protocols:** there are different possibilities about the choice of the routing protocol; at first we used a *random* one, but next, due to the possibility to incur in *deadlocks*, we adopted an *X-Y* routing protocol. We decided to emulate the time used to *search in the routing tables*, with an input fifo of the router, one for each node or ip-core connected to the current node; each

fifo keeps a packet in queue for 3 *time units*, and while the fifo is processing, only up to 10 packets can be queued; also here the *loss model* for packets congestion is applied. Obviously the routing protocol is needed only in the *mesh* topology. OMNeT++ provides also some interesting predefined routing protocols, such as one that look for the *shortest path*.

- **Processing Time:** the processing time is just the time that a router needs to send and compute a packet when it arrives. We putted this time to 10 *time units*, but it's clearly unuseful as it is, due to the fact that a real processing time depends on different aspects. We decided to use 10, just to obtain some interesting timing values; this is completely customizable such all the other ones.
- **Reconfiguration Time:** by now there is no reconfiguration time; this because we are not evaluating yet this aspect in deep. It will be easy to add this time if necessary, just adding a propagation delay that simulates the reconfiguration time. There will be different cases to consider, due to the fact that, for instance in a *mesh* network, when a module is under reconfiguration, the other ones could anyway do their jobs without been affected.
- **Simulation Restrictions:** this simulator is really power, giving vast flexibility and adaptiveness, but if we want to create an ad-hoc inter-connection structure, we had to do it by hands, specifying for each Master which are the only other ip-cores with it can communicate. In this fast simulation environment was quite unfeasible to select for each component its inter-connections with the others; so we decided to examine the worst case in which all the ip-cores can communicate (generating request packets) with all the other ones, except for the slaves that can only generate response packets. By the way, to analyze a common situation that considers the concepts of closeness among ip-cores that communicate each other, and of functionalities set of ip-cores for each master, we created an ad-hoc mesh in with there are Masters near other ip-cores which are the only ones communicating with it.

By now we are creating guide lines to the exploration of these different communication infrastructures, so for example we are not creating hierarchical networks (for instance a star topology where for each node there is a sub-net exploiting a mesh topology or other), but just *atomic* infrastructures.

Considerations

Now that all the assumptions used have been motivated, we will make considerations on how to evaluate metrics from our simulation environment; in particu-

lar which rules, policies, formulae and models we used to populate our metrics database.

So now we present all the attributes of our table, basically our database structure:

- **CI:** it is basically the name identifying the communication infrastructure evaluated, for instance:

$$\left\{ \begin{array}{l} \text{string} : 'PointToPoint' \\ \text{string} : 'Bus' \\ \text{string} : 'Star' \\ \text{string} : 'Mesh' \end{array} \right.$$
- **Traffic Level:** by now there is only a string identifying the traffic level of each row of the table:

$$\left\{ \begin{array}{l} \text{string} : 'Low' \\ \text{string} : 'Medium' \\ \text{string} : 'Hi' \end{array} \right.$$
- **Master:** represents the number of the masters of the system.

$$\left\{ \text{integer} : (1, 100] \right.$$
- **Slave:** represents the number of the slaves of the system.

$$\left\{ \text{integer} : [0, 100) \right.$$
- **Delivery Rate:** we decided to evaluate this attribute as the number of *completed* packets over the *processed* ones by each ip-core (not the router nodes).

$$\left\{ \text{float} : \frac{\text{completed}}{\text{processed}_{ipcore}} \right.$$
- **Loss Rate:** this one, instead, represents the ratio between the *dropped* packets and the *processed* ones, by each router node or bus or central node.

$$\left\{ \text{float} : \frac{\text{dropped}}{\text{processed}_{router}} \right.$$
- **Throughput:** basically a throughput is *packets per seconds*, we decided to use a conservative formula such as the total number of *completed* packets (of each ip-core) over the total simulation time (50000sec every simulation).

$$\left\{ \text{float} : \frac{\text{completed}}{50000} \right.$$
- **Latency:** we used two different formulae to evaluate the latency, due to the presence or not of fifos in the system. So the *Point to Point* and the *Bus* infrastructures have a constant latency that is 20, 10 units time to send

the request packet and 10 to receive the response one. The *Star* and the *Mesh* one have been evaluated subtracting *startTime* from *endTime* of each completed packet.

$$\begin{cases} \text{float} : 20 \\ \text{float} : \text{endTime} - \text{startTime} \end{cases}$$

- **Area Usage:** we decided to consider the area usage as only the *links complexity*, so we used the theory formulae proposed in Section 1.2 and Table 1. Assuming m the number of masters and s the number of slaves, we obtain:

$$\begin{cases} \text{double} : \frac{m \cdot (m-1)}{2} + m \cdot s & \textit{PointToPoint} \\ \text{double} : m + s & \textit{Bus} \\ \text{double} : m + s & \textit{Star} \\ \text{double} : (m + s) + 2 \cdot ((m + s) - \sqrt{m + s}) & \textit{Mesh} \end{cases}$$

- **Power Consumption:** during the evaluation of this cost factor, we found that the *power consumption* may be different for each infrastructure, due to the fact that we could have a *static* factor and a *dynamic* one; this one is obtained with respect to the utilization of the system from, for instance, the routers that generate power dissipation. So the only infrastructures that may generate *dynamic power* are the network-on-chip ones. We decided to use this approximation (considering the ratio *processed_router/processed_ipcore* as the number of hops crossed):

$$\begin{cases} \textit{Static} & \textit{Dynamic} \\ \propto \textit{Area}_{usage} + 0 & \textit{PointToPoint} \\ \propto \textit{Area}_{usage} + 0 & \textit{Bus} \\ \propto \textit{Area}_{usage} + \frac{\textit{processed}_{router}}{\textit{processed}_{ipcore}} & \textit{NoC} \end{cases}$$

With all these assumptions and considerations, we built our dataset onto which perform queries. Now that we have a good number of simulation results (up to 46593 tuples), what we need is a program that could interpretate the user requests and that perform the right query or queries on the mysql-database: **ROME**.

3.3 ROME Implementation

ROME (**R**econfiguration **O**riented **M**etrics), is a tool that acts as a mediator between the user and the database.

To interact perfectly with the database we used the *MySQL++ libraries* set for C++ language [?]. So we can query the database through a program written in C++, with respect the *configuration_file* given by the user and containing all the information about his system and what are his preferences.

MySQL++

MySQL++ is a C++ wrapper for MySQL's C API.

It is built around STL principles, to make dealing with the database as easy as dealing with an STL container. MySQL++ relieves the programmer of dealing with cumbersome C data structures, generation of repetitive SQL statements, and manual creation of C++ data structures to mirror the database schema.

ROME Flow

As perfectly shown in Figure 12, the configuration file is given as input to ROME; then a parser library written in *Flex* takes it and extracts all the information saving them in a data structure that will be used to create, and then perform, the queries to the database.

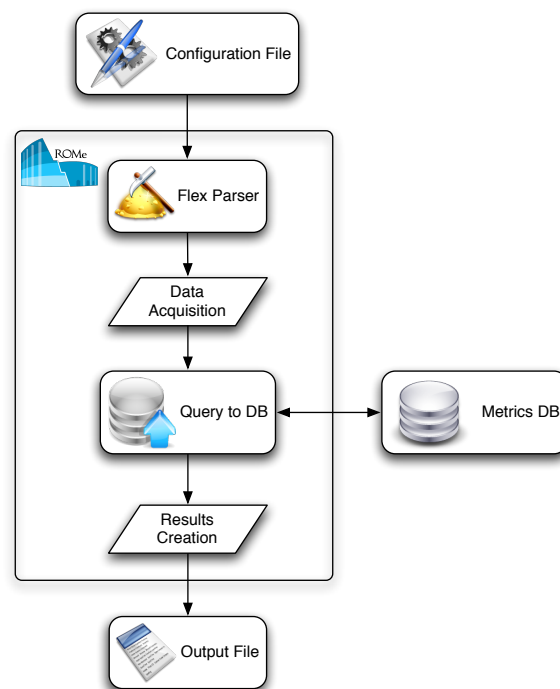


Figure 12: ROME Flow

In the configuration file there could be every type of information, starting from the trivial number of masters or slaves, ending with the complete *incidence matrix*, representing the actual connectivity between a master or slave and all the other elements.

Once the query has been performed, the results are stored in another data structure with respect to the needs of the user. In the end, an *output file* will

be generated with inside it the best communication infrastructure that fits with the information provided by the user.

3.4 Results

At the end of this project we obtained different results, considering different points of view.

Speaking about the *dataset*, it is possible to state that we gained a vast amount of data memorized in our *metrics database*, up to 46593 tuples.

Considering the simulations with OMNeT++, we achieved a good simulation environment; in particular it is a flexible and a suitable environment that can be used also in the future to obtain different values with respect to the assumptions used in the models.

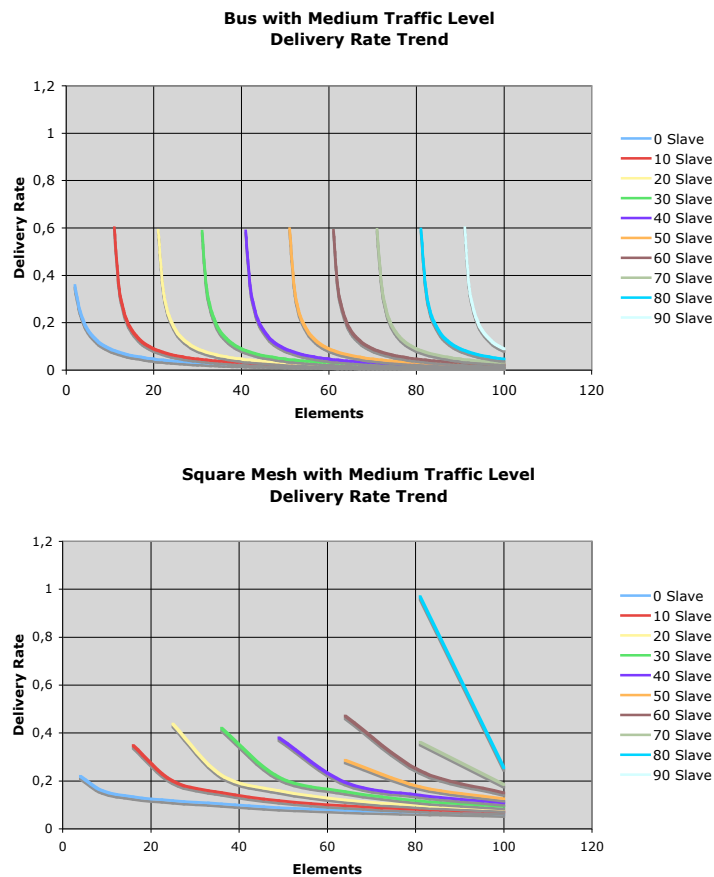


Figure 13: Examples of Simulation Results

As stated before (Section 3.2.1), we obtained interesting graphs, such as the ones reported in Figure 13. In each graph, on the X-axis, is represented the total number of system elements, and, on the Y-axis, the Delivery Rate (a non-dimensional value due to the ratio *completed_Packets / processed_Packets*) changing with the number of slaves of the system.

These two graphs represent the delivery rate trend of the bus approach and the square mesh one.

In the first it can be seen that the system is basically independent w.r.t. the number of slaves, as a matter of facts all the waveforms have the same trend; so if we put, for instance, a threshold on delivery rate equals to *0.2*, to keep only the configurations that have a delivery rate greater than this threshold, we could keep all these different configurations. It is important to notice that with the increasing of the number of elements the delivery rate goes to *zero*.

The square mesh approach is quite different, due to the presence of router nodes that increase with the number of elements of the system, and so the slaves as well; this particular approach it's asymptotic to *zero*, but never reaches this value.

In the end, the most relative result is ROME itself, because it's a very simple and useful tool that permits to gain "*answers*" about a particular system configuration in a very fast way. Also ROME, can be used as a validation environment of a given system.

3.5 Future Works

As CITiES group, we will focus on these three particular ideas.

Simulation Environment Extension

We want to create a valid model of the Network-on-Chip on which the *CITiES group* is working on, so to validate our work considering the best fitting topology.

HW Controller Implementation

Next there is the creation of an hardware element that acts as a system controller, through which perform changes to our system to improve performances.

Simulator Creation

In the end we want to realize a functional communication infrastructures simulator, through which perform different runs to have the possibility to choice the system that better suits our requirements.