

# Design principles of the Recursive InterNetwork Architecture (RINA)

Eduard Grasa<sup>1</sup>, Eleni Trouva<sup>1</sup>, Patrick Phelan<sup>2</sup>, Miguel Ponce de Leon<sup>2</sup>, John Day<sup>3</sup>, Ibrahim Matta<sup>3</sup>, Lubomir T. Chitkushev<sup>3</sup> and Steve Bunch<sup>4</sup>

1

i2CAT Foundation, Jordi Girona, Barcelona, Spain  
[eleni.trouva@i2cat.net](mailto:eleni.trouva@i2cat.net), [eduard.grasa@i2cat.net](mailto:eduard.grasa@i2cat.net)

2

TSSG, Waterford Institute of Technology, Ireland  
[pphelan@tssg.org](mailto:pphelan@tssg.org), [miguelpdl@tssg.org](mailto:miguelpdl@tssg.org)

3

Computer Science, Boston University, Massachusetts, USA  
[day@bu.edu](mailto:day@bu.edu), [matta@bu.edu](mailto:matta@bu.edu), [lrc@bu.edu](mailto:lrc@bu.edu)

4

TRIA Network Systems, LLC, Illinois, USA  
[steve.bunch@ieee.org](mailto:steve.bunch@ieee.org)

## Introduction

RINA design principles (or patterns) were first presented by John Day in his book ‘Patterns in Network Architecture: A return to Fundamentals’ [1]. This work is start afresh, taking into account lessons learned in the 35 years of TCP/IP’s existence, as well as the lessons of OSI’s [2] failure and the lessons of other network technologies of the past few decades, such as CYCLADES [3], DECNET [4] or XNS [5]. He has made some key observations that point to a new direction.

RINA departs from the basic premise that “*networking is inter-process communication (IPC) and only IPC*” [6]. Networking provides the means by which processes on separate computer systems communicate, generalizing the model of local inter-process communications. In an operating system, to allow two processes to communicate, IPC requires certain functions such as locating processes, determining permission, passing information, scheduling, and managing memory. Similarly, two application processes residing on different systems communicate and share state information by utilizing the services of a Distributed IPC Facility (DIF). Figure 1 shows different examples of application processes A and B communicating i) within a single system, ii) between two directly connected systems and iii) between two systems connected by a router. Each scenario requires one or more DIFs, each one optimized for providing IPC services over a certain scope (local, single link, network).

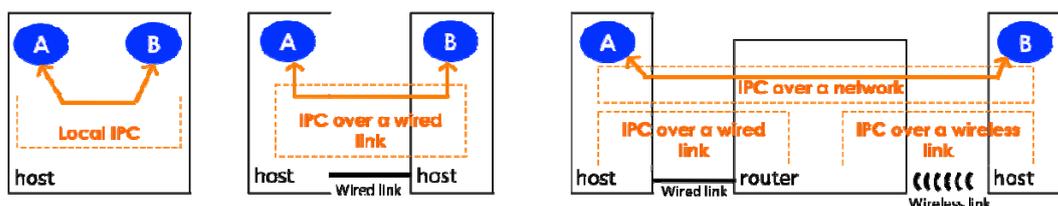
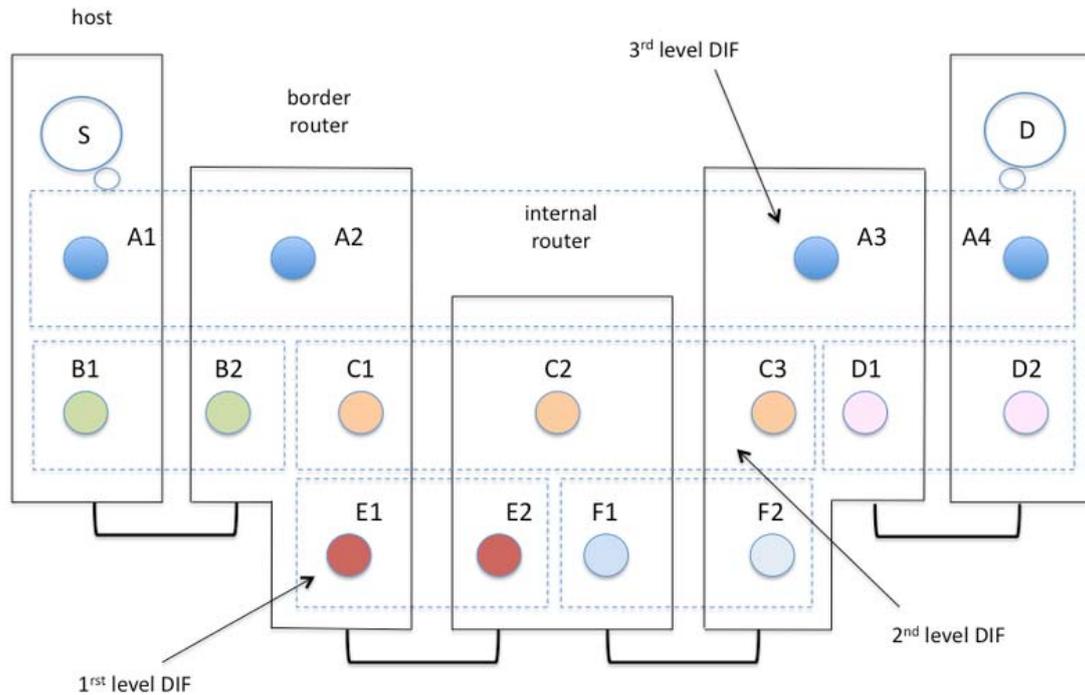


Figure 1. IPC over different scopes

A DIF is an organizing structure, grouping together application processes that provide IPC service and are configured under the same policy. A DIF can be seen as what we generally refer to as a "layer". According to this view, networking is not a layered set of different functions but rather a single layer of distributed IPC that repeats over different scopes, i.e. providing the same functions/mechanisms but tuned under different policies to operate over different ranges of the performance space (e.g. capacity, delay, loss).



**Figure 2. An example of the RINA architecture**

Figure 2 illustrates an example of the RINA architecture. Each DIF provides IPC services over a limited scope. First level DIFs operate on top of a physical medium, and its policies are optimized to deal with the particularities of the physical medium. First level DIFs provide IPC services to second level DIFs, and so on. The protocols at each layer are the same; they just use a different configuration (i.e. policies) to fulfil the particular requirements of the layer. In essence a DIF is just a distributed application, whose members (application processes called IPC processes) are specialized towards providing distributed IPC services. Therefore DIFs are not structurally different to any other distributed application, they just perform a very concrete task.

The DIF is the basic mechanism of RINA. It is a black box that operates among multiple systems; the IPC process thus runs on every system that belongs to the DIF. A DIF enforces strict layer boundaries: What happens inside the DIF is not visible outside of the DIF. What is visible at the top of a DIF is the service requested of the DIF; what is visible at the bottom is the service requested by the DIF of the DIF beneath it, if it isn't the bottom one.

## Design Principles

The design principles presented in this document address a number of the fundamental limitations presented in the document “Fundamental Limitations of current Internet and the path to Future Internet” [7]. Specifically:

- Processing and Handling Limitations
  - Lack of data and service identity (Principle 3)
  - Lack of methods for dependable, trustworthy processing and handling of network and systems (Principle 4, Principle 5, Principle 6)
- Transmission Limitations:
  - Lack of integration of devices with limited resources to the Internet as autonomous addressable entities (Principle 1, Principle 2)
  - Security requirements of the transmission links (Principle 4)
- Control limitations:
  - Lack of flexibility and adaptive control (Principle 1, Principle 2)
  - Segmentation of data, services and control (Principle 1)
  - Lack of efficient congestion control (Principle 1)
  - Support for mobility (Principle 1, Principle 3)
- Multiple categories limitations:
  - The current inter-domain routing system is reaching fundamental limits (Principle 1, Principle 3)
  - Security of the whole Internet architecture (Principle 4)

### *Principle 1. Recursive Layers: the ends are relative*

Protocol functions tend to alternate as one goes up the stack. This alternation reflects a repeating unit consistent with interprocess communication. The error and flow control protocol breaks up into two functions, data transfer, which sends data forward, and data transfer control, which provides feedback from the receiver. These functions both happened in both X.25's layer 2 and 3 protocols, and it also describes the relationship between IP (data transfer) and TCP (data transfer control). Applications themselves have a common pattern too.

Lower layers thus tend to do the same things as upper layers, but on a more local scale, with more flows aggregated into them. At the application layer, a single instance frequently communicates between two computers. At the other extreme, a large network backbone link may carry thousands of processes, headed between many different computers, part of the way.

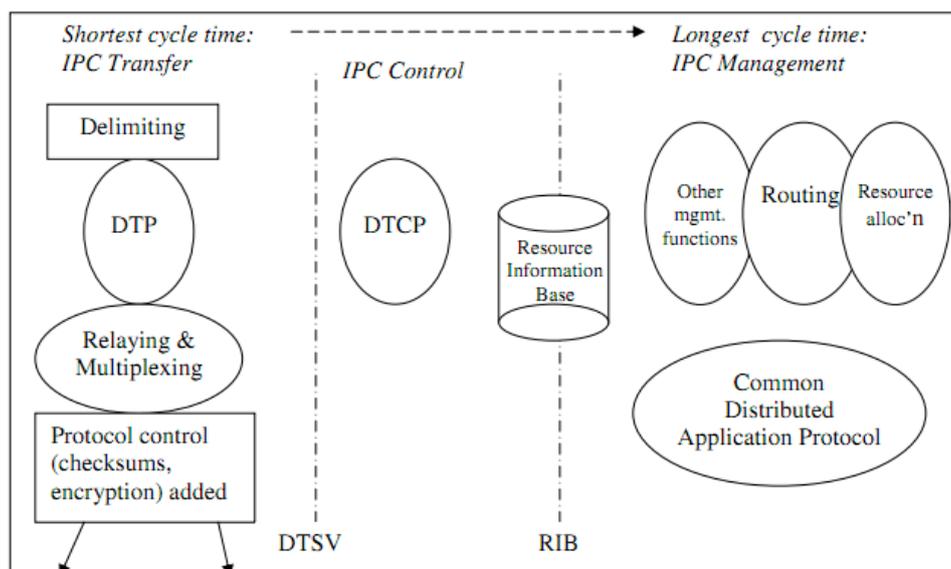
This leads to the first principle of our proposed new network architecture: *Layers are recursive*. The same data transfer protocol can be used repeatedly in a protocol stack, encapsulating each layer in another instance of itself. There is thus no need for purpose-built protocols for each layer. There is also not a fixed number of layers in the stack. The number of layers between the application and the physical medium is variable; at any given point, there are simply as many as needed, no more, no less. But any given implementation only needs to deal with itself, the layer above it, and the layer below it. The actual depth of the stack is essentially invisible.

Layers are not the same thing as protocols; more than one protocol can make up a layer. Because the same group of protocols is used repeatedly, the implementation is simpler than the TCP/IP stack.

There's no need for separate protocols for "layer 2", "layer 3", etc. This design principle is opposed to the "functional layering" principle of the current Internet, where there is a fixed number of layers and each layer in the stack performs a different function. Functional layering assumes that each function can only be performed once, and has an optimal placement in the network stack. The recursive layering model recognizes the fact that networking problems such as route discovery, flow control, name-to-address mapping, and location discovery don't have to be solved once and only once for the whole Internet. It's much more beneficial, in fact, to solve them separately in each management domain through which a packet moves. For example, let's think about error correction. Error correction, for example, is implemented at least four times in modern wireless systems: at the level of coding, at the data link layer, in the TCP layer, and in the application space for P2P protocols such as BitTorrent. This is simply because of the fact that errors creep into a system for many different reasons and have to be corrected on the time scale pertinent to the type of error.

The recursive layering principle also provides some insights about the so-called "end-to-end principle" [8]. With a recursive model in mind, it becomes clear that the "ends" are always relative to the scope of the layer we are talking about. For example, in a 1<sup>st</sup> level DIF managing a wired physical link (on top of the physical medium) the ends are the processes controlling the physical interfaces at both ends of the link (what we today call the 'drivers'); if we look at a video streaming application the ends are the sending and receiving application processes. The same consideration applies to all the DIFs in between.

The protocols that make up the basic RINA layer include the Data Transfer Protocol (DTP) which relays the payload in the forward direction. Its contents includes addressing information and protection information (a checksum and a time-to-live counter which detects routing loops). The Data Transfer Control Protocol (DTCP) performs the error and flow control functions, sending feedback from destination to source. Note that DTP has the payload, which is visible outside of the layer, while DTCP operates entirely within the black box. A layer also has a management protocol.



**Figure 3. Functions of an IPC process, executed over different timescales**

An instance of DTP and an instance of DTCP fall within a single DIF, a single layer. DTP functions operate over the shortest time frame. DTCP functions operate over a slightly longer time frame; the

two share information via the Data Transfer State Vector (DTSV). The error and flow control functions of DTCP might only be performed at the edges, as with TCP over IP, but there's no formal layer boundary between them. A DIF also includes a number of management functions, which among other things include setting up routing within the DIF or adding new members (IPC processes) to the DIF.

RINA also only requires a single Application Protocol, the Common Distributed Application Protocol. Just as there are patterns visible in the other layers, all applications can be shown to need only six fundamental operations that can be performed remotely: Create/delete, read/write, and start/stop. What changes from application to application is in the objects being manipulated, not the protocol. So the protocol remains stable, while new applications are accommodated easily by changing the object models. The common application protocol is used within the DIF for layer management functions. These include enrollment (initializing the DIF) and security management, port allocation, access control, QoS monitoring, flow management, and routing.

*Principle 2. Separate mechanism from policy: design for multiple functions*

TCP/IP was designed for the data networking functions of the day, and was not intended to replace the telephone network or for that matter the cable TV network. VoIP and IPTV are essentially done by brute force. RINA starts by asking the question, can a protocol be effective for multiple functions with very different requirements? The answer, of course, is yes. It's not hard to effectively serve different applications if that's the goal from the beginning of the design.

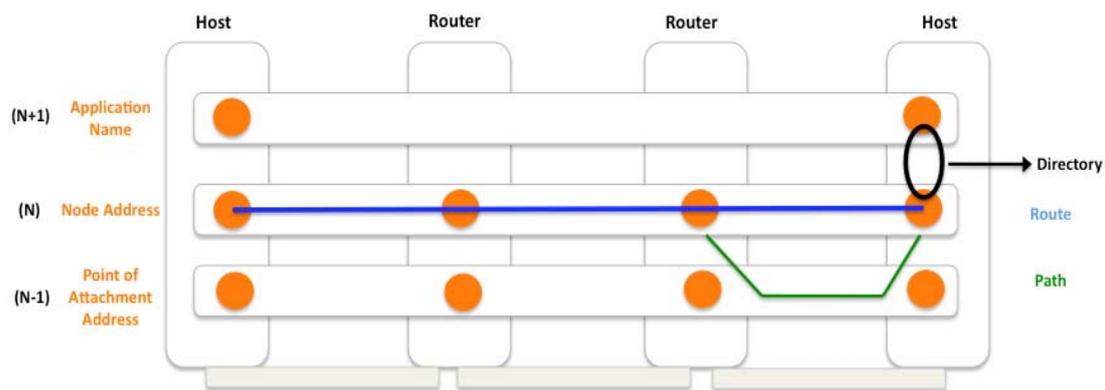
RINA does this by having many plug-ins (policies) that control how the DTP and DTCP work. One of the key differentiators of RINA vs. TCP is its support of Quality of Service (QoS) options. These include parameters such as allowable packet loss rate, error rate, delay, and jitter (delay variance). A DIF can operate in a "best effort" mode, like IP. In a recursive stack, only the top DIF might need to do the end-to-end error correction. That's how TCP/IP works and it's fine for many applications.

In RINA, policy and mechanism are separated. The mechanism of a protocol may be tightly bound, such as the headers of a data transfer packet, or loosely bound, as in some control and acknowledgment messages. Different applications may need different policies. Flexibility comes from being able to use common mechanisms in conjunction with different policies, rather than needing separate protocols [9]. The syntax of protocols dictates neither policy nor mechanism. Then each DIF can use different policies to provide different qualities of services and to further adapt to the characteristics of the physical media (if the DIF is close to the lower end of the stack) or to the characteristics of the applications (if the DIF is close to the upper end of the stack).

A DIF can be asked to provide a low-loss connection for a specified level of capacity, such as might be required for an audio or video stream. Or it can be asked for a best-effort (unspecified QoS) connection. RINA simply considers QoS to be a set of parameters plugged in to the DIF, and the specific mechanisms that it uses to provide it are an implementation detail, hidden inside the black box. If the DIF determines that it can't deliver the requested QoS, it rejects the request.

*Principle 3. Complete naming and addressing architecture: support for mobility and multihoming*

The current Internet architecture does not provide separate names for the basic entities in the architecture: nodes, points of attachment to the network (PoAs) and applications. The only names provided are PoA names (IP addresses). Although commonly referred to as “host addresses”, they are in fact interface addresses [10]. This defect originates in the ARPANET’s design where host addresses were named after the IMP port number (interface) [11]. The result is that the network has no means to understand that two or more IP addresses of a multi-homed node belong to the same node, making multi-homing hard to realize. The same choice, naming the interface and not the node, forces the Internet to perform routing on the interface level contributes to much bigger routing tables than necessary. While the problem became apparent very soon, in 1972, when Tinker Air Force Base in Oklahoma joined the Net and wanted two connections for reliability, it was never changed. Mobility, which can be seen as dynamic multi-homing with expected failures, is the next feature that suffers from having an incomplete naming schema. Numerous other problems arise as well when more sophisticated distributed applications are attempted beyond the trivial client-server schemes. It should be noted that every other network architecture (XNS, CYCLADES, DECNET, OSI, etc) developed in the 1970s and 80s avoided this problem.



**Figure 4 Complete naming and addressing architecture**

In 1982, Jerry Saltzer in his work “On the Naming and Binding of Network Destinations” [12] documented what XNS and CYCLADES were doing, noting the entities and the relationships that make a complete naming and addressing schema in networks. According to Saltzer there are four elements that need to be identified: applications, nodes, points of attachment to the network (PoAs) and paths. A service can run to one or more nodes and should be able to move from one node to another without losing its identity in the network, as in Operating Systems where file names should not change if stored on a different disk. However, Saltzer failed to notice that because nodes can have more than one path to the same next hop routing has to be on the node to prevent a combinatorial explosion. XNS and CYCLADES did not make this error because they simply routed on the address in the layer doing the relaying. As illustrated in Figure 4, a directory maps an application name to a node address, routes are sequences of nodes addresses and paths result from mapping the node addresses to PoAs of nearest neighbours. In this way, routing is a two-step process: First, we have to calculate the route, which is a sequence of node addresses and then, for each hop choose the appropriate PoA to select the specific path to be traversed. Although, clearly CYCLADES and XNS had complete naming and addressing

schema, the TCP/IP protocol suite followed the ARPANET approach, causing many of the problems of today's Internet. An analogy would be to an operating system with only physical addresses and applications named by macros for jump points in low memory addresses.

RINA adopts the complete naming and addressing schema, bringing it on step further: because layers (DIFs) are recursive, the roles of "application", "node" and "point of attachment" are relative to their position in the stack of layers. For example, given a DIF at level N (N-DIF), the IPC processes at level N+1 are application processes for the N-DIF, and the application processes at level N-1 are points of attachment for the N-DIF. If we look at it from the perspective of the N-1 DIF, now the IPC processes at the N-DIF are the applications.

In RINA all the application processes (including IPC processes) have a name. Within a DIF, each IPC processes is assigned a synonym, called 'address', whose scope is only the DIF and it is not visible outside. This synonym is the name used for routing within the layer, and it is optimized for that purpose. Since it's not visible to the end user, it is only useful if it serves some purpose inside the black box. Hence the address is likely to be topological: It conveys information about how to reach the destination. Two adjacent addresses are likely to be very near each other. An address itself could be computed based upon the graph of the network. This is most useful for large DIFs, of course, and it doesn't take the place of a routing algorithm; it merely simplifies the task.

The size of the address is a parameter of the DIF. A small DIF (like a LAN) may just use very short addresses, with each node knowing how to reach every other one. A point-to-point link is itself a DIF, one which doesn't need an address at all. At the other extreme, the entire World Wide Web, or all of the public servers on the Internet, could be cast as a DIF, and it might need relatively large addresses. But they would still be invisible to the application. An implementation of RINA would take care of addresses automatically; they would not be assigned via human intervention, or subject to an addressing authority.

#### *Principle 4. Security by design*

RINA incorporates many security features that make its design inherently more secure than current networks [13]. These are:

- Application processes cannot communicate with each other without authenticating. The strength of the authentication is a policy of the application, can vary from none to multiple handshakes using sophisticated cryptography. This way destination application processes always know the application process that is requesting the communication, and can make an informed authorization decision whether to accept or not the communication.
- Addresses are internal to a DIF. Each DIF is a black box, all the entities external to a DIF cannot address its members. Joining a DIF requires authentication (the strength of the authentication is a policy of that DIF), therefore having rogue members within a DIF is less probable than in today's Internet.
- Application data sent through a DIF can be encrypted (a policy of the application), making Deep Packet Inspection impossible or much harder to achieve.

### *Principle 5. Separation of port allocation from synchronization in the transport protocol*

The basis of DTCP is the delta-T protocol [14], invented around 1978 by IBM's Dick Watson. He proved that the necessary and sufficient conditions for reliable transfer is to bound three timers. Delta-T is an example of how this should work. It does not require a connection setup (such as TCP's SYN handshake) or tear-down (like TCP's FIN handshake) for integrity purposes. In fact, TCP uses the same three timers! So RINA lacks unnecessary overhead. Watson showed that synchronization and port allocation are distinct functions. Port allocation is part of DIF management, while synchronization is part of the data transfer protocol. In fact, maintaining the distinction also improves the security of connection setup.

### *Principle 6. Integrating connectionless and connection-oriented networking*

Connection oriented vs. connectionless has always been a hot debate in the networking community. Both approaches have its pros and its cons, so it seems natural that there will be situations where using connections make more sense and some others where using a connectionless approach is more efficient. For example, as traffic becomes denser (closer to the network core) connections become more effective; when traffic is more stochastic (closer to the access networks) connectionless is more effective. Therefore, why should this be a debate at all? We should try to find a model that accommodates both connection and connectionless, and even more, that saw pure connections and pure connectionless as the extremes of a continuous function.

The first part is having a service interface (the DIF interface) that abstracts both options, so that a user of the network can request both approaches through the same interface. Well, in fact the user shouldn't care at all. Distributed applications use the network to communicate, they need communication resources to be allocated to them, with certain characteristics (bandwidth, latency, jitter, packet loss ratio, ... ) commonly known as Quality of Service. If the network uses a connection or connectionless approach, the application doesn't matter, as long as its demands are met. RINA provides the following API:

- **allocate** - Allocates resources to support a flow between source and destination application processes with a certain quality of service. A port-id is returned as a handle for the allocation.
- **send** - Sends an amount of data to the destination application process on the specified port. The amount of data passed across two layers to be transferred to the destination is referred to as a Service Data Unit (SDU). An SDU may be fragmented or combined with other SDUs.
- **receive** - Receives an SDU from the destination application process on the specified port.
- **deallocate** - Terminates the flow and frees all the communication resources allocated to it.

The allocate primitive allocates communication resources for the requesting application to communicate to the destination application. These communication resources must meet the quality standards defined by the QoS parameters requested by the application. An identifier for the allocated resources (port\_id) is returned to the requesting application, who can use it to read and write bytes. The deallocate primitive frees all the resources previously allocated. By using this API, the decision of using connections or connectionless mechanisms to accommodate the application requests is a decision of the network, and depends on the traffic characteristics and the QoS requested.

Having defined the interface, we need to find an implementation that can see connections and pure

connectionless operation as extremes of a continuous function. Connections try to minimize the amount of routing state in the network. Although this may not look intuitive, let's see what happens when a connection fails. If a node in the connection path breaks, the rest of the nodes participating in the virtual circuit don't have enough information on what to do, and therefore cannot take an action to repair the problem. On the contrary, in a pure connection-less network all the nodes know how to route all the packets (the packets contain enough information for the nodes to take routing decisions), therefore the network can react on node or interface failures.

We can see connections as minimal distributed state and datagrams as maximal distributed state. Another way of looking at it is that with connection-oriented resource allocation, information (including routing) is only stored with nodes on the path of the connection; whereas with connectionless resource allocation, information is stored with every node. Let's consider *this* to be the function we were looking for: The probability that this PDU is processed precisely like the last PDU in this flow varies between 0 and 1.

Based on information from other routers, each router computes the probability it will be on the path for this flow and if so what resources should it "allocate." This leads to a model where flows are spread across different paths and resources are allocated on the basis of the probability of their use. Pure connectionless is represented by each path being equally likely or  $1/n$ . Distributions may be anything in between. The question is what resources should be allocated for these probabilities given the desired QoS. Routers allocate resources based on the probability that they will receive PDUs for a flow. This allows routers to reserve resources to better respond to failures during the transients while resource-allocation information is updated in response to a failure.

## Acknowledgements

The authors would like to acknowledge Fred Goldstein and Richard Benett, who have contributed to this work through the reports "Moving beyond TCP/IP" [15] and "Designed for change: End to End Arguments, Internet Innovation and the Net Neutrality debate" [8].

## References

- [1] J. Day: "Patterns in Network Architecture: A Return to Fundamentals, Prentice Hall, 2008.
- [2] H. Zimmermann: "OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection". IEEE Transactions on Communications, vol. 28, no. 4, April 1980, pp. 425 – 432.
- [3] L. Pouzin: "Presentation and Major Design Aspects of the Cyclades Computer Network". Proceedings of the NATO Advanced Study Institute on Computer Communication Networks. Sussex, United Kingdom: Noordhoff International Publishing. pp. 415-434, 1973.
- [4] DIGITAL Equipment Corporation, Distributed Data Processing Group: "DECNET Technical summary", 1980. Available online at <http://bitsavers.org/pdf/dec/decnet/>
- [5] Xerox System Integration Standard: "Internet Transport Protocols". XSIS 028112, December 1981

[6] J. Day, I. Matta, K. Mattar: “Networking is IPC: A Guiding Principle to a Better Internet”, Proc. of ReArch08, Madrid, SPAIN, 2008.

[7] EC FIArch Group: “Fundamental Limitations of current Internet and the path to Future Internet”, 2011. Available online at [http://ec.europa.eu/information\\_society/activities/foi/docs/current\\_internet\\_limitations\\_v9.pdf](http://ec.europa.eu/information_society/activities/foi/docs/current_internet_limitations_v9.pdf)

[8] R. Bennet: “Designed for Change: End to end arguments, Internet Innovation, and the Net Neutrality Debate”, Information Technology and Innovation Foundation Report. 2009. Available online at <http://www.itif.org/files/2009-designed-for-change.pdf>

[9] K. Mattar, I. Matta, J. Day, V. Ishakian, and G. Gursun: “Declarative Transport: A Customizable Transport Service for the Future Internet”. In Proceedings of the 5th International Workshop on Networking Meets Databases (NetDB 2009), co-located with SOSP 2009. Big Sky, MT, October 14, 2009. [PDF]

[10] R. Hinden, S. Deering: “IP Version 6 Addressing Architecture”, RFC 2373. July 1998.

[11] F. E. Heart, R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden: “The interface message processor for the ARPA computer network”, Proceedings of the Spring joint computer conference (AFIPS), ACM, New York, NY, USA, 551-567, 1970.

[12] J. Saltzer: “On the Naming and Binding of network destinations”, RFC 1498.

[13] G. Boddapati, J. Day, I. Matta, L. Chitkushev: “Assessing the Security of a Clean-Slate Internet Architecture”. Technical Report BUCS-TR-2009-021, CS Department, Boston University, June 22, 2009

[14] R. W. Watson: “Delta-t protocol specification”, tech. rep., Lawrence Livermore National Laboratory, 1981.

[15] F. Goldstein, J. Day: “Moving beyond TCP/IP”. 2010. Available online at <http://pouzin.pnanetworks.com/images/PSOC-MovingBeyondTCP.pdf>