Requirements for Deploying IP and ICN Network Stacks on a Common Physical Infrastructure

Renan Krishna[1(✉)] and Roger Baig Vinas[2]

[1] Interdigital Europe, London, UK
renan.krishna@interdigital.com
[2] Universitat Politècnica de Catalunya, Barcelona, Catalunya, Spain
rbaig@ac.upc.edu
https://www.interdigital.com/, https://dsg.ac.upc.edu/

Abstract. Deploying alternative networks such as Information Centric Network (ICN) in a production/commercial network with real users is challenging due to the experimental nature of these novel proposals. To meet these challenges, we adopted an ICN-IP dual stack approach. However, this was at the cost of introducing unpredictable emergent behavior. This behavior can be dealt with by a set of requirements presented in this paper. These requirements specify a constraining discipline on the deployment and operational processes for the dual stack making such processes tractable. The requirements are extracted from our experience of a lab deployment followed by a field deployment with real users as part of the three-year long EU-funded architectuRe for an Internet For Every-body (RIFE) project. We summarize them in a form that can be used by other practitioners in their own ICN/alternative network stack deployments and by tool developers for such deployments. These presented requirements compliment the current discussions within the Centric Networking Research Group (ICNRG) of the Internet Research Task Force (IRTF).

Keywords: Information-Centric Networking · Deployment · Testbed · Requirements · Tools

1 Introduction

It has been recently articulated that there are three distinct levels for the process of designing an Inter-network [23]. At the highest level are the core principles as well as basic design decisions of the architecture. Next level is the design

This work received financial support through the European Union H2020 Research and Innovation Programme, Grant Agreement No 644663, by the Spanish State Research Agency (AEI) under contracts PCI2019-111850-2 and PCI2019-111851-2 grants, and by the Spanish Ministry of Science and Innovation under contract PID2019-106774RB-C21.

Y. Weng et al. (Eds.): TridentCom 2020, LNICST 380, pp. 27–46, 2021.
https://doi.org/10.1007/978-3-030-77428-8_3

of mechanisms that convert the architecture into an implementation. Finally, at the lowest level are the set of decisions about deployment that result in an operational Inter-network.

To design an Internetwork, the Information Centric Network (ICN) architecture has emerged as one of the many new paradigms. There have been several proposals for an Information Centric Network architecture over the past few years [1–6,8,10–12,19–21]. A recent Internet-Draft [22] by the Information-Centric Networking Research Group (ICNRG) of the Internet Research Task Force (IRTF) has discussed several deployment considerations for a live deployment of the ICN architecture thereby contributing to the discussions at the lowest level of the abstraction hierarchy of [23].

In this paper, we elaborate on the lessons learnt in the field trial of the project called architectuRe for an Internet For Everybody (RIFE) [9] mentioned in that draft [22] and which is at the lowest level of the aforementioned hierarchy of [23]. In particular, we discuss RIFE's dual IP-ICN deployment which allowed us to alternate between either of them as an underlay [22] while providing Internet services to real users in the field. This paper thus compliments the discussion in that Internet-Draft.

The challenge in the field trial was to deal with problems that emerge within a tight time schedule when setting up two different OSI network layers on the same physical network. Such a setup faces the problem of having to run ICN software over an infrastructure that is optimized for the IP world. Moreover, the two network layers sharing common machines interact with each other in a manner that results in an unpredictable emergent behavior. To deal with such problems, an appropriate workflow needs to be developed to coordinate the activities of the ICN deployer and the network operator. Additionally, an easy way to switch traffic from ICN back to IP is required to deal with bugs in the ICN code and ensure that the users can still access an operational network.

We present a set of general requirements that can be followed by practitioners seeking to deploy an alternative network architecture on an operational infrastructure with real users. In addition, these requirements can guide the development of tools for such deployments. In particular, our contributions are:

- We describe a way to impose a common vocabulary through code for the naming, the order of initialization and error specification of hardware/software components that should be used both by the network operator and the ICN deployer. We discuss why this is important for dealing with unpredictable emergent behavior resulting from the way the two network stacks interact with each other on common machines (Sect. 4).
- We describe a way to achieve a switch between the IP and ICN networks such that the user's experience is not disrupted and at the same time is easy and convenient for the network operator. We discuss why this is important in field-trial deployments (Sect. 4).
- We have produced a set of requirements that provide constraining guidelines to instantiate the configuration of the IP and ICN software on the production networks whilst respecting the precedence constraints on the compo-

nents of the network. These requirements are general enough to be useful for Inter-network layers other than ICN and for development of tools for such deployments (Sect. 5).

The rest of this paper is structured as follows: In Sect. 2, we give an overview of the ICN flavor that we deployed, Sect. 3 explains in details the problems that we faced when deploying our ICN implementation in the operator's network, Sect. 4 describes how we solved those problems, Sect. 5 summarizes the lessons we learnt from our deployment into a set of requirements, Sect. 6 discusses the related work, and finally in Sect. 7 we present our conclusions.

2 A Brief Overview of ICN

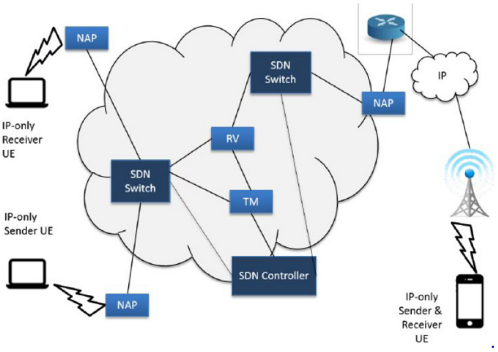

Fig. 1. ICN architecture.

The system architecture used by the RIFE project (Fig. 1) aims to replace the network of an individual network operator, so as to improve the IP-based services that it offers to its customers [3]. RIFE does not require any changes to the existing IP User Equipment (UE) connecting to the operator's network, or to the IP routers/gateways of other interconnected operators. This is achieved by combining an ICN, which forms the core of the RIFE network, with a set of Network Attachment Points (NAPs), which reside at the periphery of the RIFE network, serving as gateways between the IP and ICN worlds. The baseline architecture was derived from the Framework Programme 7 (FP7) Publish Subscribe Internet Technology (PURSUIT) ICN architecture, in which the end-user nodes can publish and subscribe to named information items. This publish/subscribe architecture implemented in a software called Blackadder is facilitated by three core functions: a Rendezvous (RV) function that matches publisher and subscriber nodes; a Topology Manager (TM) function that calculates paths between the various nodes and encodes them into Forwarding Identifiers (FIDs); and, a Forwarding Node function that allows data items to be forwarded in the network

based on the FIDs. We now describe the NAP, Forwarder (FWD), RV and TM functions of the ICN architecture:

- Network Attachment Point: In order to preserve the IP interfaces towards UEs and other operators, RIFE uses a gateway approach. The NAPs, e.g. the access gateways of customers to the network, or the gateways of the network to peering networks or operator server resources, handle all the offered protocols at the IP interface, either directly as IP packets, or, if possible, as transport or application layer messages, for example, as HTTP messages. Thus, this NAP implementation allows us to use an IP-over- ICN abstraction in the Field Trial. See [7] for more details on how this is done. The NAP function can also be used as a GateWay (GW) to the Internet.
- Forwarder (FWD): A core function is that of delivering the information from the source(s) to the sink(s), i.e., the publisher(s) to the subscriber(s). Software Defined Networking (SDN) switches are used for the Forwarding (FWD) functionality.
- Rendezvous and Topology Manager: The RV maintains IP and Fully Qualified Domain Name (FQDN) subscription information that is then used to facilitate its matching function. When the TM receives a topology formation notification from the RV, it uses a shortest path calculation between a publisher and subscriber. This path is then encoded by OR-ing the FIDs (in the form of bit strings) and the result is included in each packet.

3 Dual Stack Deployment Problems

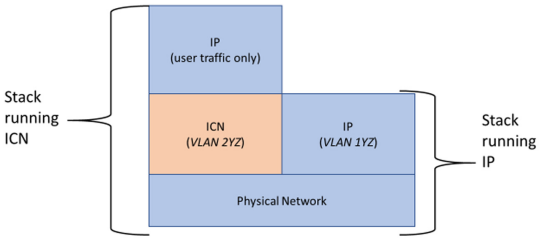

Fig. 2. The ICN and IP stacks.

Consider Fig. 2 where two stacks that we ran in parallel over the common physical network are shown. The ICN stack (left) was used to carry the users' traffic which as discussed in the previous section was only IP traffic although the same setting allows for other ICN traffic such as HTTP and Constrained Application Protocol (CoaP). Additionally, ICN signaling traffic was also carried over this stack.

The IP stack was used to carry the management traffic of the network operator and, optionally, the users' traffic. This user traffic could be switched by the

operator to the IP from ICN stack and vice-versa. The IP stack was also used as a backup to access the ICN devices.

We now identify the following six problems that needed to be addressed to enable a successful deployment:

Problem 1. The deployment network architecture had to be designed to facilitate an error-free and easy to debug running of the ICN software. The deployment that followed this design had to include a suitable topology and a suitable configuration of the network infrastructure.

Problem 2. The ICN and the management IP networks needed to be isolated from each other using techniques that facilitated an error-free deployment. This isolation is required not only to prevent the two networks from interfering in each other's functioning but also to have an ability to identify the network from which a problem originates. Moreover, such an isolation enables a systematic allocation of resources such as ports, interfaces etc. and simplifies the initialization scripts.

Problem 3. We needed a way to deal with unpredictable emergent behavior during the initialization of the ICN network. This behavior emerges even when the ICN and the IP management networks are isolated (using VLAN) from each other because the two isolated network share physical resources and the VLAN scheme provides no means to fairly distribute bandwidth between them.

The emergent behavior is a consequence of the way the configuration of the two stacks interact with each other. Often, these behaviors are hard to predict prior to a deployment. Some examples of such emergent behavior that we had to face in our deployment include:

a) Loops involving broadcast/multicast MAC frames circulating in the various VLANs led to a complete network meltdown with SSH access becoming impossible. As a result, we were unable to even run our scripts to deploy the ICN network.
b) Low ICN throughput of 0.5 Mbps–1.5 Mbps rather than the expected value of around 20 Mbps with TCP traffic was observed.

Thus, we needed a way to deal with this non- deterministic behavior of the interaction between the operator's network providing IP-based management services and the ICN network. This behavior can emerge despite all the components of the ICN and IP-based networks individually showing deterministic behavior.

Besides the deployment challenges discussed above, following challenges were faced while running an ICN in the real world:

Problem 4. We needed to develop a workflow to coordinate between the network operator and the ICN deployer to deal with debugging problems that arose out of the unpredictable emergent behavior of running ICN. Often, the process to deal with this behavior can be tedious, time- consuming and error prone. In

addition, the deployed equipment was a 100km drive away and a misdiagnosis of the causes for the errant behavior could result in a waste of time, effort and money.

Problem 5. The ICN network needed to be monitored to ensure that no component had crashed. To respond to a crash, coordination was required between the remote deployer (in London) of the ICN network and the local operator (in Barcelona) of the physical infrastructure. Similarly, the ICN network needed to be monitored for performance. To respond to performance issues, again coordination as described above was required.

Problem 6. We needed a way to ensure that the real-world users could have access to the internet even when the ICN network was stopped for debugging purposes.

4 Deployment Solution

We now discuss the solutions to the problems described in the previous section.

Solution to Problem 1. The field-trial was based on Wireless fielity (WiFi) technologies (for their ease of deployment) and deployed in the Guifi.net community network [24]. In this community network, the network infrastructure is heterogeneous (the project is defined to be technologically neutral) with WiFi and optical fiber being the most common technologies that have been used [25]. To keep our deployment simple, the ICN software that we used only supported static network topologies and at the physical network level we just used WiFi in infrastructure mode. The alternative to WiFi links were optical fiber links, but this option was discarded not only due to time and budget constraints, but also because challenging topologies are more difficult to achieve (fiber deployments are commonly much less meshed than WiFi deployments). The Ad-hoc WiFi mode was discarded due to the high variability of the link qualities and the high connectivity degree between nodes.

In infrastructure mode there are two different types of nodes (a node is a geographical location with network equipment): the Super Nodes (SNs) and the End-nodes (ENs). The nodes are connected to each other through links. Topologically speaking, the SNs are the nodes with more than one link to other nodes (i.e. they extend the network) while the ENs are just linked to one SN (i.e. they are the leaves in a graph). The links between SNs are point-to- point links, usually built with highly directive antennae, forming the backbone network. The links between ENs and their SN are usually point-to-multipoint links, built with sector antennae, also referred as Access Points (APs). The resulting topology is the same as that of the most common wired deployments.

Figure 3 shows a generic SN. In addition to the directive and sector antennae, its main components include: a core router, an (unmanaged) switch and a number

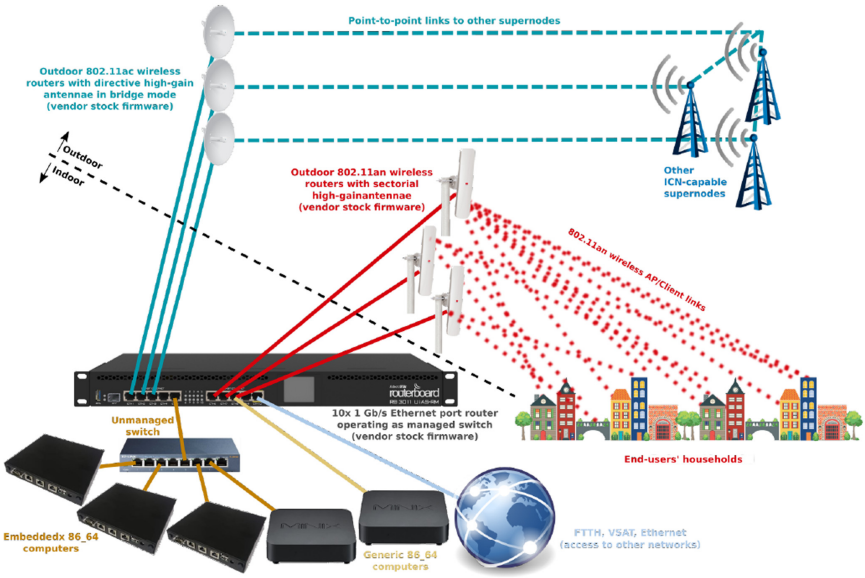

Fig. 3. Generic supernode architecture.

Table 1. Computing devices.

Name	Function
FWD	ICN FWD
NAP	ICN NAP
RVTM	ICN RVTM
SRV	Apache server
AUX	General purpose Linux box. Has one IP of each AP subnetwork
CLE	Linux box with LXC. Replaces an AP. Emulates the whole AP-end-user nodes construction
ALT	General purpose Linux box. Runs DHCP client in an interface. Aimed at reaching devices hidden behind a NAT

of ICN Computing Devices (CDs) described in Table 1, and so on. It also shows the interconnection links for transit or peering.

We deployed two experimental facilities- one was a production network with 40 end-users in several villages, and the other in a laboratory as a partial reproduction of the production network. In our implementations the core routers were connected to up to three directive antennae and up to three APs. In the laboratory's deployment, some APs and the attached Customer Premises Equipment (CPEs) were replaced by Client emulators (CLEs), Linux boxes which emulated sets of APs and the associated ENs.

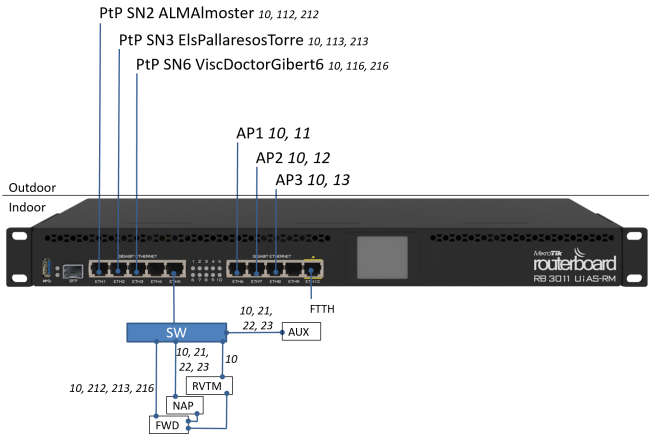

Fig. 4. SN1 router. Ports' assignment.

In our deployments the ICN functions NAP and FWD were each implemented by an independent CD to enable an easy isolation of problems such as software crashes, performance issues, misconfiguration of software/hardware etc. Figure 4 shows the ICN CDs wiring. The RV and TM functions are performed by another CD called RVTM. Note that there is one RVTM per deployment. Its first Ethernet port is attached to the switch and the second to the third port of the FWD. There is one NAP for each SN with ENs. Each SN has a forwarding function which is performed by the CD called FWD. The first Ethernet port of the NAP and the FWD is connected to the switch and the second port connects them to each other.

Solution to Problem 2. We isolated the two networks; the ICN network and the IP-based management network into separate VLANs.

We now discuss the VLAN naming conventions that we followed, how we configured the routers in the super nodes to support the VLANs, and how we organized the IP addresses to support our VLANs.

The VLANs were named so that they were semantically meaningful. For our deployments, this semantic structure was used to identify the network (IP, ICN) and the nodes and interfaces involved as described below:

- Each point-to-point link had two VLANs XYZ; X = 1 for the IP network, X = 2 for the ICN network; Y was the lower SN number; Z the higher SN number. The SNs were numbered from 1 to 6. For example, a point-to-point link between SN1 and SN6 will have two VLANs: VLAN 116 for the IP network and VLAN 216 for the ICN network.
- The FWD had the corresponding 2YZ VLANs (first Network Interface Controller (NIC)). For example, the FWD on SN1's point-to-point link to SN6 will have its first NIC connected to VLAN 216.

- The VLANs of the APs were 1A; A = 1 for AP1, A = 2 for AP2, and A = 3 for AP3.The prefix 1 here denotes that the VLAN carries IP traffic. For example the three VLANs on any SN will be 11 for AP1, 12 for AP2 and 13 for AP3.
- The NAPs and AUXs had VLAN ids 21, 22 and 23.The prefix 2 here denotes that the VLAN carries ICN traffic.
- The VLAN10 was reserved for management and had a scope that was limited to within an SN.

Thus, since a directive antenna's wired NIC was directly connected to the router, it had a VLAN10 and VLAN2XY (Prefix 2 denoting that the VLAN carries ICN traffic and X is the lower SN number; Y the higher SN number). On the other hand, since the directive antenna's wireless NIC was pointed towards a similar NIC on another super node, it had interfaces corresponding to 1XY (Prefix 1 denoting that the VLAN carries IP traffic and X is the lower SN number; Y the higher SN number)) and 2XY (Prefix 2 denoting that the VLAN carries ICN traffic and X is the lower SN number; Y the higher SN number). In the access point's sectorial antennas, only the wired NIC had interfaces corresponding to the VLANs 10 and 1A (A=1 for AP1, A=2 for AP2, and A=3 for AP3) for similar reasons as the directive antenna. Finally, in the router, interfaces corresponding to VLAN10 were defined on all the ports

We organized the bridges within a router on an SN as follows (Fig. 4 and Fig. 5):

Routers

- A bridge br1YZ for each VLAN1YZ. Each bridge only has one port and VLAN1YZ connected to the corresponding directive antenna. For example, in Fig. 5 br112 has port P1 and VLAN112 connected to a directive antenna.
- A bridge br2YZ for each VLAN2YZ. Each bridge has two ports and VLAN2YZ connected to the corresponding directive antenna and the switch of the ICN fabric network. For example, in Fig. 5 br212 has two ports P1 and P5 and VLAN212. Port P1's VLAN212 is connected to the directive antenna. Port P5's VLAN212 is connected to a switch in the ICN fabric network.
- A bridge br1A for each VLAN1A. Each bridge may only have one port and VLAN1A connected to the corresponding AP. For example, in Fig. 5, br11 has port P6 and VLAN11 connected to the AP provided by the sectorial antenna.
- A bridge br2A for each VLAN2A. Each bridge has at least one port and VLAN2A connected to the switch in the ICN fabric network. For example, in Fig. 5, br21 has port P5 and VLAN21 connected to the switch in the ICN fabric network.
- A bridge br10 for VLAN10 connected to all ports and VLAN10. For example, in Fig. 5, br10 is connected to all ports namely P1, P5, and P6 as well as to VLAN10.

FWDs

- An OpenVSwitch (OVS) based software switch runs on each forwarder. Each such switch has exactly one bridge called br1.

CLEs

– A bridge called br-cli for the LXC. It has the corresponding VLAN1A and all the LXCs' interfaces.

We had two separate deployments as discussed earlier. In the lab deployment, we had three SNs whereas in the production network in the villages, we had six SNs. We organized the IP assignments as follows:

– A 10.X/16 IPv4 block per deployment.
– A 10.X.Y/24 IPv4 block per SN.
– The 10.X.Y.0/26 IPv4 block per SN for management IPs. All devices except end-user equipment have management IP.
– The 10.X.Y.64/26 IPv4 block per SN for AP1 clients. The 10.X.Y.128/26 IPv4 block per SN for AP2 clients. The 10.X.Y.192/26 IPv4 block per SN for AP3 clients.
– In the CDs, the management IPs have a dedicated routing table.

In our design the backbone is IP-ICN dual stack but each AP can only be associated to one of the two network underlays at a time (either IP or ICN). The association to one of the networks is implemented at the router level by associating the corresponding VLAN1A to either the br1A bridge or to the br2A bridge.

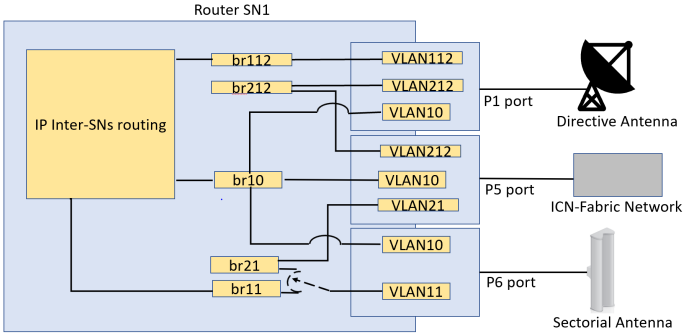

Fig. 5. Arrangement to switch between ICN and IP on SN1.

Consider Fig. 5 where we show the organization of VLAN tags, bridges and physical ports in the router on Super Node 1. The user traffic is sent and received on the physical port P6 via the sectorial antenna and is passed on to the sub-interface corresponding to VLAN11. This is the traffic that can then be switched either to the bridge br11 resulting in the user network connecting to the IP network or to bridge br21 resulting in the user network being connected to the ICN network. In this way, we were able to provide customers with highly available Internet services even when the ICN network was being debugged for crashes or performance issues by associating UE to the ICN or the IP network at the AP level.

Solution to Problem 3. We now discuss the causes of the emergent behavior described in the previous section and outline the general techniques and processes that practitioners can adopt to run ICN and IP-based networks over the same infrastructure.

One of the causes of the emergent behavior described in the previous section is that many of the default settings in the operator's network infrastructure are not the correct settings for the ICN software to operate upon. These settings then interact unpredictably with the deployed ICN network. Moreover, it was tedious and time consuming to debug the emergent behaviors during deployment because of an incomplete capturing and enforcement of the precedence relations between the configuration of the network operator and those of the ICN deployer. We now explain the strategies that we adopted to deal with these issues.

Issue 1. Emergent behavior caused by the default settings of software that were only appropriate for an IP and not an ICN network:

As described earlier, on each Supernode, we set up an OVS switch in software running on a Linux box with a single bridge having multiple ports. Each of the multiple ports correspond to a different VLAN. This bridge by default, constructs in its table a so-called 'normal rule'. Such a rule has an action that forwards all traffic from one port on the bridge to all the other ports on the bridge. This is the default configuration of an OVS switch for an IP network.

However, this created loops involving broadcast/multicast MAC frames circulating in the various VLANs leading to a complete network meltdown with even SSH access becoming impossible. Thus, we were unable to even run our script to deploy the ICN network.

Solution to Issue 1. To solve this issue, a single line of code was needed to keep the table of OVS initially empty.

The result of the line of code is a new default behavior of the bridge where it does not forward any packet arriving at a port to any other port. This caused the looping issue to disappear and we could run the ICN deployment tool. This tool wrote new rules onto the OVS-switch's bridge and we were able to deploy our ICN network.

As multiple virtual switches become common in Network Function Virtualization and network slicing based scenarios with complex topologies, looping issues will become all pervasive. Thus, setting up a virtual switch with appropriate initialization strategies until rules are written to it by an SDN controller has become important in a real deployment.

In another default Linux settings, the kernel offloads TCP segmentation to the NIC for better IP network performance.

However, this resulted in our NAP Blackadder application getting larger packets than it expected because the NIC was not respecting the MTU size set in the NAP code. The NAP would drop these larger than expected packets resulting in a packet loss and a subsequent re- transmission of the packet from the sender. The overall effect was that the throughput between a sender and a receiver was observed to be between 0.5 Mbps and 1.5 Mbps on our ICN network.

A simple solution that we adapted was to switch off the TCP segmentation offload configuration of the Linux kernel on which the NAP was running. This caused the throughput between the same sender and receiver to jump to around 18 Mbps from the initial 0.5 Mbps to 1.5 Mbps.

For increased network performance in operational deployments, the operators utilize many optimizations. Therefore, any real-life deployment that seeks to retain the HTTP/IP configuration of end-user devices while at the same time replacing the standard TCP/Internet Protocols (IPs) network stack in the internal network must carefully consider the impact of such optimizations on the operational parameters.

Issue 2: Difficulties in debugging emergent behavior during deployment caused by an incomplete capture and enforcement of the precedence relations between the configuration of the network operator and that of the ICN deployer:

The root of this problem lies in the fact that even when the ICN and IP networks are isolated, these two network stacks must be setup on the same machines resulting in the afore-mentioned constraints and making it tedious and error prone to track down the bugs causing the emergent behavior.

Solution to Issue 2. We now discuss our initial workflow and how we modified it to make it easier for us to deal with the emergent behavior and point out the sources of precedence constraints that we had to deal with for a successful deployment.

Initially, the entire network infrastructure including setting up of all the hardware and the software (excluding ICN software) was deemed to be the concern of the network operator. Once the infrastructure was setup, the ICN deployer would deploy the software stack for ICN on the network infrastructure.

However, this caused difficulties in debugging emergent behavior. It was often unclear if an emergent behavior (say network meltdown) was caused by a configuration error in the network operator's deployment or in the configuration of the ICN software or in an interaction between the two.

So, instead of following a workflow where the network operator completes their configuration and then hands over the deployment of ICN software to the ICN deployer, we changed it. In the new workflow, we decided to interleave the steps that the network operator and the ICN deployer were implementing for the deployment in a single script. This enabled us to explicitly represent the precedence relations between the network operator's and the ICN deployer's configurations in that script. A side effect was that both the network operator and the ICN deployer became aware of the other's precedence constraints making it easier to debug emergent behavior.

These precedence constraints arise out of the dependency of the ICN network elements on the configuration and initialization of machines on which they are hosted. These dependencies can be a result of the requirement of an ICN component for data so that it can set up its execution environment within the operating system running on a machine and the network to which that machine is connected. Some examples include:

a) Topology Manager (TM) needs to know what paths are available for the ICN traffic. These link layer paths are then used by the TM to calculate the topology of the network. The TM can only be set through the IP management network. Thus, the VLANs and IP network interfaces must be set up before the TM can function properly.
b) Rendezvous (RV) and TM can only run if the ICN component called Blackadder (the ICN pub/sub forwarder) is already running. Blackadder in turn can only run if the operating system has been installed, configured and running in the operator's machine. This installation of Blackadder is done via the IP management network. These constraints are imposed by the architecture specifying that RV and TM are at a higher layer compared to Blackadder (See Sect. 2).
c) RV can match subscriptions to publications only if the operator network is up and running.

Another source of a precedence constraint is the requirement that a component must wait for another component to finish its initialization before starting its own configuration. This initialization often takes time that cannot be precisely predicted. Some examples from our deployment include:

a) The scripts initializing the ICN Blackadder component need to wait for an imprecise duration of time whilst the operating system and the network is booted up.
b) The RV, TM initialization by the scripts must wait for the ICN Blackadder bootup to finish. Knowing that RV, TM need to boot up after ICN Blackadder is not enough. Additionally, the scripts initializing RV, TM must wait for an imprecise duration of time as ICN Blackadder is booted up.

We now discuss ways to enforce the precedence constraints to avoid unpredictable emergent behavior during the initialization of the ICN network. Note that once the ICN network is initialized, the responsibility to ensure that runtime precedence constraints within the ICN network are respected lies with the various ICN protocols and the design of its architecture.

When the initialization of a component depends on data from the operating system or a network element that themselves need to be initialized, we need to capture these dependencies in a precedence graph. This precedence graph can be an informal diagram or expressed in code. The precedence graph can then be used to extract a schedule informally, or by running an algorithm such as topological sort on the precedence graph. For complex deployments, more sophisticated algorithms such as the critical path method can be considered to generate the schedule.

We generated such a schedule informally by carefully walking through the order in which the ICN and IP components needed to be initialized. For example, our schedule encapsulated in scripts made sure that the network interfaces were set up in all machines for both the ICN and IP networks before we deployed the configuration and topology of the ICN network. The schedule took care of the precedence constraint requirement of an ICN component for getting data so that

it can set up its execution environment within the operating system running on a machine and the network to which that machine is connected.

We also made sure that the scripts encoding this schedule would wait for a configuration to finish before initiating the next configuration. This was done via remote configuration calls to the appropriate machines that would sleep long enough so that the configuration on the remote machine was completed. This took care of the precedence constraint requirement that a component must wait an unpredictable duration of time for another component to finish its initialization before starting its own configuration.

Thus, we were able to reduce the problem of dealing with the unpredictable emergent behavior during initialization due to the interaction between ICN software and the IP software to one of dealing with generating initialization schedule from precedence constraints using standard techniques and algorithms.

Solution to Problem 4, 5, 6. To solve the requirement for an appropriate workflow between the ICN deployer and the network operator as discussed in Sect. 3 above, we centered the workflow around code that encapsulated the deployment.

This code became the interface for interaction between the ICN deployer and the network operator. All the necessary configurations of various software components, network interfaces, SDN switches being used for both ICN and IP were initialized by this code. Thus, the code became both the documentation of the configuration as well as of the schedule of re-deployment in case of crashes, performance issues etc. As a result, both the ICN deployer and the network operator could go through the code together and agree on the necessary changes needed to address a problem. Note that once the infrastructure is captured by code, it is easy to modularize the code with a module corresponding to one component of deployment say the NAP. Capturing the various component versions in the code made the re- deployment repeatable and deterministic.

Moreover, using the code in this way to represent the IP and ICN infrastructure made it possible to use standard software development processes to coordinate between the ICN deployer and the network operator. For example, in order to update a software component, we would change the version of the deployment code thereby easily maintaining a record of working deployments in the previous versions.

Additionally, we followed good software engineering practices for naming variables in code. The VLANs, network interfaces, antenna etc. were named in a systematic and consistent manner as already discussed. These names were then easily used in discussions between the ICN deployer and the network operator without any ambiguity as part of the workflow.

Using the code as a representation of infrastructure also enabled us to automate the process of recovery in case of software/hardware crashes. The code that was being used to monitor various components would generate the necessary log files and would automatically run the deployment scripts whenever a component crashed.

Another advantage of making code as the interface between the ICN deployer and the network operator was that we could make changes in small increments (For example adding or removing computing devices) and could easily undo things if software components crashed or performed poorly. Similarly, the network operator could easily switch from ICN to IP secure in the knowledge that ICN configuration was being reworked in the code and would be easily re-deployed. This also enabled us to provide a high degree of availability of the internet service to the customers.

We also partially replicated the field trial by setting up three Super Nodes in our lab in Barcelona. All the software components, computing devices, routers, switches used in this lab setup were identical to the ones that we used in the trial. The code that encapsulated the deployment was always first tested in the lab whenever we wanted to change the configuration of the deployment software/hardware. This ensured that any bugs were caught, and our customers were assured of a high availability of Internet services.

The code described above was run from a centralized server that was accessible to both the ICN deployer and the network operator. This ensured that both the deployer and operator could redeploy the entire system on their own since all the configuration knowledge was already captured in the code.

Thus, by representing the deployment infrastructure as code, accessible to both the deployer and the operator and properly versioned, we were able to use the code as the interface between the deployer and operator. The practices we have described above are a subset of the practices called *Infrastructure as Code* being advocated by the DevOPs community for cloud deployments [26]. Each of the practice that we have described above is on its own already known and well established. However, when these practices are combined, we get emergent behavior that is useful in deployments such as ours. In particular, it enabled us to ensure a smooth workflow that minimized downtime for the customers.

5 Requirements

We now summarize the lessons learned and propose the following requirements as constraining guidelines that could be used by other researchers looking to deploy their ICN/alternative network architecture in a real-world operational network based on the dual-stack approach and to develop tools for such deployments. These requirements represent constraints that when followed lead to tractable deployments of reduced complexity. This complexity of deployment is caused by the problems that we discussed in Sect. 3 of this paper. Besides, these requirements are general enough to be useful for deployment of network layers other than ICN using a similar dual-stack approach.

Requirement 1. *Translate novelties of ICN/alternative architecture into familiar configurations for the network operator.* A deployment network involves many network technicians who might be unfamiliar with experimental architectures and may be skeptical about using them.

One way to deal with this problem is running training workshops for them to make them familiar with such an architecture.

However, often the time allocated to run the ICN/alternative software is limited due to commercial/usability considerations- in our case we were given a 3-month window to set up the network and deploy the ICN software. Moreover, the network technicians are often required to be available on demand for the day-to-day running of their operational network. This often makes it infeasible in terms of time to run training workshops for them to be familiarized with a new technique.

Therefore, we suggest that the instructions given to the technicians for deployment must translate the novelties of an architecture into configuration techniques that they already know whenever possible. The tools developed should have an interface that supports those configuration techniques. In our case this entailed providing the network operator's technicians with semantically meaningful scheme of VLAN tags and allowing them to easily switch between VLANs. This switching of VLANs by the technicians translated into switching between IP and ICN at the deployer's level.

Requirement 2. *Use precedence relations as a centerpiece of a strategy to deal with emergent behavior.* In a deployment network, prolonged network down-times are unacceptable to the customers. However, since ICN/alternative networks are not as highly engineered as the IP network, unpredictable emergent behavior that degrades the customer experience is inevitable.

Debugging such emergent behavior is tedious and time consuming. The technicians on the field are not familiar with the ICN/alternative network technology making it hard for them to debug emergent behavior. As a result, minimizing the time required to debug emergent behavior as and when it is encountered is critical in an operational network as opposed to a research network.

Therefore, debugging emergent properties in a systematic way using precedence relations should become the centerpiece of the deployment workflow. The precedence relations between components should be treated as a first-class entity for deployment tools to deal with the unpredictable emergent behavior. This means that the deployment tools should be centered around generating an appropriate initialization schedule from those precedence relations using techniques such as topological sort, critical path etc.

Requirement 3. *Represent Infrastructure as Code.* The entire deployment and its schedule as generated above should be captured in code that is supported by tools. This code then becomes the interface between the ICN/alternative network deployer and the network operator.

In particular, attention should be paid to the naming of various hardware/software components to avoid ambiguity when dealing with system failures. The component's versions and their updates must only be done via this code (rather than logging into individual machines) in order to document them and to be able to easily detect changes that broke the deployment. This code

itself should be properly versioned to capture all the working deployments and simplify a roll-back to previous configurations. This code should be tested on a replica of the actual deployment to detect any problems early. The code itself should be run from a machine that is accessible to both the deployer and the operator.

All these practices taken together result in the emergence of a smooth workflow between the ICN deployer and the network operator.

Requirement 4. *Retain IP as the fallback network.* Live deployments of IP networks are highly engineered to optimize for an acceptable user experience. However, ICN/alternative networks will inevitably be not as robust and might fail from time to time.

Therefore it is highly recommended that tools should provide an easy way to switch traffic between IP and ICN/alternative network to deal with unforeseen problems in the ICN/alternative network.

6 Related Work

To the best of our knowledge, outside our work, there have been no field deployments of ICN involving real users that provided an easy switch between IP and ICN when required. There are four styles of experimental ICN deployment configurations [22] that have been discussed in the literature. These include wholesale replacement of IP by ICN [1–3], ICN deployed as an overlay [3–6], ICN deployed as an underlay -both in the core network [7–9,19–21] and in the edge network [10–12] and finally, ICN-as-a-slice [13]. We deployed ICN in the field with real users as an underlay in the core network as part of the RIFE [9] and POINT [7] project mentioned above, and this paper discusses the practical lessons learnt from the exercise.

In addition we wanted that the ICN should also run in parallel with an IP-based management network over the same physical infrastructure (See Fig. 2). This management network was required so that we could install/update software components, debug, run diagnostic software such as IPerf etc. We used L2 network virtualization (VLAN) to enable such a parallel operation.

There are many examples of using network virtualization to multiplex between various networks and run them in parallel over a shared physical infrastructure. Tempest [14] proposed running many virtual Asynchronous Transfer Mode (ATM)s over the same physical infrastructure. The authors in [15] advocated using a network of overlay tunnels to support multiple networks. Overlay architectures such as PlanetLab [16] and GENI [17] have been used to run different kinds of networks over a common physical substrate. Techniques of slicing a physical network into multiple virtual networks using SDN technology have also been proposed [18]. However, none of these projects ran ICN and IP in parallel using VLAN in a network being used by real users as we have done.

The *Infrastructure as Code* movement has emerged as a way to deal with the complexities of cloud deployment within the DevOps community [26]. Although

many of the practices we used are a subset of the practices being advocated there, our deployments dealt with the specific problem of running ICN and IP stacks together rather than the deployment of cloud-based infrastructure such as VMs, containers etc. which has been the focus of the *Infrastructure as Code* movement.

7 Conclusions

The main concern of this paper has been to discuss general strategies to deal with emergent behavior. These strategies reduce ICN/alternative network deployment times in an operational network that is traditionally optimized for the semantics of IP. The strategies we suggest involve following a dual stack approach and encapsulating the deployed infrastructure in code. We advocate paying particular attention to integrating the code with the workflow between the ICN deployer and the network operator. The order in which the various software/hardware components of the infrastructure are deployed should be generated out of the precedence constraints of those components. The design of this workflow should include an operationally easy way to switch between the ICN and IP network while being minimally disruptive to the customers.

We hope that the general requirements presented in this paper will prove useful as constraining guidelines to develop tools for ICN and other alternative network layer deployments and compliment the ongoing discussions within the Information-Centric Networking Research Group (ICNRG) of the Internet Research Task Force (IRTF). Additionally, we hope that this paper contributes to a set of best practices for the development of tools that are useful at the lowest level of abstraction for designing an Internetwork [23].

Acknowledgement. We want to thank Dr. Dirk Trossen, Huawei, Munich, Ulises Olvera-Hernandz, Interdigital Europe and Dr. Leandro Navarro UPC Barcelona for the many useful suggestions and comments they gave during the preparation of this manuscript.

References

1. NFD Homepage. https://named-data.net/doc/NFD/current/. Accessed 13 Oct 2020
2. Jacobson, V., et al.: Networking named content. In: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies CoNEXT 2009, pp. 1–12. https://doi.org/10.1145/1658939.1658941
3. Trossen, D., Parisis, G.: Designing and realizing an information-centric internet. IEEE Commun. Mag. **50**(7), 60–67 (2012). https://doi.org/10.1109/MCOM.2012.6231280
4. PARC, CCNx Over UDP. https://www.ietf.org/proceedings/interim-2015-icnrg-04/slides/slides-interim-2015-icnrg-4-5.pdf. Accessed 13 Oct 2020

5. Hybrid ICN Cisco: Cisco Announces Important Steps toward Adoption of Information-Centric Networking. https://blogs.cisco.com/sp/cisco-announces-important-steps-toward-adoption-of-information-centric-networking. Accessed 13 Oct 2020
6. Kutscher, D., Farrell, S., Davies E.: The NetInf Protocol. Internet-Draft draft-kutscher-icnrg-netinf-proto-01 (2013). https://tools.ietf.org/html/draft-kutscher-icnrg-netinf-proto-01. Accessed 13 Oct 2020
7. Trossen, D., Reed, M., Riihijarvi, J., Georgiades, M., Fotiou, N., Xylomenos, P.: POINT: IP over ICN - the better IP? In: European Conference on Networks and Communications (EuCNC) (2015). https://doi.org/10.1109/EuCNC.2015.7194109
8. White, G., Rutz, G.: Content delivery with content centric networking, White Paper. CableLabs (2010). https://www.cablelabs.com/wp-content/uploads/2016/02/Content-Delivery-with-Content-Centric-Networking-Feb-2016.pdf. Accessed 13 Oct 2020
9. RIFE Homepage. https://www.rife-project.eu/. Accessed 2 June 2020
10. Zhang, Y., et al.: Design Considerations for Applying ICN to IoT. Internet-Draft draft-zhang-icnrg-icniot-01 (2017). https://tools.ietf.org/id/draft-irtf-icnrg-icniot-01.html. Accessed 13 Oct 2020
11. Ravindran, R., Liu, X., Chakraborti, A., Zhang, X., Wang, G.: Towards software defined ICN based edge-cloud services. In: IEEE International Conference on CloudNetworking (CloudNet) (2013). https://doi.org/10.1109/CloudNet.2013.6710583
12. Azgin, A., Ravindran, R., Chakraborti, A., Wang, G.: Seamless mobility as a service. In: Information-Centric Networks, ACM ICN Sigcomm, IC5G Workshop (2016). https://doi.org/10.1145/2984356.2988521
13. Ravindran, R., Chakraborti, A., Amin, S., Azgin, A., Wang, G.: 5G-ICN: delivering ICN services over 5G using network slicing. IEEE Commun. Mag. **55** (2016). https://doi.org/10.1109/MCOM.2017.1600938
14. Van Der Merwe, J.E., Rooney, S., Leslie, L., Crosby, S.: The tempest-a practical framework for network programmability. IEEE Netw. **12**(3), 20–28 (1998). https://doi.org/10.1109/65.690958
15. Bavier, A., Feamster, N., Huang, M., Peterson, L., Rexford, J.: In VINI veritas: realistic and controlled network experimentation. ACM SIGCOMM Comput. Commun. Rev. **36**(4), 3–14 (2006). https://doi.org/10.1145/1151659.1159916
16. Peterson, L., Anderson, T., Culler, D., Roscoe, T.: A blueprint for introducing disruptive technology into the Internet. ACM SIGCOMM Comput. Commun. Rev. **33**(1), 59–64 (2003). https://doi.org/10.1145/774763.774772
17. Peterson, L., et al.: GENI design principles. IEEE Comput. **39**(9), 102–105 (2006)
18. Yiakoumis, Y., Kok-Kiong, Y., Katti, S., Parulkar, G., McKeown, N.: Slicing home networks. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Home Networks. ACM (2011). https://doi.org/10.1145/2018567.2018569
19. Susmit, S. Fan, C., White, G.: Bridging the ICN Deployment Gap with IPoC: an IP-over-ICN protocol for 5G Networks. In: Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies. ACM (2018). https://doi.org/10.1145/3229574.3229575
20. Refaei, T., Ma, J., Ha, S., Liu, S.: Integrating IP and NDN through an extensible IP-NDN gateway. In: Proceedings of the 4th ACM Conference on Information-Centric Networking. ACM, New York, pp. 224–225 (2017). https://doi.org/10.1145/3125719.3132112

21. Moiseenko, I., Dave Oran, D.: TCP/ICN: carrying TCP over content centric and named data networks. In: Proceedings of the 3rd ACM Conference on Information-Centric Networking. ACM, New York, pp. 112–121 (2016). https://doi.org/10.1145/2984356.2984357
22. Rahman, A., Trossen, D., Kutscher, D., Ravindran, R.: Deployment considerations for information-centric networking (ICN) ICNRG draft (2019). https://tools.ietf.org/html/rfc8763. Accessed 13 Oct 2020
23. Clark, D.D.: Designing an Internet. MIT Press (2018)
24. Baig, R., Freitag, F., Navarro, L.: Cloudy in guifi.net: establishing and sustaining a community cloud as open commons. Future Gener. Comput. Syst. (2018). https://doi.org/10.1016/j.future.2017.12.017
25. Baig, R., Roca, R., Freitag, F., Navarro, L.: guifi.net, a crowdsourced network infrastructure held in common. Comput. Netw. 90, 150–165 (2015). https://doi.org/10.1016/j.comnet.2015.07.009
26. Kief, M.: Infrastructure as Code: Managing Servers in the Cloud. O'Reilly Media, Inc. (2016)