computers

MDPI

Editorial

Special Issue "Post-IP Networks: Advances on RINA and other Alternative Network Architectures"

John Day [1,*], Eduard Grasa [2] and Peyman Teymoori [3]

1 Metropolitan College, Computer Science, Boston University, Boston, MA 02215, USA
2 Fundacio i2CAT, Software Networks Research Area, 08034 Barcelona, Spain; eduard.grasa@i2cat.net
3 Digital Infrastructures and Security (DIAS), University of Oslo, Problemveien 7, 0315 Oslo, Normay; peymant@ifi.uio.no
* Correspondence: jeanjour@comcast.net

Received: 2 October 2020; Accepted: 2 October 2020; Published: 13 October 2020

Over the last two decades, research funding bodies have supported "Future Internet", "New-IP", and "Next Generation" design initiatives intended to reduce network complexity by redesigning the network protocol architecture, questioning some of its key principles. Industry groups such as the ETSI ISG NGP are looking at alternatives to the current "TCP-IP" protocol suite. However, few initiatives have really been able to "clean the slate" and question the core model and underlying principles of current Internet protocols. Of those who have done it, RINA—the recursive internetwork architecture—is probably the simplest, yet most general solution.

RINA implementations are maturing while researchers keep working on characterizing its implications for a wide variety of current networking challenges. During the last year, RINA has been featured in an ETSI ISG NGP group report as a potential candidate for future-proof protocols, and the standardization work on its core specifications at ISO has advanced to Committee Draft Standards. New projects are developing RINA expertise to implement and deploy the technology in a variety of markets.

This Special Issue investigated the application of RINA in different types of network segments and applications, papers reporting on prototype implementations, experimental deployments, and interoperability with existing technologies. Selected papers presented at 7th International Workshop on the Recursive InterNetwork Architecture (RINA 2020) were also invited.

The meeting was well attended and there were a number of interesting RINA-related topics discussed.

However, before we get to that we should spend some time explaining what RINA is.

If you have heard of RINA, you probably think it is "yet another" of the Future Internet proposals perhaps a little different from the others.

While RINA points in a different direction and very simply solves many, if not all, of the "new requirements," its origins are quite different. They harken back to the networking of the early 1970s before networking "lost its way."

The Origins of RINA

What became RINA started in the mid-1990s, when one of us (Day) decided that after more than a quarter century on the Net, "I needed to figure out what I actually knew about networking, independent of politics, current technology-constrains, installed base, myth, dogma, convention, etc. What did the **problem** say? We needed to really understand, not our kludges, compromises and mistakes, but what the **problem** said had to be the case. What structure held it all together in a complete whole? How could we move ahead and avoid wrong turns, if we didn't know where we were going? If we didn't understand more clearly what the ultimate goal had to be?"

We will let Day continue the story: Without that, we are simply wandering lost in the forest. Many would say (and did) that no such thing existed. There was a conservation of complexity that simplifying in one place merely created complexity elsewhere. On the surface that may seem to be all that is possible, but there were signs that was not the case with networking. As luck would have it, we had seen a completely worked example where that was not the case.

In 1970 at the University of Illinois, we were building the first supercomputer. While this exposed us to parallel processing, it also gave us intimate access to one of the finest examples of system design ever, the Burroughs 5500 and its second generation, the 6700, a system of which Organick said, "It appears they got everything right."

The system required we eat, drink, and breathe recursion!

Thinking recursively became second nature.

(I would contend that thinking recursively facilitates seeing distinct cases as degenerate cases of a whole even when it is not strictly recursive.)

There was a consistent logic, a rationale, a gestalt that permeated the entire system. If you knew how A and B worked, you could guess how C would work and be right. This was inspiration that simple elegant designs, devoid of special cases and kludges, was possible. One just had to learn to listen to what the problem was telling you. The problem is always smarter.

During my decade plus as Rapporteur of the OSI reference model, I had seen patterns go by that looked promising but were not aligned with various political agendas, especially those of the PTTs, which were often met by the canard cited above to avoid change "if we simplify here, ..."

My sense of the problem said they were wrong, but I could not prove it.

I needed to work across the whole problem to show it did not get more complex.

There were three fundamental areas that appeared to be key that needed to be explored:

1. separation of mechanism and policy in networking,
2. naming and addressing, which was still not well understood, and
3. finding a synthesis of the connection/connectionless dichotomy that did not make it an either/or choice.

The developers of the ARPANET had been quite successful in taking these "oil and water" problems and finding a synthesis that made the extremes degenerate cases.

After much thought without the pressures of political agendas, it became clear that connection/connectionless was a function of a layer, not a service. It only involved the data transfer phase, not the allocation or enrollment phases, which were used depending on the density of traffic, connectionless when traffic was stochastic, connections when it was more deterministic.

As for addressing, we knew from John Shoch [1] that application names indicated what (location-independent) and mapped to network addresses indicated where (location-dependent, but route independent relative to the graph of the network), from Saltzer [2], which mapped that point-of-attachment addresses were route-dependent.

The insight was that routes were sequences of node addresses, which then required the node address to point-of-attachment address mapping of all nearest neighbors to select which path to the next hop. The realization that the mapping of application-name-to-node-address and the mapping of node-address-to-point-of-attachment address are the same mapping (architecturally, both are one hop away) implying a repeating structure clarified the naming and addressing problem. (Obviously there is much more that can be said about this but space does not permit here).

Every operating systems textbook [3,4] talks about the importance of separating mechanism and policy, for example, that the machinery (mechanism) for switching processes is the same, but what process to run next is policy; the machinery to manage memory is the same, what pages to replace is policy, et cetera. Given the close relation of OSs and networking and the similarities in protocols, it always seemed peculiar that separation of mechanism and policy was never mentioned in networking. Networking is much more a resource allocation problem, than a telecom problem.

To start, I applied the concept to the class of error and flow control protocols, sometimes referred to as the transport or data link layer. What I found was illuminating to say the least. These protocols naturally cleave between "tightly-bound" mechanisms (those that have to be with the data: ordering, fragmentation reassembly, delimiting, etc.) and "loosely bound" mechanisms (those that do not have to be with the data, the feedback mechanisms for flow and retransmission control), the traditional separation of control and data. (Data corruption detection/correction, applies to everything.) The two are very loosely linked through a state vector.

Data transfer writes the state vector, data transfer control reads the state vector and generates retransmission and flow control packets.

The only interaction between the two is when flow control stops data transfer from sending.

Similarly, layer management (enrollment, routing, resource allocation) are decoupled through a resource information base. Additionally, we see, that the problem yields just what one wants to see with a process consisting of functions of different duty cycles and computational complexity, that the functions required for the greatest performance are computationally simple and decoupled from those that are somewhat more complex but have a longer duty cycle are, in turn, decoupled from the most computationally complex and have the longest duty cycle.

We also utilized the result from OSI that distinguishing the abstract syntax from the encoding rules [5] allows protocol specifications to be invariant with respect to syntax. The syntax does not materially affect the operations on fields in the protocol. Most importantly, we utilized Richard Watson's seminal result [6] that the necessary and sufficient condition for synchronization is to enforce an upper bound on three times:

- Maximum packet lifetime (MPL),
- Maximum time to wait before sending an Ack, A; and
- Maximum time to exhaust retries, R.

Watson's result simplifies the protocol, makes it more robust and more secure. After datagrams, this 1980 result is undoubtedly the most important (and least known) result in networking. It is the case, this implies that the three-way handshake has nothing to do with establishing synchronization for a connection.

Combining all of this implied that there was only one error and flow control protocol with different syntaxes and policies. This points at a huge collapse in complexity, and at a repeating structure as did my deeper inquiry into naming and addressing.

Those indications did not explain what and why it was a repeating structure. A serendipitous question about TCP from a colleague and a poor answer from me pushed me to go back to the beginnings and show how the elements of networking arise from interprocess communication (IPC). That exercise not only explained why various aspects of protocol were needed, it also answered the question: Instead of five or seven layers, there was one layer, a distributed IPC facility (DIF), a distributed application that did IPC that repeats over different ranges of data rate, QoS, and scope. This is all worked out in my book, Patterns in Network Architecture: A Return to Fundamentals [7].

(I was very surprised when we first published this and IPC was seen as a radical new view of networking.

It was the common view in early days networking.

Dave Walden, who led the development of ARPANET switches (IMPs) wrote RFC 61 on IPC for a resource sharing network; Robert Metcalfe, inventor of Ethernet, said in passing in a paper "networking is interprocess communication"; and as late as 1982, RFC 871 takes IPC as axiomatic. It was common knowledge. What happened?)

The Implications of RINA

From these basic concepts, there were a number of new and surprising results:

1. All layers have the same functions. They operate over different ranges of bandwidth, QoS, and scope. There is one kind of layer that repeats (see Figure 1).
2. A layer is a distributed resource allocator.
3. This does not contradict Dykstra's characterization [8] that functions do not repeat in the layers. Functions do not repeat in the same scope. (In 1968, computers were so resource constrained that they only had one scope.)
4. All the user of the layer knows is the destination application name and its local handle for a flow, called a port-id, e.g., a file descriptor.
5. There is one error and flow control protocol with different syntaxes and policies (see Figure 2).
6. There is one application protocol operating on different object models.
7. Error and flow control protocols modify state internal to the protocol; application protocols modify state external to the protocol.
8. Port-Ids are the only identifier shared with the layer above and are never carried in protocol. connection-endpoint-ids are carried in protocol (required by Watson's result).
9. That the error and flow control protocols naturally cleave into two nearly independent tasks, one responsible for data transfer, and one for the more complex feedback functions.
10. The increase in commonality implies a huge collapse in complexity.
11. Repeating layers as distributed resource allocators for different ranges implies that most layers are private, or at most common to specific domains. This is definitely not a search for one size fits all, for one answer, but for answers that fit specific cases within the common architecture.
12. The IPC process naturally consists of three largely independent loci of processing with increasing cycle time and increasing complexity.
13. Addresses belong to layers, not to protocols. Protocols only carry addresses.
14. Routing is a resource allocation policy of layer management.
15. Repeating layers allows router table size to be bounded.
16. Multihoming, both for redundancy and load-leveling is inherent to the structure. It is free and requires nothing special.
17. Mobility is inherent in the model. There are no home agents, no foreign agents, no tunnels, no anchors, or no new protocols required. It just works and scales easily. Mobility is simply multihoming where points of attachment change a bit more frequently.
18. Multicast and anycast are found to be simpler and two forms of a more general construct: whatevercast, the name of a set with a rule for selecting members of the set.

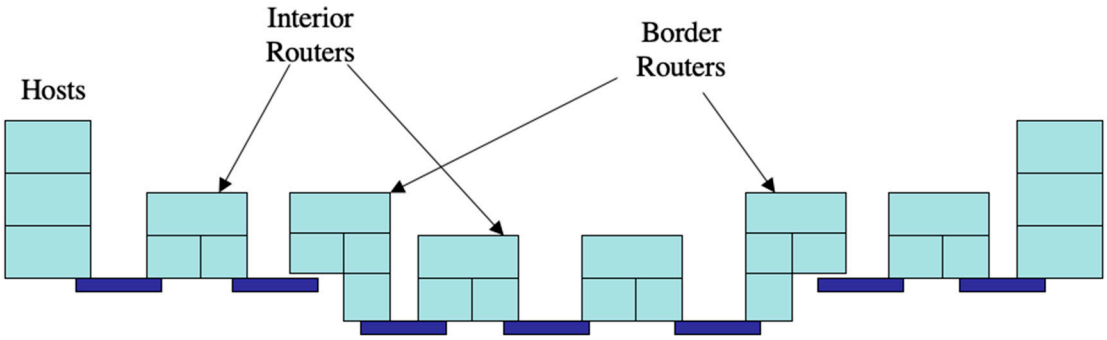

Figure 1. There are only three kinds of systems in Recursive InterNetwork Architecture (RINA). No middle boxes are necessary.

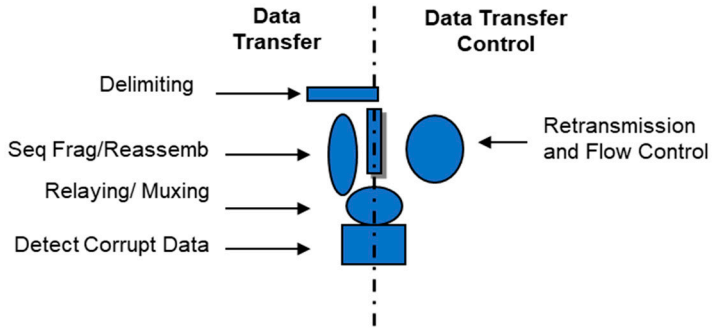

Figure 2. The structure of all error and flow control protocols from MAC and HDLC to IP, UDP, and TCP.

More Pieces Begin to Fall into Place

As we looked at aspects of networking not covered in the book, we found that the problem and the model pointed at simple answers to new topics as well as new capabilities we had not expected:

1. A global address space is unnecessary, yet all applications can be reached. The model implies the ability to find applications on other networks and dynamically create layers for them to communicate. This has major implications not only for security but beyond, that still remain to be explored.
2. The model told us where the security functions went and that the layer was a securable container. Not only does this greatly reduce the cost of security, but it makes firewalls unnecessary, and Watson's result avoids several security flaws found in TCP. Not only was no attention paid to security when developing the IPC model, but Watson's delta-t protocol was designed over a decade before the attacks on TCP were known, yet this design is inherently more secure. Suggesting that strong design may have more to do with security than security.
3. Watson's result implied the bounds of networking: If MPL could be bounded, it was networking; otherwise it was remote storage.

If a DIF is a distributed application that does IPC, then we had to answer what was a distributed application.

4. A distributed application facility (DAF) consists of two or more processes with tasks that coordinate scheduling, storage management, and IPC among its members, plus the tasks that do the work of the distributed application.
5. DAFs and DIFs are securable containers. How secure depends on policies. The structure is inherently more secure and an order of magnitude less costly.
6. Distributed programming becomes local programming.

There is still much about DAFs to explore.

7. Operating systems have a recursive structure as well, not just hacking virtualization. Every process has scheduling, storage management, and IPC for its tasks, which are processes. OS security reduces to bounds checking memory and IPC security. There is much to explore here.
8. Of course, a DIF is a specialization of a DAF.
9. The bottom process is the kernel of what corresponds to a traditional OS. All of the peripherals in today's OS have their own processor; hence the OS is a DAF.
10. Abstraction is invariance. What we had been doing was separating the invariant from what had to be able to vary.
11. We are working toward a unified model of application processes, distributed applications, operating systems, and networks.

Additionally, there is much more to explore in all of these areas.

Getting Computer Science Back to Being Science

Science is more than just using scientific method to prove things. It must also create a theory that fits the data and makes predictions. (Theory that describes is natural history.) Method without theory is craft. We found that the reason the model was proving to be so powerful was that we were extracting the invariants and then constructing the architecture to not break the invariances. This is why we never encounter "devils in the details," but instead we find angels.

The problem told us what to do. It provided the answers. This was both exciting and unnerving and led to our three mantras:

1. **Maximize invariance, minimize discontinuities**. In other words, look for invariances and avoid special cases, instead look for invariances that make the special cases degenerate cases. Make as much invariant, let what has to be variant be variant.
2. **Do what the problem says!** What does the problem tell us? Do not just look for something that works. The problem is smarter than we are.
3. **Prove the model wrong!** We cannot prove the model right, so we must try to prove it wrong. This will either uncover what we have missed or point to fundamental mistakes in our thinking. This is science. We must ensure it is honest.

So far our last mantra has not had much success. While the successes, the unexpected results, tend to confirm we are on the right track, we are still trying to find exceptions. If anything, areas we thought would turn up exceptions that were touted that their requirements required special solutions, turned out to be degenerate cases. What is more, the topics always tell us something new we did not know about the topic.

It makes for an exciting area of research, essentially a green field. While we do have specifications for the error and flow control protocol and the common distributed application protocol and other fundamentals enable the model to be explored. There is still much to be explored.

There is still much to learn about networking. Undoubtedly, there are capabilities in this model, like dynamically creating layers that are still unrecognized that will open up whole new areas.

Even with specific capabilities there are wide ranges of policies to explore. In particular, the support for applications is so much richer that we are sure it enables modes of distributed computing and networking applications far beyond the rudimentary client-server supported by the Internet.

As we have explored the IPC model with great success and found its predictions unexpected and exciting.

The New Paradigm Is not that New

I have also been looking back to understand both how the Internet got to where it is and how we lost sight of the importance of the IPC model. That too has been surprising. It appears to rest with the almost serendipitous good fortune to have had a unique experience in computing. I have already noted part of that unique experience with our exposure to the Burroughs systems.

We were also the 12th node on the Net.

It is very important that from the beginning the task of the ARPANET was not to merely replicate telecom with minicomputers, but to build a "resource sharing network." It was that that drove the distributed computing vision, not simply telecom.

Furthermore, our group was working closely with two of the founders of cybernetics (one, Wittgenstein's nephew) and the founder of modern neurophysiology. I cannot over emphasize the profound effect this had on critical thinking, of how to take a problem apart, and see what was going on, on how we analyzed problems, on how to construct a theory with elegant properties. (I will skip what we learned from being in the middle of student demonstrations against our project (against classified research on campus, which we agreed with and there was not any), or having our office nearly firebombed. (It did not go off. That shack would have gone up in seconds.) Or a conservative

legislature that balked when we published the location of marijuana fields in the surrounding area.) Ours was not your normal CS curriculum!

The early years of networking were just as exciting. Not only was creating a resource sharing network a very interesting problem, but it was also a true paradigm shift in thinking.

Networking was not telecom; but distributed computing. This was true for most in the ARPANET, where we have already pointed out, that the purpose of the network was resource sharing based on IPC. However, the ARPANET was not only the first packet switch network, but was built to be a production network to reduce the cost of research. As with any first attempt, it laid out the problem more clearly to see what was going on. There was considerable progress when these new ideas were turned into a true paradigm shift by Louis Pouzin and the CYCLADES project. (Paradigm has become vastly over-used. Here it is used in the true sense of the original proposed by Kuhn [9].)

Unlike the ARPANET, CYCLADES was a network to do research on networks. Having seen the ARPANET, CYCLADES could take a clean slate approach with help from the people who built the ARPANET. CYCLADES went back to fundamentals and quickly arrived at the 5 layer model, the concept of datagrams and end-to-end transport we all know, but also much more. They realized that this new paradigm was based on four premises:

1. it was a network of peers, hosts were participants in the network, not simply attached like telephones or terminals;
2. there was coordination in layers of different scope;
3. there was a tendency toward non-deterministic methods, not just in networks, but OSs as well with the transition to interrupt driven OSs; and
4. it was a distributed computing model.

They realized that in this new networking, it was not boxes and wires that were dominant as in telecom, in the ITU model; but processes and cooperating layers of different scope that were the dominant concepts.

This is where things go awry. As the mid-1970s neared, ARPA shut down the resource sharing emphasis in the ARPANET community, and even then not everyone picked up on the implications of resource sharing focus and saw it as an add-on to traditional networking. (Keep in mind that in the early 1970s, a datacom textbook spent 350 pages on the physical layer and 50 pages on everything else.) The shift away from resource sharing (and distributed computing) left the Internet focused on just the network itself, so many saw it as telecom (For example, I was shocked last March (2020) when Geoff Huston said "We originally constructed the Internet as a computer facsimile of the telephone network" Really? It was news to me!) Some were very much still in the traditional datacom (ITU) model. (Seeing Shannon as fundamental to networking is analogous to thinking the metallurgy of engines is fundamental to mechanical engineering, rather than Carnot.

For the Internet, it was the concrete aspects, datagrams, and end-to-end transport applied to traditional networking that was important.

Here datagrams and end-to-end transport were an end. Whereas, for CYCLADES and others it was the fundamental rethinking of networking as distributed computing, where datagram and end-to-end transport were just the beginning.

Note that by 1972, CYCLADES recognized the importance of congestion control, whereas it was not until the Internet experienced congestion collapse 14 years later that the Internet developers recognized it could happen.

CYCLADES encountered political difficulty and was shut down by the late 1970s before many of the new ideas they were pursuing came to fruition. Some of the ideas were kept alive in the OSI work but there they were under incredible pressure from the PTTs and the ITU model and the ideas became quite distorted and buried.

What I found when I looked back at the ideas was that RINA was picking up where the initial resource sharing ideas in the ARPANET and the clean-slate thinking of CYCLADES left off and carrying the new paradigm further.

RINA is on a straight line from the ARPANET to CYCLADES to RINA; showing that the early thinking that networking was interprocess communication is paying off handsomely.

Opening up exciting new areas for research that yield simpler more powerful results; while, the Internet without a model to guide it made a full-fledged embrace of the ITU model and still struggles with its inadequacy. As "enhancement" is heaped on "enhancement" (read "patch", "kludge") and new buzzwords with less meaning create more and more complexity, all are ominous indicators of the failure of every "new architecture" or "future Internet" effort of the last 20 years. It is clear that the major reason is that they are in the wrong paradigm.

A Selection of RINA Papers

With that introduction we can now turn our attention to this year's 7th RINA Workshop.

Every workshop there is a combination of new technical papers exploring aspects of the RINA architecture and an update on RINA developments in general as well as a discussion of our vision for the future.

This year was no exception. From the workshop, this special issue has selected four papers.

Let me give each a quick introduction:

"A P4-Enabled RINA Interior Router for Software-Defined Data Centers" by Carolina Fernández, Sergio Giménez, Eduard Grasa and Steve Bunch [10].

One of first things I did that led to RINA was the exercise of separating mechanism and policy.

Those results made it obvious that the complexity collapse was very reminiscent of a high performance operating system design we had done in the late 1970s. However, even more exciting was that it would be greatly enhanced by hardware. One just had to shift one's thinking about protocol implementation.

Here the EFCP implementation of the mechanisms are analogous to a universal Turing machine (UTM); a vector of policies for each layer, Turing machines (TM); and the packet is the tape. (For those whom automata theory was too long ago, a UTM reads a description of a TM and executes it, analogous to a computer running a program).

For each layer, the EFCP implementation would load the polices for that layer, process the header for that layer, move the "tape" down, load the policies of the next layer, process the header for that layer, and so on for both incoming and outgoing packets. It would be very fast. This paper reports the first chance to try out this strategy with an available hardware language, P4. While P4 is not sufficiently rich to directly implement the "packet as tape" approach, these initial results are exciting and indicate that this is a strategy to pursue vigorously. What is also interesting is how it isolates information flow within the IPC process of the model.

"Addressing Bandwidth-driven Flow Allocation in RINA" by Michal Koutensky, Vladimir Vesely, and Vincenzo Maffione [11].

One of the major aspects that RINA opens up is the exploration of the policies for resource allocation. In this paper, we have the beginnings of that exploration as policies for the flow allocator. The flow allocator has two main purposes: (1) to find the destination IPCP or application requested by the allocate service primitive and confirm that the requester has access and (2) to select the policies that will be necessary to meet the requested QoS. RINA recognizes that these systems are not that sensitive and that it is impractical to create a flow that meets a point in a multi-dimensional QoS-space, but can only provide QoS within a range of multi-dimensional QoS-space, referred to as a QoS-cube. Of course, two on-going goals of RINA research are to tighten those ranges where desirable and to move to QoS parameters that are more orthogonal or where the trade-offs are better understood. This paper concentrates on this last aspect.

Besides determining to what QoS-cube the flow should be allocated, even more important is determining whether the resources exist to meet the requirements of the requested allocation. This paper explores the environment for high density flow that ensures overprovisioning does not occur. To paraphrase its conclusion, this approach is best when the data-rate requirements are known in advance and availability is more important than predictability and latency. While it relies on a centralized update scheme, it can ensure that the resources are adequate to allocate the flow or, even better, reject the request if the resources are not available. This is a very interesting paper that, as a good paper should, inspires other aspects to explore.

"Towards a RINA-Based Architecture for Performance Management of Large-Scale Distributed Systems" by Peter Thompson and Neil Davies [12].

Speaking of exploring the QoS-space and the trade-off between QoS-parameters, this paper undertakes an ambitious attempt to make a significant contribution to this problem by applying ΔQ.

ΔQ explores the trade-off (as the paper calls it) of quality attenuation between delay (the continuous) and loss (the discrete). Rather than looking at specific policies, this paper explores creating a framework for performance management, in other words, the input and then analysis into what the policies within the layer should be and how the policies in the layer might be adjusted to respond to changing conditions.

The authors find RINA especially complementary to ΔQ because "RINA distinguishes key features that have been catastrophically conflated in the current TCP/IP architecture," and "nested scopes within which management of performance is feasible, and control loops can be short enough to address congestion in a timely fashion."

This is one of the best papers I have seen on management whose goals coincide with those of RINA in making quality of service more quantitative and less qualitative. There is much here to ponder and how it can be expanded to contract the ranges of QoS-cubes and to increase utilization without increasing overbooking solutions that spread the traffic of a flow over multiple paths.

The reader should spend solid time contemplating the implications of this paper.

"ARCFIRE: Experience with the Recursive InterNetwork Architecture" by Sander Vrijders, Dimitri Straessens, Didier Colle, Eduard Grasa, Miquel Tarzan, Sven van der Meer, Marco Capitani, Vincenzo Maffione, Diego Lopez, Lou Chitkushev, and John Day [13].

We wrap up this series of papers returning to the wide-range of experiments done under the ARCFIRE project to confirm the RINA model.

ARCFIRE is the latest of the RINA projects following on from IRATI and PRISTINE to explore the implications of the RINA model.

This project undertook to define and implement the detailed polices, syntaxes, security requirements, operating regions, et cetera, for a converged operator network (in other words, fully define one RINA network) that can be analyzed in detail.

In addition, ARCFIRE undertook experiments with scaling in RINA, resiliency within a single layer, renumbering RINA networks, providing QoS guarantees, distributed mobility management, simpler network management, and speed of node creation and validation.

These experiments serve to confirm what the model has predicted:

That the key to simpler network management is commonality that makes it possible to predict the interaction among layers and to make more meaningful measurements of the network; that the very simple fast network re-numbering capability is a consequence of the simpler, scalable solution to mobility management.

Conclusions

There are several things to point out here besides the excellent results of the experiments.

First, as noted above, the original intent of this effort has not been to propose solutions for the future Internet, but to find what the principles of networking are regardless of their implications. We are constantly trying to prove the principles wrong. RINA is the instantiation of those principles.

Interestingly enough, pursuing that work has gone a great distance to meeting the requirements of a future Internet. Second, all of the insights and innovations we have noted here have come from the principles, not from the implementation.

In fact the model has uncovered properties that have been obscured for decades. Furthermore, we have yet to have the implementation contradict the model. There are no so-called "devils-in-the-details."

This is by recognizing the invariances in the problem and constructing the model to not break the invariances. In fact, quite the opposite:

When we consider a new area, we find there are "angels in the details."

The model provides the solution without us having to propose anything. For the past several decades, the emphasis has been on implementation. It would seem that this has kept us too close to the trees to see the forest.

With that we hope this introduction to the papers for the special interest issue will be of great interest to the reader.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shoch, J. Internetwork Naming, Addressing, and Routing. In Proceedings of the IEEE Proceedings COMPCON, Washington, DC, USA, 5–8 September 1978; pp. 72–79.
2. Saltzer, J. On the Naming and Binding of Network Destinations. Available online: https://www.semanticscholar.org/paper/On-the-Naming-and-Binding-of-Network-Destinations-Saltzer/fa2cd80ba328b9b0c7225f3adcabfc9fea113275 (accessed on 2 October 2020).
3. Tanenbaum, A.; Bos, H. *Modern Operating Systems*; Pearson: London, UK, 2015.
4. Silberschatz, A.; Galvin, P.; Gagne, G. *Operating Systems*; Wiley: Hoboken, NY, USA, 2018.
5. Dubuisson, O. ASN.1 Commmunications between Heterogeneous Systems, Trans. by P. Fouquart. Available online: https://www.itu.int/en/ITU-T/asn1/Documents/chap7.pdf (accessed on 2 October 2020).
6. Watson, R. Timer-Based Mechanisms in Reliable Transport Protocol Connection Management. In *Computer Networks 5*; Prentice Hall: Upper Saddle River, NY, USA, 1981; pp. 47–56.
7. Day, J. *Patterns in Network Architecture: A Return to Fundamentals Prentice-Hall*; Pearson: London, UK, 2008.
8. Dijkstra, E. The Structure of THE Operating System Multiprogramming System. *Commun. ACM* **1968**, *11*, 341–346. [CrossRef]
9. Kuhn, T. *The Structure of Scientific Revolutions*; Univ. of Chicago Press: Chicago, IL, USA, 1962.
10. Fernández, C.; Giménez, S.; Grasa, E.; Bunch, S. A P4-Enabled RINA Interior Router for Software-Defined Data Centers. *Computers* **2020**, *9*, 70. [CrossRef]
11. Koutenský, M.; Veselý, V.; Maffione, V. Addressing Bandwidth-Driven Flow Allocationin RINA. *Computers* **2020**, *9*, 63. [CrossRef]
12. Thompson, P.; Davies, N. Towards a RINA-Based Architecture for Performance Management of Large-Scale Distributed Systems. *Computers* **2020**, *9*, 53. [CrossRef]
13. Vrijders, S.; Staessens, D.; Colle, D.; Grasa, E.; Tarzan, M.; van der Meer, S.; Capitani, M.; Maffione, V.; Lopez, D.; Chitkushev, L.; et al. ARCFIRE: Experimentation with the Recursive InterNetwork Architecture. *Computers* **2020**, *9*, 59. [CrossRef]