# Internet Lessons for Designers of Networks on Chips

Muhammad Ali, Michael Welzl

***Abstract*—Important lessons can be learned by taking a look at history, mechanisms that succeeded and failed and issues that are commonly known in the research community. We therefore point out similarities between NoCs and the Internet and discuss some findings that can be used as rules for NoC designers, who should neither repeat known mistakes nor reinvent the wheel.**

***Index Terms*—Internet, Network on Chips, design**

## I. Introduction

A key difference between Networks on Chips (NoCs) and the Internet --- arguably the smallest and largest types of network in existence --- is their age and the corresponding amount of research that has been carried out. Internet researchers have seen all kinds of technologies come and go, succeed and fail, which has led them to agree upon a number of issues that are known in the community even though they are not listed in a single “Internet Commandments” sort of document. This fact manifests itself in typical recurring design choices made in the standardization body of the Internet, the “Internet Engineering Task Force” (IETF).

This paper constitutes an attempt to extract some important rules from this common Internet knowledge that we consider to be relevant for NoC designers, who should neither reinvent the wheel nor repeat mistakes that the Internet community has already overcome. In the remainder of this article, we will first outline three key aspects of resemblance between NoCs and the Internet and then use them to classify some rules from the Internet community. For each rule, we provide the underlying reasoning and explain its applicability to NoCs.

## II. NoC and Internet Similarities

NoCs have been proposed to overcome the scalability problem of current on-chip buses in SoCs. In such a case, packet based communication model is used where packets are routed over a network of switches laid on a single chip [1]. The idea is inspired from the Internet where a packet is composed of a header and a payload and each packet is routed independently. Analogous to the OSI layers in the Internet, a so-called layered protocol stack governs the communication among all the partners in a NoC [2]. Packet oriented networks are better than connection based networks as they do not require establishing a physical link before the communication starts, hence making their usage more flexible and less power consuming. Packet based communication is already an integral part of the Internet.

Basically being an on-chip communication network which interconnects computational, storage and I/O resources that may belong to different IP blocks, a NoC mainly resembles the Internet with regard to its requirements. It must be:

**Efficient:** messages should traverse the network as quickly as possible; this means that they should take the shortest path and thereby avoid time-consuming buffering. Ideally, one would be able to quantify the performance --- for example, by specifying delay bounds [3].
**Scalable:** the underlying mechanisms must not cease to work as the size of the network increases; the design productivity gap demands that they are able to accommodate an arbitrarily large number of interconnected nodes [4].
**Able to cope with heterogeneity:** the structure of one IP block is unknown to another one, even if they are interconnected via a NoC; the types of interconnected resources vary just like properties of routers and links [5].

In what follows, we will take a closer look at these three aspects.

## III. Efficiency

Let us now consider fast data transfer ---transmission along short (in terms of intermediate devices) and, if possible, uncongested paths. The original design goal for the Internet was to provide robust communication for military purposes; thanks to packet switching, communication can still be provided by routing along an alternate path if a link becomes unavailable.

While this requirement is indeed fulfilled by the Internet, the dynamicity of this functionality is somewhat limited: normally, packets use the path with the smallest number of “hops” (intermediate devices) until a link becomes unavailable. Path changes occur on a timescale that is several orders of magnitude greater than a round-trip time (RTT), which is the time it takes for a packet to be transmitted from source to destination and back. The RTT is generally the duration of interest for dynamic adaptation to network behavior, where it becomes the feedback delay of a control loop. Convergence of routing mechanisms until a new path is intact is also relatively slow.

Manuscript received June 1, 2010. This work is supported in part by the Higher Education Commission (HEC) of Pakistan.

*Muhammad Ali* is an Assistant Professor at the Computer Science Department in the Institute of Management Sciences, Peshawar, Pakistan (email: muhammad.ali@imsciences.edu.pk).

*Michael Welzl* is an Associate Professor at the Department of Informatics, University of Oslo, Norway (e-mail: michawe@ifi.uio.no).

As Van Jacobson, a well known researcher whose results had major impact on networks as a whole, stated in his ACM SIGCOMM 2001 keynote speech, the design of Internet protocols is largely governed by a focus on circuit rather than packet switching. In the early days of the Internet, the telephone network already worked well --- it appeared to be a natural choice to emulate its behavior. Now, things have shifted: the Internet has become the largest and most successful network in existence. According to Van Jacobson, the time has come to consider much more dynamic routing --- his vision is a network that resembles the power grid.

It appears that such things would do the network good, but they have not yet come into being. Some problems with ideas such as congestion based routing are still unresolved. As a simple example, if each sender chose path B because path A is congested, path B would become the congested one, leading to oscillations. Moreover, end-to-end congestion control as it is implemented in the Internet for reasons of scalability (we will explain this in section IV-A) makes implicit assumptions about the network path in use and would cease to work if paths would change all the time. These problems have not yet been solved, but Van Jacobson made it clear that this is research worth looking at; routing based on QoS constraints was also identified as unresolved but promising in [6]. From all of this, NoC designers can learn the following:

**Rule 1** *Consider mechanisms that mix routing with performance constraints or contention (congestion) avoidance, but be careful: while they are promising, the corresponding problems have not yet been solved in the Internet.*

One possible interpretation of this rule is that, while the general goal of such mechanisms may be the right one, simply applying, say, "hot potato routing" in an NoC may not be a good idea because these well known mechanisms lead to well known problems in networks that resemble the Internet. For instance, in hot potato routing, each router table contains minimum distance across the link. If a link goes down and new route takes longer than the previous, the node cannot update its routing table, as it increases path distances by using a moving average [7]. If the distance reported by a packet is larger than the link entry, the intermediate node will increase the table entry by some fractions. This way, the average distance will rise slowly with path failures but again would drop back once shorter paths become available.

Due to the unstable nature of the connectivity infrastructure of the internet, deterministic routing schemes do not prove to provide optimal output in most of the cases. However, unlike the internet, in NoCs the topology is more organized and homogeneous with short distance links providing more than one alternate paths for delivery of packets. Taking multiple paths can considerably reduce the network load along with providing alternatives for bad or broken links. Moreover, devices on-chip are unlikely to fail as frequently as in networks making it feasible to implement complex dynamic routing protocols to provide reliable and deterministic communication [8].

## IV. Scalability

### A. *The end to end argument*

Arguably, the most important design rule of the Internet is the "end to end argument" [9]. Strict adherence to this guideline is regarded as the primary reason for the immense scalability and thus, success of the network [10]. The end to end argument is frequently understood as follows:

> Complexities should be moved "out of the network" (towards network endpoints, upward in the stack), rendering the network itself as "dumb as possible".

While this interpretation is common, it is too limited --- still, it captures an essential part of what the end to end argument says. The misunderstanding may stem from the fact that the argument as described in [9] is applicable not only to computer networks but to systems design in general; for example, the design choices upon which the RISC architecture was built are very similar. A slightly altered version which directly refers to computer networks can be found in [11]:

> Functions required by communicating applications can be correctly and completely implemented only with the knowledge and help of the applications themselves. Providing these functions as features within the network itself is not possible.

According to this principle, some functions *must* be provided at the endpoints. Therefore, keeping the network as "dumb as possible" conforms to the end to end argument, but it is too restrictive --- for example, the end to end argument does not forbid using complicated functions to support routing. Additionally, the argument says that:

> Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.

Thus, functions should not be *redundantly* located in the network, but rather replicated where necessary only to improve performance [11]. In fact, [9] does not describe one single argument but several arguments. These arguments have several facets and, according to [12], two complementary goals:

- Higher-level layers, more specific to an application, are free to (and thus expected to) organize lower-level network resources to achieve application specific design goals efficiently (*application autonomy*)
- Lower-level layers, which support many independent applications, should provide only resources of broad utility across applications, while providing to applications usable means for effective sharing of

resources and resolution of resource conflicts (*network transparency*).

These two statements may in fact constitute the most concrete and directly applicable way to communicate the essence of the end to end argument without being too strict. We therefore deduce the following guideline for NoC designers:

**Rule 2** *Adhere to the end to end argument by ensuring application autonomy and network transparency.*

A good example is the reliability issue in NoCs. On-chip communication will face somewhat the same reliability issues that are common in the networks. Bringing packet based communication on-chip introduces new challenges in terms of fault tolerance. For instance, a transient fault may cause a bit flip either in the header or payload of the packet. In the former case, such an incident may lead the packet to be routed to a wrong destination whereas in the latter case, the contents of the packet are scrambled. In either case, a retransmission is required. To deal with such faults, a fault tolerance mechanism can be provided either at the link level or end-to-end level. At link level, each router needs to be possessed with error detection code which, after detecting an error, requests for a retransmission. In this case, each router has to maintain buffers to hold the packets until their successful delivery to the next stage. In end-to-end fault tolerance, buffers are only maintained at the end systems, that is, the source and destination and requests for retransmission originates from the destination [13]. Considering the fact that the soft error rate is assumed to be in the range of $10^{-9}$ and $10^{-20}$ BER (Bit Error Rate) [14], end-to-end error control seems to be a viable solution.

## B. *State and perflow State*

In modern networks with applications ranging from email to video and audio transmission or interactive online games, it becomes more and more interesting for ISPs to sell differentiated end user services which are accordingly paid for. Thus, following early efforts with ATM networks, the Internet “Quality of Service (QoS)” architecture “Integrated Services (IntServ)” was devised. Along with its special (albeit architecturally separated) reservation protocol “RSVP”, IntServ was designed to provide strict service guarantees at the granularity of individual data flows, but it failed as a commercial product. This failure was generally accredited to the fact that its scalability was limited by the fine service granularity which led to the design of “Differentiated Services (DiffServ)”, where packets are classified into traffic aggregates at domain endpoints in order to reduce the state for core routers [15].

Architectures such as IntServ require each router along a path to participate in the system; in particular, in order to provide guaranteed services to an end-to-end flow, an IntServ capable router has to be aware of it. Knowing a data flow means that each packet has to be classified based on multiple fields in the IP header (this process is known as “multi-field classification”) --- a table of source and destination IP addresses, port and protocol numbers must be maintained [15]. If no packet from a specific flow arrives for a long time, it makes sense to remove the flow from the list; this means that a timer must be kept and updated for each entry in the table.

Identifying flows can be tricky: IP packets can be carried within other IP packets (“IP-in-IP tunneling”), they can be encrypted (using “IPSec”) or fragmented. For example, it is possible that a fragment carries only one port number and a router must wait for the next fragment to arrive until it can fully classify the complete packet. Since fragments do not necessarily arrive one after another, this requires another table to be built and maintained. Altogether, the effort for routers is significant, and it grows with the number of flows in the system --- since memory is a limited resource, these tables cannot grow endlessly, and the number of flows that can be accommodated is limited.

On the other hand, DiffServ, a successor to IntServ, introduces a hierarchy: routers at the edge of a DiffServ “cloud” (typically an ISP domain) examine end-to-end flows and place them in a fixed number of aggregates. Routers in the core of the “cloud” only differentiate based on the aggregate, and hence, the architecture remains scalable. This leads us to the following rule:

**Rule 3** *Avoid state (and, in particular, per-flow state) in inner network nodes. If state is inevitable, consider introducing a hierarchy.*

The concept is illustrated in figure 1, where state aggregation is applied in a typical NoC mesh topology: sources S1, S2, S3 and S4 want to communicate with destination D. The data flows require special treatment for QoS from intermediate routers --- that is, R1 and R2 must be able to identify the incoming flows from the sources. The dashed arrows indicate aggregated state: R3 and R4 only need to know about the traffic aggregate that originates from R1 and R2, respectively, and D is only concerned with the traffic aggregate from R4. This method limits the amount of state that must be kept at each level of the hierarchy --- R4, for example, does not have to distinguish between four but only two individual data flows.

While DiffServ is a reasonable technical solution to the scalability problem, it did not lead to the envisioned success of end-to-end QoS in the Internet. The reasons are not of a strictly technical nature --- there are several other unresolved issues, like finding global agreements for charging high class versus low class customers, or convincing a peering ISP to make the first step in QoS deployment [6]. Unlike computer networks however, which are built for ongoing expansion, future growth and standards compatibility, on-chip networks can be designed and customized for a pre-known set of computing resources, given pre-characterized traffic patterns among them. The chip design is under control of a single administrative entity (the company which produces it), and therefore, the non-technical aspects of the Internet do not seem to apply here --- that is, QoS may work for NoCs.

A service is said to be guaranteed if a commitment is made, otherwise it is termed as “best effort”. SoCs may need to handle traffic ranging from real time data to regular data that have no specific constraints in terms of time and ordering. The idea is that services can be provided on the same silicon substrate for dealing with traffic classes that require commitments and classes without such requirements. Such a combined best effort and guaranteed services architecture for NoCs has been proposed in [4]. This seems to be quite feasible considering the desired efficient utilization of on-chip resources. However, the proposed architecture bears some resemblance to the IntServ mechanism devised for the Internet; if the traffic load requiring guarantees is high, it can let the tables at each node grow endlessly, hence posing a serious scalability problem due to memory shortage.

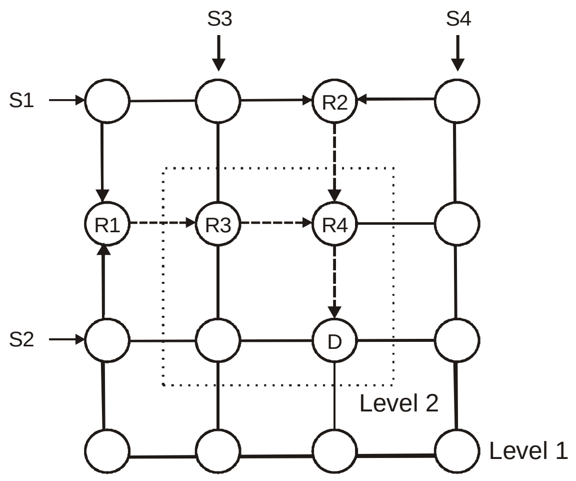

**Figure 1. State aggregation on an NoC**

DiffServ is a way to divide a global problem into local solutions. In SoCs, something similar is already done in a different context: problems like clock skews that are associated with global clocking have been well addressed by the adoption of GALS (Globally Asynchronous Locally Synchronous), where the system is divided into separate blocks, each of which has its own independent clocking mechanism [16]. This trend has emerged due to the need for IP reuse and platform-based design for chips and it is quite reminiscent of the DiffServ method to create global services from local building blocks.

To summarize, not only does DiffServ provide the scalability that the aforementioned on-chip QoS architecture does not show, it also appears to be in line with other SoC design concepts; this makes a strong point for applying rule 3 on NoCs.

## V. HETEROGENEITY

The Internet has managed to cope with heterogeneity in an impressive manner. It is available in wired and wireless environments in all countries on the globe and probably soon also on other planets (the "Interplanetary Internet" is ongoing NASA research), with connected devices ranging from desktop PCs to robots and cell phones. This is possible because the part of the TCP/IP stack that *must* be implemented is relatively small, and the Internet Protocol (IP) serves as a binding element between a variety of technologies above and underneath it.

This fact is best explained with the "IP Hourglass" (see fig. 2): all kinds of protocols and applications can be used on top of IP, and IP does not make any assumptions about lower layers (leading to a service that is often referred to as "best effort"). A common phrase in this context is "everything over IP, IP over everything". This view has replaced the notion of network layers in the sense of OSI in the Internet community: IP packets may even carry IP packets to form an "overlay network" such as a VPN, and so may upper layer protocols (e.g., in the case of SSL-based VPNs).

IP itself was designed to be slim, and the IETF made a serious effort not to include unnecessary functionality in the newer version of the protocol. Note however that "best effort" service contradicts the vision of differentiated and guaranteed services for end users --- in fact, the heterogeneity of technology underneath IP may just be another reason for the failure of end-to-end QoS in the Internet.

"IP over everything" was the path chosen by the community a long time ago, and it led to the broad applicability we have today, but it also has its disadvantages. One particular problem with IP is the fact that, once everybody has agreed upon a common binding element, it is hard if not impossible to change it. This became very obvious when the IETF decided that the envisioned maximum number of IP addresses will not suffice to accommodate the growing number of connected devices in a few years. A solution ("Internet Protocol, Version 6" (IPv6)) was designed, but it is still hardly used in the Internet despite the serious effort of both the research community and the industry to convince users, application and operating system developers as well as ISPs to upgrade their systems. Meanwhile, the IP address problem has been solved by local interim mechanisms that are considered as bad design in the IETF but were still standardized due to their wide deployment: "Network Address Translators" (NATs) [17]. The lesson is quite obvious:

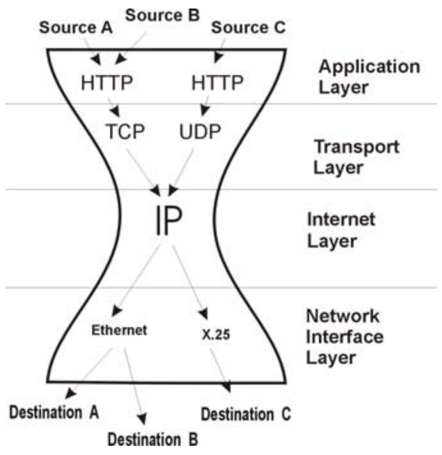

**Figure 2. IP hourglass**

**Rule 4** *Introduce a network protocol as a slim "binding element" if it is necessary to* *accommodate heterogeneity at upper (application) and lower (communication technology) network layers. Carefully specify and standardize it: it may not be possible to change it afterwards.*

In [4], the authors have given a somewhat related argument where they propose that NoCs may provide a separation between application and communication at the transport layer since it is the first "network independent layer in the OSI model". It further states that the NoCs will provide small set of services yet allowing the upper layers to provide differentiated services. The authors acknowledge their argument's similarity with the internet where IP plays a similar role. Our defined rule 3 basically emphasizes upon the standardization policy and its possible implications.

In terms of NoCs, most of the research is concentrated at the architectural design level and little has been said about the software layers. Various proposals have already been produced for effective communication paradigm in a NoC. In most of the cases, the network architecture is assumed to be homogeneous with same speed links and identical switches. However, this network is supposed to be integrated with set of resources which necessarily will be heterogeneous in nature. The idea is to reuse the existing IP (Intellectual Property) blocks from different vendors and connect them to the on-chip network fabric. In order to create transparency among the communicating partners, a Network Interface (NI) is supposed to be placed between the resource and the network switch. The NI is a very complex piece of architecture which does a whole lot of things ranging from creating packets from bits, controlling the network load, and ensuring safe delivery of data to the resources. Simply saying that the NI would deal with all kind of such intermediate adjustments would be a naive imagination as we know that various on-chip resources with distinct characteristics would be a big integration liability. The problem may get worst when a whole block of resources is connected to a single switch. This will increase the complexity of the NI architecture which, due to limited storage and processing capabilities available on-chip, would be highly undesirable. Furthermore, things can become more complicated when NoCs are connected to other NoCs each of which may be originating from different vendors — *NoCs treated as IPs*. Hence, keeping in view the above-mentioned discussion, it is important to distribute the set of services in such a way that the underlying network architecture in a NoC remains light and reusable.

## VI. Conclusion

From the perspective of network researchers, NoCs are a chance to begin with a clean slate: design must be carried out carefully in order to avoid repeating mistakes that have already been overcome in other areas. In this article, we noted that NoCs resemble the Internet in that they share requirements regarding efficiency, scalability and heterogeneity. In an effort to help NoC designers to learn from the past, we looked at aspects of these areas and deduced rules ("lessons from the Internet") that we believe to be relevant. We only focused on a scenario where presumably, all the links are of the same speed. However, when it is intended to connect two or more NoCs where each one acts as an IP block, then issues like congestion may arise because of discrepancies that might exist in the communication and computational architecture. Such problems have long been known in the Internet and a lot has been written and proposed to deal with them; these solutions might become relevant as NoCs grow. This reminds us that, in the future, we should look over the fence every once in a while and take a glance at solutions from seemingly foreign research areas.

## References

[1] William J. Dally, Brian Towles, "Route Packets, not wires: On-chip Interconnection Networks", *Proceedings, Design Automation Conference (DAC)*, June 2001, Las Vegas, NV, pp. 684-689.
[2] Luca Benini, Giovanni De Michelli, "Networks on chips: A New SoC Paradigm," *IEEE Computer Society Press*, Jan. 2002, pp. 70 – 78
[3] E. Rijpkema, K. G. W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade-offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip", *Proceedings, Design and Test in Europe (DATE)*, 2003.
[4] Axel Jantsch, Hannu Tenhumen, "Networks on Chips", *Kluwer Academic Publications*, 2003, Boston, USA.
[5] Axel Jantsch, "NoCs: A new Contract between Hardware and Software", *Euromicro Symposium on Digital Systems Design (DSD)*, September 01 - 06, 2003, Belek-Antalya, Turkey
[6] G. Huston, "Next Steps for the IP QoS Architecture", *RFC 2990*, November 2000.
[7] Marc Elliott Mosko, "Routing in Mobile Adhoc Networks", *PhD dissertation*, University of California, Santa Cruz, 2004.
[8] Muhammad Ali, Michael Welzl, Martin Zwicknagl, Sybille Hellebrand, "Considerations for fault tolerant Network on chips, *Proceedings, 17th International Conference of Microelectronics (ICM), Islamabad 2005.*
[9] J. H. Saltzer, D. P. Reed, D. D. Clark, "End-to-End Arguments in System Design", *Proceedings, Second International Conference on Distributed Computing Systems*, April 1981, pp. 509-512.
[10] B. Carpenter, "Architectural Principles of the Internet," *RFC 1958*, June 1996.
[11] James P.G. Sterbenz, "Protocols for High-Speed Networks: A Brief Retrospective Survey of High-Speed Networking Research", *Proceedings, PFHSN 2002* (Protocols for High Speed Networks 2002 - IFIP TC6 WG6.2 / IEEE Comsoc TC on Gigabit Networking), Springer Verlag, Berlin, Germany, 22-24 April 2002.
[12] David P. Reed, Jerome H. Saltzer, David D. Clark, "Comment on Active Networking and End-to-End Arguments", *IEEE Network* 12, 3 (May/June 1998), pages 69-71.
[13] William Dally and Brian Towles, "Principles and Practices of Interconnection Networks", *Morgan Kaufmann Publishers*, first edition, 2004, USA.
[14] Michael L. Bushnell, Vishwani D. Agarwal, "Essentials of Electronic Testing for Digital, Memory and Mixed-Signal Circuits", *Kluwer Academic Publishers*, USA, 2000
[15] Grenville Armitage, "Quality of Service in IP Networks: Foundations for a Multi-Service Internet", *Macmillan Technical Publishing*, April 2000.
[16] Daniel M. Chapiro, "Globally-Asynchronous Locally- Synchronous Systems", *PhD thesis*, Stanford University, Oct 1984.
[17] K. Egevang, P. Francis, "The IP Network Address Translator (NAT)", *RFC 1631*, May 1994.