ICN software tools: survey and cross-comparison

M. Tortelli[a,*], D. Rossi[a], G. Boggia[b], L.A. Grieco[b]

[a] *Department of Computer Science and Networking, Telecom ParisTech, Paris, France*
[b] *Department of Electrical and Information Engineering, Politecnico di Bari, Bari, Italy*

Abstract

Research interest on Information Centric Networking (ICN) has been sharply growing. Although new architectures, algorithms, and analytical models have been proposed, their evaluation remains often isolated and not rigorously verified by the research community. This paper initially portrays the composition of open source software tools available for ICN, certifying the predominance of Content Centric Networking (CCN)/Named Data Networking (NDN) simulators. Then, inspired by similar works related to the P2P field, it surveys related research papers to qualify the ICN literature produced so far, finding that a large fraction of contributions either uses custom, proprietary, and unavailable software, or even plainly fails to mention any information in this regard. By adopting a rigorous methodology, in the second part of the paper four simulators, namely *ndnSIM*, *ccnSim*, *CCNPL-Sim*, and *Icarus*, are cross-compared under several key aspects. Our findings confirm both their accuracy with respect to reference theoretical models in simple settings, and their consistency in more complex scenario. Additionally, our analysis can both assist researchers in the choice of the tool that best fits their needs, and provide guidelines to avoid common pitfalls in the ICN performance evaluation.

Keywords: Information Centric Networks; Content Centric Networks; Open-Source Software; Prototype; Emulation; Simulation; Performance Evaluation.

1. Introduction

Information Centric Networking (ICN) is a novel paradigm that encompasses different architectures with a common concept: data objects become first class citizens that can be referred directly by their name, instead of being retrieved through addressable endpoints [1, 2]. Most of the ICN architectures (Sec. 2) can

*Corresponding author
Email addresses: tortellimichele@gmail.com (M. Tortelli), dario.rossi@telecom-paristech.fr (D. Rossi), gennaro.boggia@poliba.it (G. Boggia), alfredo.grieco@poliba.it (L.A. Grieco)

be progressively deployed over the consolidated Internet network infrastructure, in order to better handle, at an architectural level, current trends like mobility and content distribution, instead of continuously deploying patches (e.g., mobile IP [3] and Content Distribution Networks (CDNs)). Deploying a large scale network with content-awareness at the network layer, like ICN, could introduce remarkable benefits with respect to the current Internet protocols, like [2, 4]: (i) better support to mobility; (ii) robustness of content-based schema, which directly secure transmitted contents, with respect to IP-based ones, which secure end-hosts; (iii) native multicast and multipath communications.

For the above reasons, ICN attracted considerable attention in the last decade, and the investigation of pros and cons of its principles let a number of proposals emerge, that are surveyed in [1, 2]. These proposals differ from each other regarding naming schema, routing algorithms, deployment models, and interoperability between IP-based and name-based solutions. As such, performance evaluations of each ICN architecture still remains isolated. Building over our previous effort [5], this paper makes the first steps in this direction: our aim is not to compare different ICN proposals at an architectural level (which is the focus of [1, 2]), but rather to perform a critical overview of available software tools for performance evaluation (an aspect that is, instead, out of the scope of [1, 2]). Similarly to [6], that considers the state of Peer-to-Peer (P2P) simulators, in this work we build a critical census of ICN software, along with its usage in the literature, and we additionally carry on a scientifically rigorous cross-comparison aimed at verifying the accuracy, consistency, scalability, and fidelity of these tools. We believe this contribution to be useful in both providing the scientific community with solid guidelines for choosing the most suitable simulator according to the type of investigation that needs to be carried out, and confirming the scientific soundness of the results that the ICN community has already published.

In particular, after a background on ICN (Sec. 2), the first contribution of this paper is to portray the general picture of ICN projects and software tools, quantifying, also, their usage in the ICN literature (Sec. 3). After, the main features of all tools are illustrated (Sec. 4). Results of this census show that:

- Overall, the ICN software ecosystem is lively and variegated, with a rich spectrum of open source software, including prototypes, emulators, simulators and possible bindings between the different categories.

- Generally, however, ICN architectures released mostly prototype implementations, so that the lack of simpler tools, such as simulators, already hampers the evaluation of each architecture in isolation.

- Availability of software tools for the CCN/NDN architecture [7], is far bigger than for other ICN architectures: overall, CCN/NDN software represents about half of the whole ICN software ecosystem.

- Diversity of software tools for the CCN/NDN architecture is far bigger than for other ICN architectures: additionally, while all the spectrum of tools is well represented (e.g., simulators, emulators, prototypes, etc.), about half of the CCN/NDN software tools is constituted by simulators.

- Yet, despite this richness of open-source software, our survey of the ICN literature reveals that about two thirds of the considered works either claims to use a custom proprietary tool, or just plainly fails to even mention the adopted tool, thus precluding any possibility to reproduce or validate the published results.

Knowledge gathered in the first part lays the ground for comprehensive cross-comparison of a representative subset of ICN software. We indeed find that, whereas in the current stage it is impossible, and perhaps meaningless, to conduct a rigorous cross-comparison in the *overall ICN scope*, this comparison becomes resonant and feasible by limiting the focus to *CCN/NDN simulators*, i.e., the architecture and software-category pair for which the largest set of tools is available. We lay out a rigorous methodology (Sec. 5), based on which we contrast four simulators, namely *ndnSIM*, *ccnSim*, *CCNPL-Sim*, and *Icarus*[1]. Our comparison involves both qualitative and quantitative aspects, such as accuracy (Sec. 6), consistency and scalability (Sec. 7), as well as fidelity (Sec. 8), that we analyze thoroughly. From our cross-comparison it emerges that:

- The limited set of common features shared among the tested simulators reflects that each tool is generally conceived for specific aspects (i.e., architectural completeness, caching performance, congestion control, and so on) of the whole architecture.

- Since available tools present different approaches regarding the automatic configuration and consistency check of the simulated scenario (e.g., dynamic vs fixed warmup period), users should put attention in carefully engineering their scenario in order to attain acceptable levels of accuracy (e.g., generating a proper number of simulated events, in accordance with the popularity distribution and the cardinality of the content catalog, allows to drastically reduce estimation errors).

- A similar care is required to ensure fidelity of the results (e.g., simulating arrival rates that are compatible with available link capacities to avoid distortions of collected metrics, especially for those simulators that neglect link capacity, delays, and transmission buffers).

- Results provided by the selected simulators are both accurate with respect to analytical benchmarks in case of simple scenarios (i.e., single cache), as well as consistent with each other on more complex scenarios (e.g., large scale network with routing).

[1]Despite Icarus is proposed as a simulator for general ICN architectures, it can be easily adopted to perform CCN/NDN based simulations, as explained in Sec. 5.1.

This is comforting in that, despite there is still considerable room for improvement in making ICN results more reproducible, the ICN software ecosystem facilitates this progress in yielding consistent results, unlike in related networking area (Sec. 9). Hopefully, this work will facilitate even further this process by guiding newcomers into choosing an existing software tool that best fits their needs according to technical features and performance characteristics (as opposed to developing a proprietary and untested tool). Finally, we discuss community-wide efforts concerning universally accepted benchmarks and guidelines (see Sec. 10) that would be desirable to further progress along solid and paved ICN research roads.

2. Background

This section aims at providing not a thorough survey, but the minimum amount of information that would make this paper self-contained, along with useful pointers to extended documents, as published scientific articles [1, 2, 8, 9], or Internet drafts [10, 11, 12] written and updated by an IRTF research group, namely Information-Centric Networking Research Group (ICNRG). More attention is dedicated to the CCN/NDN architecture, since it is the focus of our cross-comparison in Sec. 6, Sec. 7, and Sec. 8.

2.1. ICN at a glance

In ICN, contents are named and addressed independently of their location; as a consequence, data can be stored and retrieved from everywhere in the network. Digital cryptographic signatures are carried inside each packet, and used to directly verify integrity, authenticity, and provenience; that is, security is implemented at the information level and not around the communication session [1, 2]. This means that ICN nodes, being aware of the contents that are forwarded, can play an active role in caching and forwarding decisions, thus directly satisfying requests at the network layer. Furthermore, requests for the same contents can be aggregated at intermediate routers along the paths toward the original content providers, thus transparently building multicast diffusion trees and unloading popular content providers in case of flash crowds. The possibility to decouple contents from node locators is, also, beneficial in terms of mobility; burdensome solutions (i.e., mobile IP [3]), as well as disruptions generated by handovers, can be, indeed, avoided [13, 14]. These changes are not only of paramount importance for nowadays' killer applications (e.g., video [15]), but are expected to facilitate future services as well (e.g., ICN-based Internet of Things [16] [17]).

However, it is worth highlighting that, despite the centrality that named objects have against network locators (i.e., IP addresses) in each communication primitive, ICN does not completely disrupt the current TCP/IP end-to-end model [18].

2.2. ICN architectures

While seminal works leading to ICN can be traced back to the TRIAD project [19, 20] and other works [21, 22, 23] which date back to almost twenty years ago, the architectures that are generally referred to as "Information-centric" have been mainly proposed in the last decade: Data-Oriented Network Architecture (DONA) [24], CCN/NDN [7], Publish Subscribe Internet Technology (PURSUIT) [25], Network of Information (NetInf) [26] of the Scalable and Adaptive Internet Solutions (SAIL) project [27], COntent Mediator architecture for content-aware nETworks (COMET) [28], CONVERGENCE [29], and MobilityFirst [30], eXpressive Internet Architecture (XIA) [31] (considered part of ICN architectures for its in-network caching feature and for the possibility to use content names as identifier), and GreenICN [32]. Although they share the aforementioned principles, they differ in implementation details of some key aspects, like naming schema, request and data routing, caching and security.

The NDN architecture adopts hierarchical names to uniquely identify pieces of contents, namely *chunks*, as opposed to flat-label IDs that are, instead, used by other architectures, like DONA and MobilityFirst. In PURSUIT, SAIL, CONVERGENCE, and GreenICN instead, content names, or part of them, can be either flat or hierarchical. The XIA architecture, in the end, differs from the previous ones since its addressing scheme is more "flexible" [33]: Directed Acyclic Graphs (DAGs) can be used to address multiple identifiers (e.g., host, service, content, network). The type of content name directly influences other aspects, like routing and security. In fact, in NDN, a *route-by-name* approach is adopted, which means that content requests are directly forwarded based on content names; like NDN, also DONA, COMET, and CONVERGENCE adopt name-based routing. PURSUIT, SAIL, MobilityFirst, GreenICN, and XIA, instead, require a two-step resolution phase (i.e. *lookup-by-name*), in which a locator, obtained by looking up the name of the requested content, is used to forward content requests. Furthermore, the namespace structure has an impact on security aspects; indeed, flat names can by easily self-certifying (e.g., by hashing all the content fields), while hierarchical names require in-packet meta-data to testify integrity, authenticity, and provenience of the retrieved content.

In NDN, nodes interact through a receiver-driven communication model by using only two types of messages: *Interest* and *Data* packets. *Interest* packets are sent by clients to fetch contents, whereas *Data* packets are generated only in response to *Interest* packets; this one-to-one match makes NDN flow-balanced. *Interest* and *Data* processing in NDN nodes is done by means of three structures: Content store (CS), Pending Interest Table (PIT), and Forwarding Information Base (FIB). The CS is a cache memory where forwarded contents can be stored. The PIT is a table used to keep track of those forwarded *Interest* packets that are still unsatisfied. These states left along traversed nodes [34] bring several benefits: *Interest* packets for the same content can be aggregated into one PIT entry, recording only the respective incoming faces. This gives NDN an innate support for multicast communication. Indeed, generated *Data* packets can simply follow back the created paths towards their requesters. Furthermore, PIT entries allow NDN nodes to detect loops and discard the relative *Interest* packets. Finally, the FIB is a table where forwarding information about the known content prefixes are stored; differently from IP, multiple faces can be associated to a single name, thus simplifying the support for multipath communications.

3. Survey of tools and usage

3.1. ICN software tools

Tab. 1 summarizes the results of the census of ICN open source softwares considered in this contribution: to the best of the authors' knowledge, the list includes *all* open source software tools.

It is worth to note that, about half of them pertains to different proposals under the general ICN umbrella (top portion of Tab. 1), whereas the other half pertains to a specific architecture (i.e., CCN/NDN [7], bottom of Tab. 1). Interestingly, despite CCN/NDN is among the last proposals in the general ICN timeline [2], it however attracted, by far, the broadest interest, as also reflected in the heterogeneity of available tools for its evaluation.

For each software, the table reports the latest release and its date, that correlates with the development activity (notice that some software reports no explicit tags for its release date, thus file headers are inspected). Then, the language used by the core library (correlated with the learning and startup costs) and the additional languages for accessory software (either bindings of the main library API for a number of scripting languages, or languages used to develop the accessory applications) are also reported in the table. Operating systems supported by each software are reported next, which could represent a potential barrier to development and deployment. Finally, type of software and state of support are reported. Legend under Tab. 1 reports, also, the funding authority and the relative project under which each tool has been released. As a final remark, the table is voluntarily kept simple to allow to grasp the ICN software census at a glance. As a result, the table is missing some relevant information (e.g., the software license, the existence of proper documentation, the number of active developers, the number of releases, the frequency of commit, the number of lines of code, and so on), that are either not available across all systems, or that would anyway quickly become outdated.

From Tab. 1 it appears that, for most ICN proposals, only a single type of software, and most often a prototype implementation, is available. Exceptions to the above rule, i.e., where also simulators are available, are represented by the GreenICN project, the Publish Subscribe Internet Routing Paradigm (PSIRP)/PURSUIT, and by the CCN/NDN architecture; in particular, the Icarus simulator is associated with the GreenICN project, while the ICNsim simulator is available for PURSUIT, along with a couple of prototype implementations (Blackhawk/Blackadder).

It appears clearly that the only ICN architecture with a complete software ecosystem is CCN/NDN, where not only prototype implementations exist (i.e., CCNx , NFD, CCN-lite, CCN-Joker), but also bindings of these implementations with known emulators (e.g., MiniNet for CCNx), or simulators (e.g.,

Table 1: Taxonomy of ICN open source softwares.

	Software$^{\$}$	**Latest rel**		**Language core (extra)**†	**Operating system**$^{\odot}$	**Tool type***	**Active**
		ver	**date**				
Others	Blackadder$^{\diamond 1}$	0.4	04/13	C++, C (Py,R,J)	L,F	P(Click)	Yes
	ICNsim$^{\diamond 2}$	0.2	04/15	C++, C	L	S(Omnet++)	Yes
	openNetInf$^{\diamond 3}$	1.0	10/11	J	Portable	P,VM	No
	NetInf (nilib)$^{\diamond 3}$	0.2	10/12	C (all)	L	P	No
	GIN$^{\diamond 3}$	1.0	01/13	C (PH)	F	P	No
	PeerKit$^{\diamond 4}$	-	12/11	J	Portable	P	No
	Conet$^{\diamond 4}$	-	12/12	C	L	P,T(OFELIA)	No
	XIA$^{\triangleleft 1}$	1.2	06/15	C (Py)	L,M	P,VM(Click),T	Yes
	MobilityFirst$^{\triangleleft 2}$	-	12/14	C++	L,A	P(Click)	Yes
	Icarus$^{\diamond 5}$	0.5	05/15	Py	L,M	S,VM	Yes
CCN/NDN	CCNx$^{\triangleleft 3}$	1.0	07/15	C (J)	L	P,T(NDN)	Yes
	NFD$^{\triangleleft 4}$	0.3	07/15	C++	L,M,F	P,T(NDN)	Yes
	Mini-CCNx	-	05/15	C,Py	L	E,VM(Mininet)	Yes
	CCN-Joker	2.1	10/13	J	L,A	E,T	No
	CCN-Lite	0.3	07/15	C	L,M,A, RFduino	P,E, S(Omnet++)	Yes
	ns3-DCE CCNx$^{\ddagger 1}$	4.0	12/13	C (Py)	L	S(NS3)	Yes
	CCNPL-Sim$^{\ddagger 1}$	1.0	02/13	C++	L	S	Yes
	NDNsim$^{\triangleleft 4}$	2.0	01/15	C++(Py)	L,M	S(NS3)	Yes
	ccnSim$^{\ddagger 1}$	0.3	10/14	C++	L	S(Omnet++)	Yes

$^{\$}$*Software legend*: Symbol=Funding Authority, #=Project
$^{\diamond}$=FP7: #1=PURSUIT, #2=PAL, #3=SAIL, #4=Convergence, #5=GreenICN
$^{\triangleleft}$=NSF: #1=XIA, #2=MobilityFirst, #3=CCNx, #4=NDN
‡=ANR: #1=Connect
†*Language legend*: Py=Python, J=Java, PH=PHP, R=Ruby, cl=clojure, all=P,J,R,PH,cl
$^{\odot}$*Operating System legend*: L=Linux, M=MacOS, F=FreeBSD, A=Android
**Tool legend*: P=Prototype, VM=Virtual Machines, E=Emulator, S=Simulator, T=Testbed.

NS3-DCE for CCNx, Omnet++ for CCN-lite), thus allowing the execution of the same code in operational networks, controlled testbed, or simulators. Additionally, more traditional "stand-alone" simulators are available (i.e., ndnSIM, CCNPL-Sim, ccnSim), each of which focuses on different CCN/NDN aspects, enhancing the spectrum of tools available for performance evaluation.

Overall, our census indicates that ICN is a lively and healthy domain, with a number of prototype implementations. Furthermore, APIs for the core ICN libraries are available in many languages, and often support multiple operating systems, thus lowering the additional startup cost of approaching unknown experimental environments. At the same time, with the exception of CCN/NDN, ICN architectures generally lack of simpler tools, such as simulators, which simplify the performance evaluation of each architecture in isolation.

This paper presents a first attempt to systematically address the above problems. In particular, one of our main aims is to assess if, and to what extent, available ICN software tools allow a scientifically sound evaluation: we consider the CCN/NDN architecture as a reasonable starting point for this task, due to both its relevance in the ICN domain, and to a larger availability of tools reflected by our census.

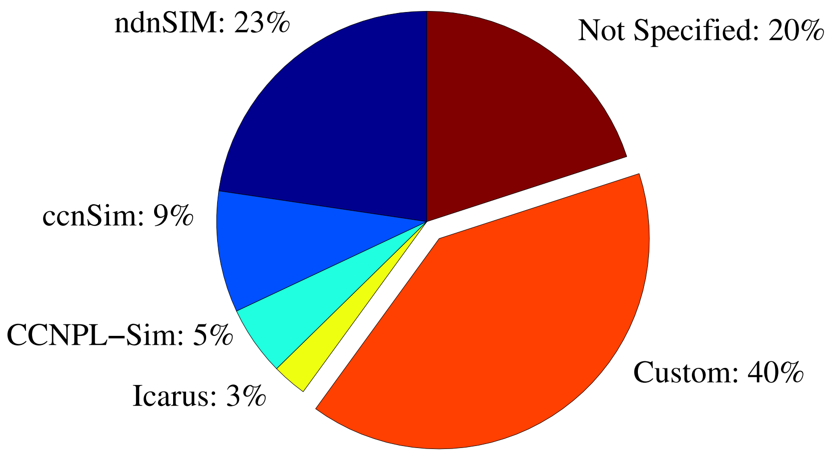


Figure 1: ICN Simulation tools used in the surveyed papers.

3.2. Software usage in the ICN literature

As a second step of our survey, we take a snapshot of the actual use of the aforementioned tools in the ICN literature. This kind of investigation has been inspired by findings outlined in [35, 6], that considers 9 popular P2P simulators and surveys about 300 papers on P2P topic; the discovery is that only about 20% of papers used one of these 9 popular simulators, whereas the vast majority of papers either claimed to use a custom proprietary simulator, or did not even made an explicit mention – implying that P2P simulation results were generally reported in the literature in a fashion that precluded any reproduction, validation or invalidation of the published results.

Therefore, in similar spirit, we overview papers in the ICN literature[2]. In particular, we select 75 papers that specifically employ *simulation tools* in their performance evaluation, out of 93 papers that conduct an evaluation of ICN performance using different tools, like emulators, prototypes, testbed, and so on. Our selection criterion did not use any filter regarding number of citations and/or area of interest inside the broad ICN domain. Clearly, with respect to [35, 6] that overviews P2P research in (or just after) its climax, this work is still comparatively premature – as research in ICN is, though fast growing, still younger. Despite the amount of surveyed papers is limited, our picture allows to gather interesting insights – that are very similar to that in [35, 6].

The pie chart in Fig. 1 reports the results of the survey, where the label *Custom* stands for both papers in which the authors claim to use their custom simu-

[2]Namely, we include the ICN conferences, as well as the following series of ICN workshop or conferences having an ICN session: ACM SIGCOMM ICN, IEEE INFOCOM NOMEN, ACM/IEEE NoM, IEEE ICNP, IFIP Networking, IEEE CCNC, CFI and NoF. We also include IEEE, ACM, and Elsevier Journals.

lator, and papers in which generic tools, like Matlab, Omnet++, ns-2, QualNet, ONE, and BitTorrent are mentioned without any indication about the required modifications and/or without any reference to the used code. From the picture it clearly emerges that: (i) there is a variegated set of ICN simulation tools mentioned in the surveyed papers, the majority of which are CCN/NDN simulators; (ii) with the exception of Icarus, the only simulator outside CCN/NDN, i.e., ICNsim, is not mentioned; (ii) the most popular simulator is ndnSIM (23%); (iii) about 2/3 of presented results are not reproducible, because either the authors have used a custom simulator (i.e., 40%), or they have not even specified the tool used for the evaluation part of their proposal (20%).

While these numbers are slightly more encouraging than [6], it is worth to stress once more that the situation can evolve, and diverge, with, e.g., an increasing popularity of proprietary tools without any clear indication of their soundness. Another aim of this paper is thus to suggest good practices to reinforce the soundness of research results published in the ICN community. We do so by both proposing rigorous methodologies (e.g., to cross-check results of a new software against those of already tested tools in order to prove their validity) as well as means to enforce them. We hope that our contribution can encourage the ICN research community to take the right countermeasures, illustrated in Sec. 10, to avoid being stuck in unpleasant scenarios like the one depicted in [35, 6].

4. Software Description

This section provides a brief description of all the software tools listed in Tab. 1, Due to the predominance of the CCN/NDN in the ICN software tools ecosystem, we discuss the former separately from the other architectures.

4.1. ICN software (except CCN/NDN)

The scope of the ICN software can be classified according to the respective reference project. Excluding CCN/NDN, tools can be connected to either the PSIRP/PURSUIT or 4WARD/SAIL project suites, to the GreenICN project, or to one of the CONVERGENCE[3], MobilityFirst or XIA projects[4].

As previously outlined, most of the software released in the ICN scope are prototype implementations of a reference ICN architecture; the only two exceptions are represented by the ICNsim implementation, carried out in the PAL project [36], of the architecture proposed in the PSIRP/PURSUIT project suite,

[3]The knowledgeable reader will notice that though CONVERGENCE builds over CCN, it also modifies some aspects with respect to its original proposal [7], so that it fits best under a larger ICN umbrella.

[4]Projects such as COMET (http://www.comet-project.org) or COAST (http://www.coast-fp7.eu) are, instead, omitted from the list because, although closely related in scope, no commitment in releasing their software tools as open source to the scientific community has been found

and by the Icarus simulator, related to the GreenICN project, which is promoted as a Python-based simulator for general ICN architectures. Notice that it is possible that multiple software implementations of the same architecture are released by the same (or followup) project, as for the Blackhawk/Blackadder implementations (PSIRP/PURSUIT project suite), or openNetInf/nilib implementations (4WARD/SAIL suite). In the following, a brief overview of each software is provided.

4.1.1. PSIRP/PURSUIT and PAL

This project suite [37, 38] has released two software prototypes. The earliest one, named Blackhawk [39], is implemented in C and it integrates the publish/subscribe system directly in the FreeBSD kernel. To reduce the duration of the initial learning phase, wrappers for the low level library are provided in several high-level scripting languages, such as Python and Ruby, that are obtained through SWIG [40]. Similarly, to lower the start-up cost, ready to use FreeBSD virtual images, running Blackhawk, are provided for KVM, VMware or VirtualBox. However, the most recent Blackhawk software was released on June 2010, and the Blackhawk development has been discontinued. The PSIRP/PURSUIT development, then, continued in the Blackadder [41] prototype, which is currently maintained in the context of two new European projects, that are iP Over IcN the betTer IP (POINT) [42] and RIFE [43]. The most relevant changes with respect to the previous Blackhawk are: the adoption of the C++ language on Linux (instead of C and FreeBSD); the implementation of the entire prototype as a Click [44] modular router component (supporting both kernel and user mode); the addition of some wrappers (e.g., Java and C), and applications (VLC-based pub/sub video). An important effect of having implemented Blackadder as a Click modular router component is that, albeit the software was primarily meant to be a full-blown prototype, it is possible to run simulations through NS2/NS3 integrations of the Click module (known as nsclick/ns-3-click).

However, further effort in the PURSUIT area [45] seems to suggest that running the prototype code in a simulated environment, by hacking low level calls of the OS networking stack, lacks of the necessary flexibility and further burdens the development with respect to simpler abstractions commonly used in simulations. This motivated the development of ICNsim [46], an Omnet++ based simulator of the PSIRP/PURSUIT architecture. ICNsim is developed in the context of the PAL project [36], which aims at studying the impact of future health-care services on current and future communication infrastructures. Although ICNsim is able to simulate a fair number of nodes and publisher-subscriber pairs with all their basic modules, it lacks of a faithful representation of the whole PURSUIT architecture, being mainly focused on the network topology management part in order to design and study algorithms for path computation. Two aspects contribute to the unclear upper-bounds of its scalability: (i) [45] takes into account only relatively simple topologies to showcase the scalability of ICNsim; (ii) it uses the INET framework for OMNET++, whose packet-level design introduces additional burden (like message passing between modules), with respect, for example, to a monolithic simulator that rely on analytical models and/or flow-based design principles [47].

4.1.2. 4WARD/SAIL

This project suite [48, 27] has released three open source implementations of their NetInf and Global Information Network (GIN) architectures.

openNetInf [49], the oldest between the two implementations concerning NetInf, is developed using the Java language, thus introducing portability over several platforms, including Microsoft Windows. The core library has a significant amount of documentation and it is complemented by plugins for popular applications, such as Firefox and Thunderbird, which let browse NetInf-enabled servers and send emails to Information Object (IO) identifiers rather than email addresses, respectively. As for the Blackhawk prototype, openNetInf is also available as an Ubuntu-based Linux virtual image for VirtualBox. Interestingly, open testbed facilities are available, with one publicly accessible node[5] running the openNetInf software (which can lower startup cost for testing purposes).

nilib [50], a C implementation of NetInf (with wrappers in Python, Java, Ruby and Clojure) has been released next. A notable difference with respect to openNetInf is the implementation of Named Information (NI) URI schemes, including truncated hash suites, and binary vs human-readable name formats. As the authors explicitly stated, the library has been used in interoperability tests, though the code can be considered as a work in progress, and the ecosystem surrounding it is, at time of writing, less mature with respect to the openNetInf one (e.g., absence of virtual machines or testbeds).

Finally, in the context of the SAIL project, an open source implementation of the GIN architecture [51] has been released (albeit the relationship with the NetInf architecture is not fully clarified, it seems that GIN is a more general architecture in which NetInf could be included). The GIN prototype has been developed on FreeBSD 8.2, in C and PHP languages (while the demo and evaluations employ virtual machines, these are not readily available for download, and are thus not listed in Tab. 1).

Overall, it can be pointed out that, as the 4WARD/SAIL projects ended, code maintenance either stopped (as for openNetInf[6]) or it relies only on the effort of individual partners, which, consequently, could slow down the progress of the projects.

4.1.3. CONVERGENCE

This project [52] has released prototype implementations of its architecture, available in two software packages, namely PEERKIT and CONET (which is developed in cooperation with the OFELIA (http://www.fp7-ofelia.eu/) project).

[5]Actually any `*.testbed.netinf.org` domain name is aliased to the same machine running multiple NetInf virtual nodes `vhosts.eim.uni-paderborn.de`

[6]The mercurial branch has been tagged as v1.0_prePg3 and v2.0_pg3_final on Nov. 4th, 2011.

In particular, PEERKIT [53] is a Java middleware extension of the new MPEG-M standard which supports distributed applications that create, trade and consume digital objects (the released software includes a simple application for photo/document sharing).

Network level functionalities are provided by CONET [54], which adopts both an evolutionary approach, in the sense that name-based communications are integrated in existing IP networks by using a new header option, and alternative lookup methods based on a name-system (instead of a full dissemination of names via a routing protocol as in CCN). CONET nodes use essentially the same interface and APIs of CCNx (see Sec. 4.2.1): this allows, on the one hand the reuse of existing CCNx applications, and, on the other hand, it facilitates a direct comparison between CCNx and CONET (indeed, a simple testbed scenario for CCNx vs CONET comparison is also provided). It is worth noting that, while the first version of CONET was based on the CCNx 0.6.2 prototype, recent releases have been updated to 0.7.1 – yet it remains unclear in which way and for how long it will follow further updates of the CCNx codebase, whose latest release is the 1.0.

Additionally, a CONET testbed is available through the OFELIA project, that aims at providing open experimental facilities to the research community. More precisely, as OFELIA focuses on Software Defined Networks (SDNs), OpenFlow-enabled switches in the testbed can process CONET-compliant traffic (i.e., serving it from local caches) and fallback to standard IPv4 otherwise.

4.1.4. XIA

The XIA project [31] released a prototype for Linux and Mac OS [55], built over the Click modular router [44]. The prototype includes all architectural components as well as APIs (in C and Python) along with sample applications and libraries. As previously noticed for Blackadder, it is possible to run simulations through the NS2/NS3 integrations for the Click module – though the same comment also applies here in terms of the learning cost and complexity of prototype-based simulative approach.

Like the aforementioned projects, XIA released not only the source code to be built and run on custom systems, but it also provided VMs that can be used to evaluate the tool in local testbed. Additionally, XIA can be run on the Global Environment for Network Innovations (GENI) testbed, through which large scale experiments can be carried out.

4.1.5. MobilityFirst

The MobilityFirst project [30], funded by the NSF's Future Internet Architecture (FIA) program, is currently focused on the design and evaluation of enhanced network services targeting mobile data and content, Internet-of-Things, as well as cellular-Internet convergence (i.e., single unified Internet architecture supporting the needs of cellular systems including both 4G HetNet and 5G). Recently, a prototype code has been released [56] as a collection of independent and interoperating components implementing the basic features of the MobilityFirst architecture, such as a Click-based software router, a Global Name Resolution

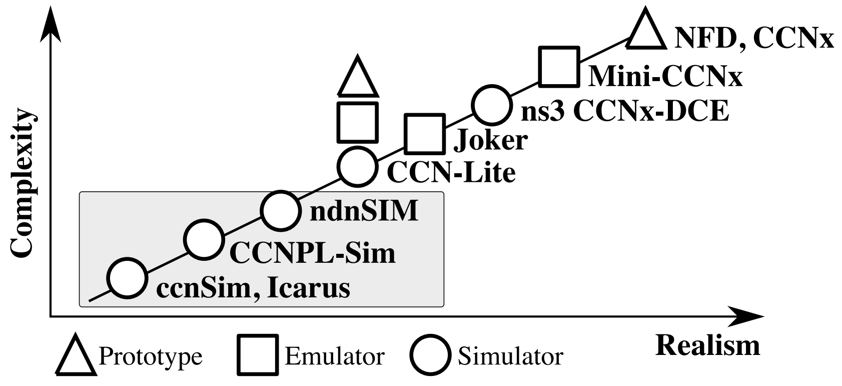


Figure 2: Realism vs Complexity in CCN/NDN softwares.

Service (GNRS), and a host protocol stack. C/C++ and Java libraries are available as APIs to interface with the client network stack, as well as tutorials explaining how to create MobilityFirst experiments on several testbeds, such as ORBIT and GENI.

4.1.6. GreenICN

The GreenICN project [32], which focuses on the issues of scalability and energy efficiency in ICN networks, has its own software reference in the Icarus simulator, available at [57]. Being entirely written in Python, the Icarus simulator, described in [58], can leverage on networking libraries such as NetworkX and Fast Network Simulation Setup (FNSS). The authors claim that Icarus is not tied to any specific ICN architecture; indeed, its main target consists in the study and performance evaluation of networks of caches. It follows that, its most developed components concern cache replacement/decision policies, and cache/routing strategies. Icarus makes the choice of scalability, which is achieved at the expense of fidelity: basic structures are simplified (e.g., name space, FIB), and network features like link capacities, buffers, etc. are neglected by default. Additional utilities are provided for modeling the performance of caches and work with traffic traces, as well as instructions to run Icarus within a virtual machine.

4.2. CCN/NDN Software

As emerges from Tab. 1, many tools are available for CCN/NDN with a complementary design, i.e., there are different operational points in the fidelity vs scalability space, (or, equivalently, in the scalability vs complexity space), depicted in Fig. 2. The separation of the CCN project into two different branches has also contributed to this heterogeneity. Indeed, the initial and unified research efforts boosted by the seminal paper [7], have split into two directions which differ in both protocol specifications and associated software tools: from on side there is the CCNx project [59], conducted by the Palo Alto Research

Center (PARC), with its CCNx software reference, while, on the other side, there is the NDN project, conducted by UCLA and other institutions, with their NDN Forwarding Daemon (NFD) reference codebase.

Trying to shortly describe each CCN/NDN-related software with a single keyword, we might say that CCNx and NFD aim at *deployment*, meaning that, as reference codebases for the CCN/NDN deployment on real networks, they include the whole set of features provided by the specifications of the relative architectures, concerning, e.g., security, fragmentation, encapsulation, and packet format. CCN-lite and CCN-Joker, instead, aim at *simpler emulation*, in the sense that they implement a reduced set of functionalities in order to execute CCN/NDN code on resource constrained devices. Mini-CCNx, on its side, aims at *flexible emulation*, since it still implements a reduced set of features like CCN-lite, but it exploits all the advantages of Mininet (e.g., Container-Based Emulation) in order to alleviate the burden of real emulation. NS3 CCNx-DCE and ndnSIM aim at *faithful simulation*, in the sense that they adapt the codebases of the relative architecture, that are CCNx and NFD, respectively, inside a simulation environment, like NS3, without penalizing completeness. CCNPL-Sim, ccnSim and Icarus aim at *packet-level simulation*, but with a different level of granularity which reflects the respective goal: indeed, CCNPL-Sim offers a *fine-grained* implementation of some aspects, like link capacity and delay, or packet size, since it is particularly designed for congestion control studies; on the other hand, ccnSim and Icarus offer a coarse-grained implementation of network dynamics since they aim at providing a *scalable* tool to study properties of large-scale network of caches.

Rather, as emerges from their brief description presented in this section, each of these tool covers (almost) non overlapping portions of the design space spectrum shown in Fig. 2. To facilitate the exposition, tools are presented by grouping them into two categories: (i) prototypes and emulators vs (ii) simulators, briefly described in the following.

4.2.1. Prototypes and Emulators

CCNx. The CCNx prototype [59] is the reference implementation of the CCN architecture; it is a fully fledged prototype initially supported by US government funding through the National Science Foundation (NSF); its development is lead by PARC, with the support of a very broad community. Since the seminal paper [7], the CCNx project has evolved with new specifications regarding different aspects of the protocol stack; the most recent ones related to CCNx version 1.0 can be found at [60].

The core of the CCNx prototype is written in C, and it perfectly adheres to the specifications of the CCN protocol, such as those related to security aspects, packet format, semantics, fragmentation, Data chunking, and so on. It, also, includes some applications, developed in Java and C, that natively run over CCN; examples are a simple text chat tool (that allows users to build simple applications and/or to manage testbed operations), an audio conferencing tool, as well as VLC and Wireshark plugins. As CCNx does not provide per se any testing, validation or experimentation capabilities, users have to instrument ad-hoc local testbeds or use dedicated infrastructures. Hence, two software tools (discussed next) were developed to facilitate the execution of CCNx emulation (i.e., Mini-CCNx) and simulation (i.e., ns3 CCNx-DCE), maintaining the same CCNx codebase. At the time of writing, Mini-CCNx and CCNx-DCE are based on CCNx version 0.8.2 and 0.6.2, respectively.

As previously stated, the latest open source release of CCNx has been the 0.8.2, whose components where distributed under both the GNU General Public License (GPL) and the GNU Lesser General Public License (LGPL) version 2. After that, the development of the CCNx continued, but as a proprietary software with version 1.0. More specifically, at the time of writing, CCNx 1.0 is distributed either as a binary package under a PARC license, or with its source code under two different forms: an evaluation license for educational institutions, which is free, or an evaluation license for commercial institutions, with an annual subscription that needs to be paid for commercial use. In addition to the type of license, CCNx 1.0 introduced some changes with respect to its previous version, such as a richer and organized set of APIs concerning Interest and Content object handling, key/value store, message queue, etc. It also comes with results of unit testing which prove its increased simplicity and efficiency with respect to version 0.8.x.

NFD. The NDN project [61] was, also, initially started through US government NSF funding as one of the proposals aimed at shaping the Future Internet Architecture (FIA). Currently, several universities and research laboratories, under the coordination of the UCLA university, contribute to the definition of the NDN architecture, as well as to the development of the relative tools.

The NDN project has its own reference codebase in the NFD [62], which definitively breaks the compatibility with CCNx. The first difference is represented by the programming language, which for NFD is C++, thus strengthening characteristics like *extensibility* and *modularity*. In addition, NFD comes with the GPL 3.0 license, which allows the open source community to freely test and develop the code.

NFD faithfully reproduces all the features of the NDN protocol, from NDN Type-Length-Value (TLV) packet format, link layer protocols, main tables, forwarding and strategy support, routing management, to hash computation routines, DNS resolver, and so on. Furthermore, it makes use of the ndn-cxx, which is a C++ library implementing NDN features useful to develop various NDN applications. As part of the NDN project, APIs, called NDN Common CLient Libraries (NDN-CCL), are, also, available in C++, Python, JavaScript, and Java to develop client applications which interact with the NFD.

Mini-CCNx. Built on top of the container-based emulation environment provided by Mininet-HiFi [63], this tool [64] allows users to instantiate experiments with hundreds of CCNx nodes that run the official CCNx project code (version 0.8.2). Interestingly, Mini-CCNx allows to automatically instantiate a replica of the NDN testbed, including the actual topology, annotated links, and name-based prefix configuration – to further bridge the gap between experimental evaluation in controlled environments vs real networks. Mini-CCNx can be installed either directly on Linux devices, or on pre-configured VM.

CCN-Joker. CCN-Java Opensource Kit EmulatoR (Joker) [65] is an open source platform for NDN emulation. It is entirely written in Java (which simplifies the development workflow) and, as for CCN-lite, only the fundamental features are implemented (such as Interest/Data handshakes and cache management policies, the Additive Increase Multiplicative Decrease (AIMD) congestion control mechanism for NDN, etc.), to make it suitable for devices with low computational and/or memory capabilities [66].

CCN-lite. The CCN-lite tool [67], instead, aims at reducing development (as well as experimental) complexity by providing a lightweight prototype implementation of both CCNx and NDN protocols, tightly integrated with *emulation* and *simulation* environments. CCN-lite codebase, indeed, is made up of about 2000 lines of C code, through which the interoperability between a fully fledged CCNx forwarder and a CCN-lite node is guaranteed (i.e., it is possible to interface with the NDN testbed.) Interestingly, CCN-lite supports both CCNx (both the old version 0.8 and the new one 1.0) and NDN protocols; however, due to its simplicity, only the main components are reproduced, such as ccnb and TLV encoding variants, basic CCN data structures, longest and exact prefix matching, etc. At the same time, other aspects are not covered, like crypto functions, exclusion filters, TCP connectivity, SYNC server, and so on.

As a benchmark of interoperability, CCN-lite supports packet fragmentation and lost packet detection for running the CCNx protocol natively over raw Ethernet. Additionally, the CCN-lite code is portable, as the same code runs unchanged in user and kernel space of both x86 and ARM architectures, or on the Omnet++ simulation platform (using the INET framework). However, in this latter case, only the very basic structures are provided (i.e., nodes and topology initialization, C++ interface to the CCN-lite code), while all the remaining parts need to be implemented from scratch, like content store management, application layer, and so on. The rationale for CCN-lite is surely of great interest for the research community as it provides a unified platform (albeit simplified) for testing and experimentation.

4.2.2. Simulators

NS3 CCNx-DCE. Direct Code Execution (DCE) [68] is a NS3 module that provides facilities to execute, within NS3, existing implementations of user-space and kernel-space network protocols. For instance, the Quagga routing protocol, as well as recent versions of the Linux network stack run under DCE; recently, the support for the CCNx prototype implementation (version 0.6.2) has been added [69]. This is of great help because NS3 features, like debugging, troubleshooting, and so on, greatly simplify the development of the CCNx prototype codebase. However, simplicity and scalability are lost with respect to standard simulators that are described in this section.

CCNPL-Sim. Written in C++, this simulator [70] is based on CBCBsim [23], from which it imports part of the forwarding layer and the Combined Broadcast and Content-Based routing protocol, while the CCN protocol features have been developed from scratch. The simulator has been conceived to assess per-hop forwarding behavior and receiver-based congestion control, where a fine-grained control over individual packets is imperative to get accurate performance results. CCNPL-Sim is the only CCN/NDN simulator to offer out of the box implementations of flow control algorithms, like AIMD, thus representing a natural choice to avoid the burden of an equivalent implementation from scratch [71, 72]. Nevertheless, differently from other simulator-based tools, such as CCNx-DCE and ndnSIM (both based on NS3), or from CCN-lite and ccnSim (both based on Omnet++), CCNPL-Sim has the drawback of using a custom discrete-event simulator. Hence, apart from small-scale studies focused on congestion control, other simulators may by preferable to CCNPL-Sim for wider purposes (for which ndnSIM is a better option) or large scale studies (for which ccnSim is a better fit).

ndnSIM. This NS3 module [73] is part of the NDN tool set. It allows the execution of NS3 simulations (see [74] for the description) by faithfully porting the NFD codebase. In particular, thanks to a recent refactoring of the code, ndnSIM 2.0 introduces some novelties with respect to its previous version; for example, it takes NDN primitives directly from the ndn-cxx library, and it implements forwarding and management operations using the source code of NFD and the NDN-TLV packet format. At the same time, the integration with the NS3 simulator allows an easy modification of the core functionalities, such as the handling of Interest and Data packets, Content Store management, PIT and FIB lookups, and so on; the respective C++ virtual functions, in fact, can be easily overridden by user-defined classes.

In addition, being ndnSIM fully compliant with NS3, it can take advantage from the consolidated modules of NS3, such as its tracing system, as well as its support for multiple physical layers, allowing the simulation of NDN scenarios also in wireless environments. Completeness does not come without cost: for instance, a faithful implementation of structures like PIT, FIB, content names, packet formats, or security functions with packet-level operations may limit the scale of the evaluation due to both processing (CPU seconds) and memory requirements. For example, [75] shows that storing metadata associated to content chunks translates into cache entries which require roughly 0.8 kBytes of RAM memory each; a 10 nodes topology with a content store size of 10^6 elements would require 8 GB of RAM (only considering the memory requirements of the content store). Hence, ndnSIM represents a fairly good compromise between completeness (its main aim) and scalability.

ccnSim. ccnSim is a chunk-level CCN simulator [76], written in C++ under the Omnet++ framework (and thus possibly inherits INET network models), that is described and benchmarked in [77]. The main aim of ccnSim is scalability, allowing to address scenarios with very large CCN content stores (up to 10^6 chunks) and catalog sizes (up to 10^8 files) on off-the-shelf commodity hardware. Scalability is achieved with compromise in the fidelity of, for example, the name space: the memory footprint of individual entries in CCN PIT and Content Stores is 64 bits long (about 2 orders of magnitude smaller with respect to ndnSIM). It focuses on the performance of caching decision/replacement policies and of strategy layer forwarding algorithms: hence, a large number of policies is implemented (e.g., with respect to ndnSIM) to simplify cross-comparison. At the same time, ccnSim neglects routing, security (ndnSIM is a better fit) and congestion control (CCNPL-Sim is a better fit).

5. Comparison methodology

To complement the survey of ICN software tools, as well as their adoption in the ICN literature, in the reminder of this paper we carry out a comprehensive comparison of a relevant set of ICN software. For reasons detailed in the following, we focus on CCN/NDN simulators, and consider fundamental aspects, such as *accuracy* and *consistency* of generated results, *scalability* performance, and *fidelity* with respect to limitations and/or approximations, which could introduce artifacts in the obtained results.

This section describes the methodology that we employ in the cross-comparison: we first illustrate and motivate our selection of software tools (Sec. 5.1), after which we detail steps we undertake to carry out a rigorous investigation of their accuracy (Sec. 5.2), consistency (Sec. 5.3) and fidelity (Sec. 5.2).

5.1. Software selection

The selection of representative elements to be compared is not straightforward: indeed, while a too small set of tools is hardly representative of the full ecosystem, however, a too broad set could be counter-productive (e.g., hard to decide the focus of comparison, practical limits in managing a large number of heterogeneous prototypes, etc.)

From the taxonomy presented in Tab. 1, and by the software description presented in earlier sections, we know that CCN/NDN simulators represent the largest portion of the available ICN software. Thus, it follows that their cross-comparison represents a meaningful starting point in the whole picture of ICN software cross-comparison. It is worth noticing that our aim is not to qualitatively order tools by electing the "best ones", but to check their accuracy, consistency, and fidelity against baseline scenarios, and to provide a taxonomy that could highlight the peculiarities of each tool in order to help users choose the one the better fits their needs. After having routed the comparison towards CCN/NDN simulation tools, the selection process still remains complex, due to the different focus of each simulator – which makes the set of features common to all simulators to be rather limited. In other words, considering all CCN/NDN simulators would significantly restrict the boundaries of the possible investigation, as it is difficult to find implementations of the same algorithm –if not for simple cache replacement and decision policies– across all simulators.

Table 2: Main features of selected simulators.

	RP$^{\odot}$	**DP**†	**FS***	**W**$^{\square}$	**FC**$^{\diamond}$	**Workload**‡	**TR**$^{\triangle}$	**Main focus**
ndnSIM	RND FIFO **LRU** LFU	**LCE** **FIX**	**SP** BR FLD NCC Manual	-	-	Synthetic: **CBR+ZM** **Unif+ZM** **Exp+ZM** C-Batches C-Window	A R	Completeness
ccnSim	RND FIFO **LRU** LFU	**LCE** **FIX** LCD BTW PC	**SP** FLD NRR Manual Random Parallel	A	-	Synthetic: **CBR+ZM** **Unif+ZM** **Exp+ZM**	.ned files	Scalability
Icarus	RND FIFO **LRU** LFU	**LCE** **FIX** LCD BTW PC HR	**SP** NRR HR	F	-	Synthetic: **Exp+ZM** Tracedriven: GlobTraff Squid Wikibench	FNSS	Scalability
CCNPL Sim	RND FIFO **LRU**	**LCE**	**SP** Manual Random Balance	-	AIMD: ICP FIX or RAQM	Synthetic: **CBR+ZM** **Exp+ZM**	.brite files	Network congestion

Note: Features in bold are available in all simulators.
$^{\odot}$ *Replacement Policy (RP)*: RND=random, LRU=Least Recently Used
LFU=Least Frequently Used
† *Decision Policy (DP)*: LCE=Leave Copy Everywhere, FIX=Fixed probability
LCD=Leave Copy Down, BTW=Betweenness Centrality
PC=Probabilistic caching [78], HR=HashRouting [79]
* *Forwarding Strategy (FS)*: SP=ShortestPath, BR=BestRouting, FLD=Flooding
NRR=NearestReplicaRouting [80], NCC=CCNx v0.7.2 strategy
HR-variants=HashRouting [79]
$^{\square}$ *Warmup (W)*: A=Adaptive, F=Fixed
$^{\diamond}$ *Flow Control (FC)*: AIMD=Additive Increase Multiplicative Decrease
ICP=Interest Control Protocol, FIX=Fixed
RAQM=Rate-based Active Queue Management
‡ *Workload*: CBR=Constant Bit Rate, Unif=Uniform, Exp=Exponential, ZM=Zipf-Mandelbrot
$^{\triangle}$ *Topology Reader (TR)*: A=Annotated, R=Rocketfuel, FNSS=Fast Network Simulation Setup

Therefore, this paper focuses on a subset of the available simulators tools, excluding those based on prototype codebase (i.e., NS3 CCNx-DCE and CCN-lite), that would instead require a significant additional effort to be included in the comparison (e.g., the implementation of a LRU replacement strategy for NS3 CCNx-DCE, or the complete development of content store management and application layer for CCN-lite in its Omnet++ adaptation, to name a few).

Accordingly, four simulators are selected, that are: *ndnSIM*, *ccnSim*, *CCNPL-Sim*, which explicitly target the NDN architecture, and *Icarus*, which, despite being a simulator for general ICN networks, can be fairly adopted to execute NDN simulations too. Indeed, due to its generality and its main aim of assessing performance of networks of caches (similar to *ccnSim*), it makes sense to include it in the cross-comparison. A detailed description of their features is reported in Tab. 2, where commonly implemented decision policies and forwarding strategies are reported in bold. In particular, we compared the selected softwares in their latest versions: ndnSIM v2.0, ccnSim v3.0, CCNPL-Sim v1.0, and Icarus v0.5. Fig.3 represents, as a Sankey diagram, the ICN software tools reported in Tab. 1, highlighting the four selected simulators.

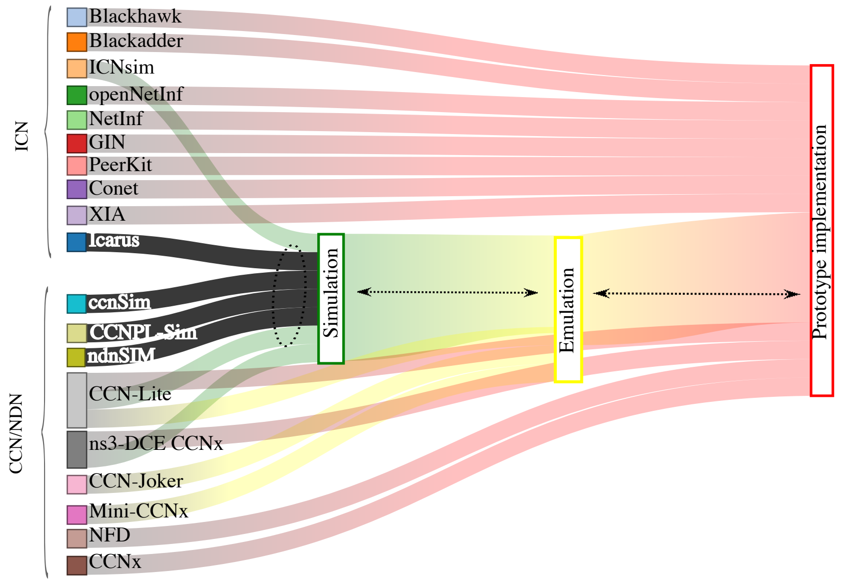


Figure 3: Sankey diagram of the ICN and CCN/NDN software ecosystems, highlighting simulators that are cross-compared in this paper.

For the sake of clarity, we define the adopted terminology in the following. A *Configuration* ($\mathcal{C}$) identifies a set of algorithms to be used in a CCN/NDN network (e.g., routing and forwarding strategies, caching decisions and replacement policies), and that are available in one or more *Simulators* ($\mathcal{S}$). Finally, a *Network scenario* ($\mathcal{N}$), instead, denotes the set of exogenous factors (e.g., catalog skewness, workload, network topology, number of simulated events, etc.) under which the strategy is simulated. Therefore, performance of metric X, gathered under CCN configuration c, with network scenario n and simulator s, can be denoted as $X^{c,n,s}$.

According to this terminology, notice that while each simulator provides multiple configurations $\mathcal{C}$ (i.e., different combinations of parameters listed in Tab. 2), there are basically two configurations common to all simulators (namely, shortest path (SP) routing with LRU replacement and either LCE or FIX cache decision policies), with the exception of CCNPL-SIM, which does not implement a FIX cache decision policy. It follows that steps need to be undertaken to ensure a sound, fair (i.e., apple-to-apple), but also rich comparison, which we describe next.

5.2. Accuracy

Given that a very restricted set of features is shared between the four selected simulators, as reported in Tab. 2, and that the use of a theoretical model is advisable as common reference point for the accuracy comparison, a first analysis is focused on the evaluation of the hit probability in two simple scenarios: *single* cache and a *tandem network*, reported in Fig. 4. Note that the main goal of this work is not the analysis of caching performance. Yet, testing the accuracy of the simulators is straightforward in this particular case, due to the presence of knowingly very accurate analytical models, such as Che's approximation for a single cache with Least Recently Used (LRU) replacement policy [81]. Indeed, with a solid theoretical base it is possible to define specific metrics to (i) quantify the accuracy of each simulator and to (ii) express guidelines that could emerge from this cross-comparison. Following the terminology introduced above, the aim of this phase is to check if $p_{hit}^{c,n,s}$, with $s \in \{ndnSIM, ccnSim, CCNPL-Sim, Icarus\}$, is accurate with $p_{hit}^{c,n,t}$, where t denotes the theoretical model.

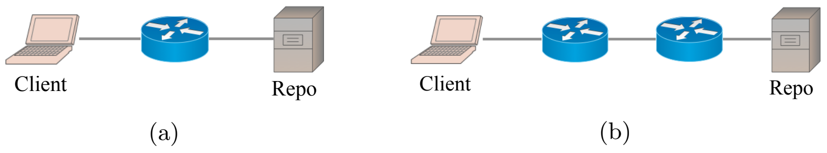


Figure 4: Sample Topologies: (a) Single Cache; (b) Tandem Network.

5.2.1. Reference models

In a single cache scenario with LRU replacement, Che's approximation [81] is known to be extremely accurate. Specifically, denoting with M the cardinality of the catalog, with B the cache size, P_{H_i} the hit probability for the content of rank i, $\lambda_i = \Lambda p_i$ the mean inter-arrival rate of requests for the content i, and p_i its request probability, T_C the *characteristic time* of the specific cache (i.e., the maximum time interval between two content requests without a cache miss), it results that:

$$P_{H_i} = 1 - e^{-\lambda_i T_C}; \qquad \sum_{i=1}^{M} \left(1 - e^{-\lambda_i T_C}\right) = B. \tag{1}$$

While Che's approximation applies to the single cache scenario due to the validity of the Independent Reference Model (IRM) model, its extension to general cache networks is more challenging. Focusing on the tandem network considered in the second scenario of the accuracy comparison, the "classic" approach is to approximate P_H at the second cache by applying the IRM model on a request arrival process filtered by hits on the first cache, i.e., $\lambda_{i,2} = \lambda_{i,1}\left(1 - P_{H_{i,1}}\right)$. Yet, this extension is known to overestimate the cache hits at the second cache; indeed, several "refined" models have been proposed so far. In this work, a recent and accurate proposal [82] is considered, which expresses the hit probability at the second cache as:

$$P_{H_{i,2}} \approx 1 - e^{-\lambda_{i,2}(T_{C_2} - T_{C_1})}, \tag{2}$$

where T_{C_1} and T_{C_2} are the characteristic times of the first and the second cache, respectively (the reader is referred to [82] for technical details).

5.2.2. *Metric definition*

Precise metrics need to be defined in order to quantify discrepancies between simulation and modeling results, simplify the cross-comparison task, and provide a systematic analysis of the simulation accuracy with respect to important scenario parameters. Considering a generic content of rank i, integral errors ξ_i^H on the catalog head (ID$\in[1,i]$) and ξ_i^T on the catalog tail (ID$\in[i,M]$) are defined as:

$$\xi_i^H = \frac{1}{i} \sum_{j=1}^{i} \left| \frac{P_{H_j} - \widetilde{P}_{H_j}}{P_{H_j}} \right| \tag{3}$$

$$\xi_i^T = \frac{1}{M-(i-1)} \sum_{j=i}^{M} \left| \frac{P_{H_j} - \widetilde{P}_{H_j}}{P_{H_j}} \right| \tag{4}$$

where P_{H_j} refers to Che's approximation, considered as the real value used to compare the hit probability $\widetilde{P}_{H_j}$ gathered by the simulators.

Intuitively, the head (tail) error for the i-th object represents the average error over the whole set of objects that are more (less) popular than the object having rank i-th. For instance, ξ_m^H represents the average error for the m most popular objects in the catalog, accounting for a certain percentage of requests. Respectively, ξ_m^T denotes the integral error for the $M-m$ less popular ones.

5.2.3. *Tracing system*

To compute the above metrics, we additionally point out that each simulator provides its own tracing system to collect different metrics (such as hit probability, hit distance, file download time, network load, etc). These metrics are, however, logged with different levels of details (e.g., some provide per-content statistic, while others provide only global measurements; some provide only averages, while others provide also higher moments). Tools differ, also, in their way of handling the simulation transient period: not all simulators test for convergence of the metrics under observation, and the implementation anyway differ when present.

Therefore, to avoid any bias due to the way metrics are computed in each simulator, we implemented the same logging system on all the simulators, which traces object-level hit and miss events at every node. We point out that we rely on our common tracing system only to perform accuracy assessment, whereas we employ the standard tracing system to perform consistency assessment.

5.3. *Consistency and Scalability*

If simple scenarios are useful to evaluate accuracy with respect to a known reference (i.e., theoretic model in our case), they are hardly representative of the real settings ICN networks are expected to operate in. As a side effect of the consistency comparison, it is also useful to evaluate the scalability of the simulators in larger scale scenarios. Therefore, we need to widen our focus to include more complex scenarios, in terms of both networking and algorithmic features. As a direct proof of the scalability of the tested tools, we monitor two performance indices, that are *execution time*, expressed in CPU seconds, and *memory requirements*.

As thoroughly discussed in [11], a reference scenario (both topology and traffic load) that could be used as a ground truth to easily evaluate all the aspects of the ICN paradigm still lacks. Both topology and traffic load depends on the type of study that needs to be carried out; for example, a Web-scale ICN simulation would require a topology of 45k ASes and 200k links, and a content catalog with cardinality 10^{12} [11], which would pose serious limits to the scalability of the adopted tool and to the feasibility of the evaluation on commodity hardware. As a consequence, we consider a Video On Demand (VoD) like scenario (i.e., with catalog cardinality of 10^4 contents [11]) which permits us to have a fair assessment of the scalability features of the compared tools, and also to execute the evaluation on the available commodity hardware.

The same scenarios are used for the consistency verification, where we resort in mutual comparison of the results from the the different simulators gathered in the same settings. By cross-comparing their results, the aim is to confirm the consistency, or to identify with more precision the boundaries within which we can expect simulation results to be in agreement. It is worth clarifying that we purposely kept out CCNPL-Sim from this more complex comparison (i.e., it will be based on a topology with 100 nodes) due to its lack of scalability, which will be described in the following (it was also necessary to implement a fixed probabilistic decision policy from scratch, with the risk of introducing some inefficiency/inconsistency).

As we have previously outlined, the simulators share a limited set of common features, on which we base our consistency evaluation (albeit they offer a rich set of features individually). More formally, we compare the performance that the same CCN configuration $c \in \mathcal{C}$, with an identical networking scenario $n \in \mathcal{N}$, has in different simulators $s \in \mathcal{S}$ (i.e., $s \in$\{ndnSIM, ccnSim, Icarus\}). That is, the aim of this section is to compare $X^{c,n,s}$, $X^{c,n,s'}$, $X^{c,n,s''}$ for different c. For details about the comparison of heterogeneous strategies implemented in different simulators we refer the interested reader to our preliminary work [5].

5.4. Fidelity

The aim of the last step of the comparison is to raise awareness of the need for a careful design of the scenario, to avoid simulation results to be affected from artifacts that are due to underlying limitations of the simulator in use.

A very simple, but very telling, example concerns, for instance, the presence of a complete link model, able to simulate link capacity, delays and transmission buffers scheduling. Unlike ndnSIM and CCNPL-Sim, where packet-level fidelity is enabled by default, Icarus and ccnSim simulate links with infinite capacity by default (albeit capacity limitations can be enforced by standard modules provided by FNSS and Omnetpp), and they assume to operate at a load below congestion. It follows that, unlike ndnSIM and CCNPL-Sim, Icarus and ccnSim by default neglect phenomena like congestion and packet losses. While simulators properly advertise their capabilities, it is possible that scenarios are engineered in such a way that the Icarus and ccnSim tools are used at an operational point they do not assume (e.g., close to congestion or in overload).

It is thus relevant to consider scenarios where access congestion can lead to drop of Data and Interest packets (e.g., the single cache case). Clearly, we expect packet losses to impact simulation dynamics: indeed, the loss of an Interest packet implicitly changes the object popularity, hence the arrival process at the cache; the loss of a Data packet, instead, affects LRU decisions. Both ultimately affect the content hit probability, which can be gauged via the ξ_H and ξ_T metrics.

6. Accuracy comparison

6.1. Evaluation scenarios

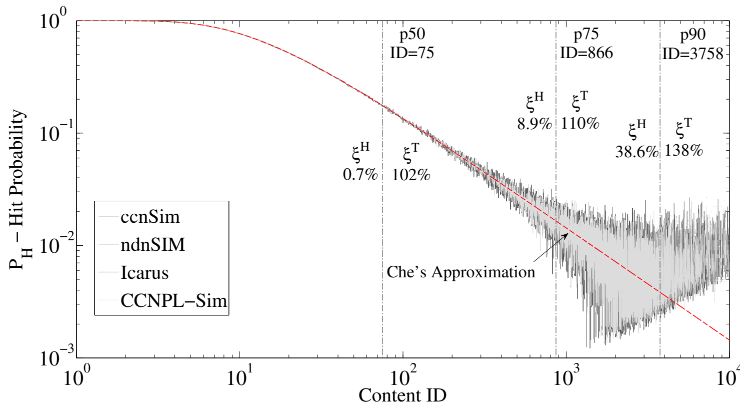
The first scenario models a single CCN router directly connected to a repository. The router receives requests from a set of clients, and forwards them to the repository in case of a cache miss. The second scenario, instead, comprises an additional CCN router, thus configuring a tandem cache network. For the sake of simplicity, we consider, for the time being, Least Recently Used (LRU) replacement and Leave a Copy Everywhere (LCE) decision policies (so that contents replicate in every traversed cache).

Given that the *relative* accuracy of all tools is the main focus, as opposed to the *absolute* value of performance indicators, it follows that the precise value of some parameters (like the content catalog cardinality, M, the cache size to catalog ratio, C, and the request rate λ), are not of primary importance. A tuple (namely, $M = 10^4$, $C = 0.01$, and $\lambda = 4$ request/s) is, therefore, selected for both scenarios, which allows to perform a fair cross-comparison of the selected simulators, at a furthermore manageable scale.

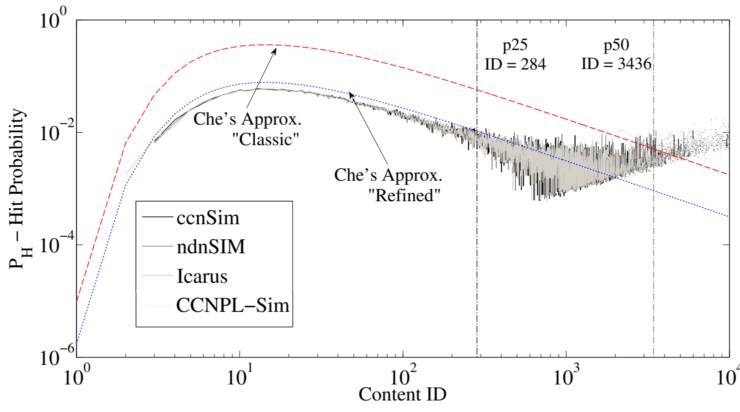
On the other hand, parameters that can significantly influence the dynamics of the simulated scenarios, as well as the accuracy of the measured metrics, are of primary importance: in this case, simulations with multiple values are executed. For instance, the ICN literature, with few exceptions, models content popularity as a Zipf's probability distribution, that is $P(X = i) = i^{-\alpha}/\sum_{j=1}^{M} j^{-\alpha}$, of which three distinct settings (namely, $\alpha \in \{0.8, 1.0, 1.2\}$) are considered. Similarly, the number of clients' requests, R, heavily impacts on the statistical significance of the collected metrics, of which disparate settings (specifically, $R \in \{10^6, 10^7, 10^8\}$) are considered.

6.2. Simulation results

Fig. 5 reports, as an example, results related to the content hit probability obtained for $\alpha = 1$ and $R = 10^7$, averaging 10 runs for each simulator, for the single cache (top) and the second cache of the tandem network (bottom). The correspondent theoretical models are reported as dashed lines and clearly



(a)



(b)

Figure 5: Experimental vs Theoretical Hit Probability in two simple scenarios with $M = 10^4$, $\alpha = 1$, and $R = 10^7$: (a) Single Cache; (b) Second Cache of Tandem Network.

labeled. Each picture further marks the content IDs corresponding to specific percentiles of the overall requests: e.g., requests for content whose ID$\in[1, 75]$ (ID$\in[1, 3436]$) account for 50% of the requests in the single cache (tandem) case.

Observing Fig. 5, the need for defining a compact, yet insightful, metric to compare the accuracy of the three simulators, clearly emerges. Indeed, it can be noticed that: (i) curves of the different simulators overlap, so that it is hard to precisely assess differences between them and/or discern the respective curves; (ii) simulation results are, as expected, unreliable in gauging rare events: this appears both as noise towards the tail of the catalog, in both scenarios, as well as absence of results in the head of the catalog for the second cache (due to the filtering effect of the first cache).

More precisely, several regions can be identified in the plots, where either modeling results, or simulation results, can be considered as more reliable. In the single cache scenario, reported in Fig. 5-(a), it emerges that: (i) simulation and modeling results are in perfect agreement up to the 50th percentile; (ii) variance in simulation results is tolerable and unbiased (i.e., symmetric noise) up to the 75th percentile; (iii) above the 75th percentile, simulations tend to overestimate hit probability; this is due to the fact that requests for those contents represent rare events for the whole simulation, being them equal to $R_i = R \cdot p_{Hi}$, where p_{Hi} is the Zipf's probability of content i, and R_i the associated number of requests. This phenomenon is more evident especially for small cache regimes, since the cache hit is conditionally reported only for contents that were found in the cache. As the total number of requests R increases, network caches will converge more and more to a point where only the most popular contents are cached, thus increasing the proportion between R_i and p_{Mi} for unpopular contents, where p_{Mi} is the *miss* probability for content i. This will result in a closer estimation of the empirical hit probability with respect to the theoretical one, as it is confirmed by the sensitivity analysis presented in the next section.

Similarly, in the second cache of the tandem network, reported in Fig. 5-(b), it appears that: (i) results from modeling capture rare events more accurately, in both the head and the tail; (ii) the tail significantly shifts due to the filtering effect contributed by the first cache, so that significant noise and bias affect simulation results, already between the 25th and 50th percentile of the requests process, albeit (iii) simplifying assumptions make modeling results less accurate in the body of requests up to the 25th percentile, which is especially evident for the classic extension of Che's approximation, and that is, instead, significantly corrected (but not entirely eliminated) by its refinement [82].

The defined metrics (Sec. 5.2.2) permit to have quantifiable values that reflect the aforementioned observations. Fig. 5-(a) reports values of both head and tail integral errors for some significant percentiles; for instance, $\xi^H_{866} = 8.9\%$ represents the average error for the 866 most popular objects in the catalog accounting for 75% of the requests, while $\xi^T_{866} = 110\%$ quantifies the average error for the remaining 25% of requests. As a typical user will both request popular and unpopular contents, it follows that measured performance could be, in some cases, significantly distorted.

6.3. Sensitivity analysis

Several simulations are executed, experimenting with different scenarios[7], α, and R values, from which it emerges that the four simulators consistently report very similar ξ^H and ξ^T statistics over all tested scenarios. For the sake of example, Tab. 3 reports the mean and standard deviation of the head and

[7]Notice that, since in more complex scenarios (such as the tandem network) simulations are more reliable than analytical models for some regimes, the ξ^H and ξ^T quantify discrepancies with the refined Che's approximation, more than simulation errors. Yet, these metrics are still helpful in quantifying whether these discrepancies are consistent across simulators, which is among our primary goals.

Table 3: Head and Tail Integral Errors ($\mu \pm \sigma$) for selected simulators in the single cache scenario: $M = 10^4$, α =1, $R = 10^7$.

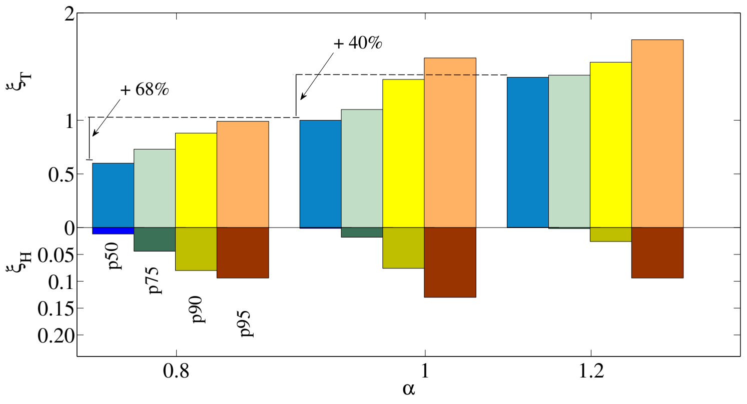
	p50		p90	
	ξ^H[%]	ξ^T[%]	ξ^H[%]	ξ^T[%]
ccnSim	0.7 ± 0.1	101.5 ± 1	38.6 ± 1	138.2 ± 1
ndnSIM	0.7 ± 0.1	101.8 ± 1	38.6 ± 0	138.6 ± 1
Icarus	0.7 ± 0.1	101.4 ± 1	38.8 ± 1	138.0 ± 1
CCNPL-Sim	0.8 ± 0.1	101.5 ± 1	39.1 ± 0	137.8 ± 1
α=0.8	6 ± 0.2	60.8 ± 1	40 ± 1	88.5 ± 1
α=1	0.7 ± 0.1	101.8 ± 1	38.6 ± 0	138.6 ± 1
α=1.2	0.03 ± 0	141.4 ± 4	13.3 ± 0	154.5 ± 5
$R = 10^6$	2.2 ± 0.3	159 ± 3	110.6 ± 1.5	186.3 ± 4
$R = 10^7$	0.7 ± 0.1	101.8 ± 1	38.6 ± 0	138.6 ± 1
$R = 10^8$	0.3 ± 0	33.2 ± 0.2	12.3 ± 0.2	45.3 ± 0.4

tail errors obtained from the single cache scenario with $\alpha = 1$ and $R = 10^7$, for a few percentiles.

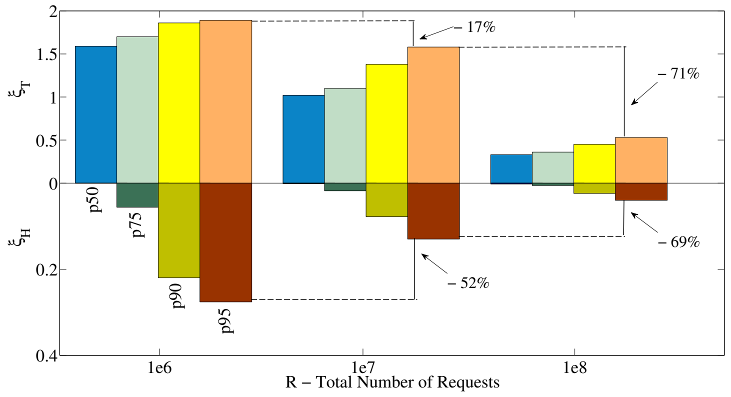
Since, at least in this scenario, the four simulators yield identical results, for practical purposes, one of them is selected (namely, ndnSIM) to perform a sensitivity analysis of the error as a function of the input parameters α and R. In particular, either (a) the number of requests $R = 10^7$ is fixed and the Zipf's exponent $\alpha \in \{0.8, 1, 1.2\}$ is varied, or (b) $\alpha = 1$ is fixed and $R \in \{10^6, 10^7, 10^8\}$ is varied. Results are both tabulated in the bottom part of Tab. 3, and graphically reported in Fig. 6. Note that in Fig. 6, the bottom part of the y axes has a different scale with respect to the upper one.

An interesting dual phenomenon emerges: for increasing popularity skew, fewer more popular objects make the ξ^H error smaller, at the expense of a degradation of ξ^T accuracy for the heavier tail (relative ξ^H improvement and ξ^T degradations are annotated in the picture for the sake of clarity). For example, the ξ^H error associated to the 50^{th} percentile is almost equal to zero for $alpha = 1.2$, whereas the respective ξ^T is equal to 141%, reflecting the characteristic long tail of the Zipf's probability distribution for $\alpha > 1$. The influence of the tail is further confirmed by the relative increment of the ξ^T when increasing α: indeed, from $\alpha = 0.8$ to $\alpha = 1$, the ξ^T of the 50^{th} percentile increases by 68%, whereas, from $\alpha = 1$ to $\alpha = 1.2$, there is an increment of almost 40%.

Moreover, a clear phenomenon can be observed from Fig. 6-(b): the number of simulated events has a remarkable impact on the accuracy of the measured metrics. Indeed, the decrement for both the ξ^T and the ξ^H is evident: focusing on the integral errors associated to the 95^{th} percentile, for example, a reduction of 52% (17%) and 69% (71%) of the ξ^H (ξ^T) can be seen when passing from $R = 10^6$ to $R = 10^7$, and from $R = 10^7$ to $R = 10^8$, respectively.



(a)



(b)

Figure 6: Head and Tail Integral Errors for the single cache scenario: (a) $M = 10^4$, $R = 10^7$, and α variable; the relative ξ^T increments for the 50^{th} percentile of about 68% and 40%, when shifting from $\alpha = 0.8$ to $\alpha = 1$, and then to $\alpha = 1.2$, respectively, confirm the influence of the tail. (b) $M = 10^4$, $\alpha = 1$, and R variable; in this case the number of simulated events is the relevant factor: when passing from $R = 10^6$ to $R = 10^7$, and from $R = 10^7$ to $R = 10^8$, the integral errors associated to the 95^{th} percentile experience a reduction of 52% (17%) and 69% (71%).

Our evaluation is not exhaustive enough to provide precise and rigid guidelines regarding the number of requests that need to be simulated according to the considered scenario; what, instead, emerges clearly is the interaction and influence that both the catalog skew and the number of requests have one the accuracy of the produced results. In the reported case of the single cache scenario, with a fixed $\alpha = 1$, simulation results where accurate only for the catalog body (with a ξ^H<1%) due to the skew in the object popularity, despite $R = 10^7$ corresponds to an average of 1000 requests per object. It follows that with skewed distributions, a total number of simulated events of two orders of magnitude bigger than the catalog cardinality is considered as a lower bound to obtain consistent results also for the heavier tail of the catalog. Our hope is that the presented study can encourage researchers in presenting their results with all the necessary information needed to evaluated their scientific soundness, like total number of simulated requests with respect to the catalog cardinality, total number of simulated runs for each scenario, and so on.

6.4. Summary of findings

The conducted analysis in simple scenarios allows to conclude that (i) CCN-/NDN simulators yield remarkably consistent results, (ii) accuracy of results gathered for different skews is highly variable for a fixed request budget, (iii) increasing the request budget, accuracy in the tail of the catalog is achievable as well. General guidelines can be tabulated to express simulation parameters (such as number of requests R) as a function of the scenario (e.g., catalog size C and skew α), required to bound head ξ^H and tail ξ^T errors below some arbitrary thresholds. Yet, in the process of defining and agreeing on common scenarios (e.g., which is in progress at IRTF ICNRG [12]), defaults settings of these crucial parameters (or guidelines for non-default settings) should be jointly included and considered by the ICN community, as they are of primary importance for a scientifically sound evaluation.

7. Consistency and scalability comparison

7.1. Evaluation scenarios

We now widen the focus to verify cross-consistency of simulation results over more complex and larger scale scenarios. Differently from the accuracy assessment, consistency results are based on the default tracing systems of each simulator. As previously described (Sec. 5.3), we consider *homogeneous* scenarios, where exactly the same algorithmic configuration $\mathcal{C}$ is tested over the same network scenario $\mathcal{N}$ implemented over all simulators, in order to ensure soundness of our findings.

Specifically, we fix the well known LRU replacement policy in every scenario, and consider variations of cache admission policy and Interest forwarding strategies. As far as cache admission policies are concerned, we use either (i) Leave Copy Everywhere (LCE), where copies of retrieved contents are systematically cached in every traversed node along the reverse path, or (ii) FIX [83], where contents are cached with a fixed probability P.

As far as forwarding strategies are concerned, we use Shortest Path (SP), where a control plane routing protocol (like Named-data Link State Routing Protocol (NLSR) [84]) is assumed to proactively disseminate name-level reachability to build FIBs of nodes,

Regarding the networking scenario, a 10x10 grid topology is considered, where a single repository, randomly placed in each of the 10 simulated runs, stores the entire catalog of $M = 10^4$ contents. 30 clients are randomly placed for each run; they issue requests for contents characterized by a Zipf's probability distribution with exponent $\alpha = 1$. The considered cache to catalog ratio is $C = 0.005$, and the number of simulated events is $R = 10^7$. We make all results shown in this section, that were already previously published in our preliminary work [5], interactively browsable at [85].

7.2. Homogeneous comparison

As previously stated, we purposely do not consider CCNPL-Sim for its scalability issues (shown in the next section) and for its lack of a fixed probabilistic cache decision policy, as emerges from Tab. 2. As a consequence, in this phase two configurations that are commonly implemented in all the three simulators (i.e., ndnSIM, ccnSim, and Icarus) are selected; in particular, as emerges from Tab. 2, SP with LCE and LRU (c_1), and SP with FIX and LRU (c_2) are the ones implemented in all simulators.

More precisely, in accordance to the terminology defined above, the aim of this section consists in comparing $X^{c,n,s}$, $X^{c,n,s'}$, and $X^{c,n,s''}$ for different c. As performance metrics, the *hit ratio*, the *hit distance* (that relates to the quality of ICN performance), and the *network load* (that relates to the cost for ISP, and is calculated by counting the number of generated *Interest* and *Data* packets in the whole network) are considered.

Results are reported in Fig. 7, where multiple metrics X are reported at the same time by using a parallel coordinates graph, where each metric is normalized to its respective observed maximum value (i.e., $X^{c,n,s}/\max(X^{c,n,s})$, so that a hit ratio of 1 does not imply that all contents are cached).

As it can be seen clearly from the picture, two separate families of curves emerge, which pertain to a different configuration (i.e., c_1 vs c_2, respectively). Results in Fig. 7 are expected, since it is known that a probabilistic caching decision (FIX) performs better than systematically admitting object in the cache (as in LCE) [82, 86]. As such, the most interesting observation regarding Fig. 7 is that, for any given configuration, curves are very similar, regardless of the simulator they have been obtained from. Hence, all the simulators provide very consistent results, despite heterogeneous implementations, codebases, and tracing systems. It is worth to notice that the little variations among the metrics gathered from different simulators (i.e., curves in Fig. 7 do not perfectly overlap each other) are due to the randomness introduced in each simulated run; that is, each simulator, by means of its random number generators, vary, per each run,

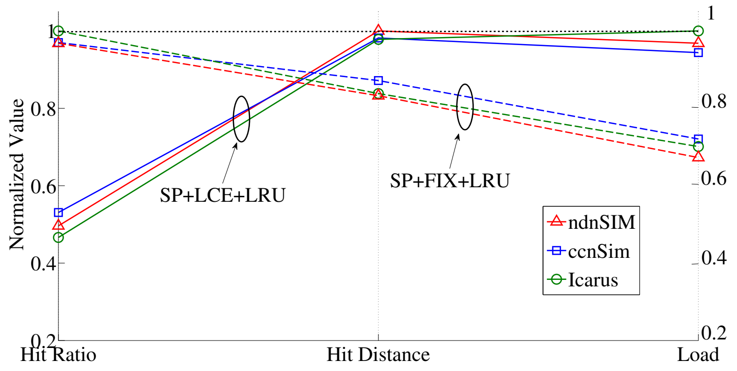


Figure 7: Homogeneous comparison of the selected simulators through shared configurations, i.e., c_1=SP+LCE+LRU and c_2=SP+FIX+LRU, for a grid topology of 100 nodes, $\alpha = 1$, $R = 10^7$, and $C = 0.005$.

both the placement of the 30 clients inside the grid, and the request pattern issued by each client. Therefore, the average of 10 different runs generates only small variations between the results provided by the compared simulators. This represents a comforting result, considering the disagreement experienced in other networking areas [87, 88] (that could be tied to several reasons, such as bugs in the software implementation, poor entropy of the random number generator, etc).

7.3. Scalability comparison

Both the naive scenario used for the accuracy comparison, and the more complex ones of the consistency comparison have been used as a reference to extract significant performance indexes, like CPU and memory usage. In particular, mean values and standard deviations, reported in Tab. 4, are obtained for the single cache scenario and for the grid topology when executing the SP+LCE configuration. To avoid any bias, all the simulations have been performed using the same machine (8-core Intel Xeon at 3.6 GHz, equipped with 32 GBytes of RAM, and with Ubuntu Linux 12.04).

It is worth noticing that the CCNPL-Sim simulator has a different design with respect to the others; indeed, the simulation consists in two parts: the first one is the workload (WL) generation, where all the events (like content publications or content requests) are generated and logged inside a text file; the second one, instead, is the actual simulation (SIM), where the previously created workload is taken as an input to execute the simulation and to produce the desired output.

It follows that two different measures are reported in Tab. 4 for the CCNPL-Sim, for which scalability problems emerge even for the single cache scenario. It clearly appears that the limitation of CCNPL-Sim regards the CPU time: it takes almost 20 minutes only to create a workload file, which has an average size

Table 4: CPU and Memory Usage ($\mu \pm \sigma$) for two scenarios: Single Cache ($M = 10^4$, $R = 10^7$, $\alpha = 1$, $C = 0.01$), and Large Scenario (100 nodes, $M = 10^4$, $R = 10^7$, α =1, $C = 0.005$).

		Single Cache		Large Scenario	
		CPU [s]	Mem [MB]	CPU [s]	Mem [MB]
ccnSim		118 ± 1	12 ± 1	489 ± 41	54 ± 1
ndnSIM		377 ± 2	68 ± 2	2634 ± 304	68 ± 3
Icarus		261 ± 1	51 ± 1	557 ± 44	55 ± 2
CCNPL-Sim	WL	1202 ± 0.4	10 ± 1	–	–
	SIM	686 ± 94	14 ± 1	–	–

of 710 MBytes. Considering that the scenario is naive, and that this procedure needs to be executed for each run, it follows that CCNPL-Sim is not suitable for large scale scenarios, having its focus on packet-level simulations for congestion control studies.

Tab. 4 reports, also, measures taken from the execution of the large scale scenario used for the consistency comparison. It clearly shows that scalability is a strength of ccnSim and Icarus simulators, which present a mean execution time of 489 and 557 seconds, and a memory usage of 54 and 55 MBytes, respectively. These results confirm their main focus of assessing performance of different strategies and algorithms even for large scale network of caches. In the end, from Tab. 4, it is possible to verify that ndnSIM is conceived for a different scope with respect to ccnSim and Icarus; indeed, its main aim of completeness and fidelity with respect to the real NFD protocol and daemon results in more complex data structures and procedures that slow down its execution time, which is not comparable with respect to ccnSim and Icarus. Again, our purpose is not to present an unfair comparison, but to make the reader even more aware in the selection process of the simulator according to his/her needs (e.g., fair reproduction of the NDN protocol or large scale performance study of networks of caches). What is remarkable, however, is that despite its complexity, ndnSIM 2.0 presents a memory usage of only 68 MBytes, which is very close to the one of ccnSim and Icarus.

7.4. Summary of findings

The investigation carried out in this section shows simulation results gathered from homogeneous CCN/NDN configurations to be in agreement across different simulators. In other words, we conclude that available open source CCN/NDN simulators yield comparable results on different, but accurate, implementations of the same configuration, and that scenarios are thus well calibrated across simulators.

Finally, we find that fairly large scale simulations are possible on off-the-shelf hardware, although the specific statistics concerning execution time and memory requirement are clearly tied to the level of abstraction of the simulation.

8. Fidelity

8.1. Evaluation scenarios

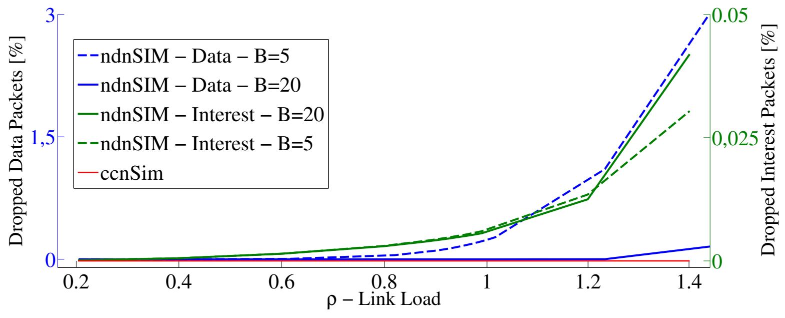
As simulation models differ in their respective levels of abstraction, it is important to assess whether a wrong design of the simulated scenario could lead to distorted results. An obvious, but telling, example concerns link capacities: while simulators, like ndnSIM and CCNPL-Sim, trade scalability off for fidelity by considering an accurate link model by default (including link capacity, delays and transmission buffers), others, like Icarus and ccnSim, make the opposite choice to ameliorate scalability at the expense of fidelity by assuming ideal link models by default (with infinite capacity), and expect users to explicitly include additional network-related components via NetworkX/FNSS or Omnetpp/INET, respectively. While this latter choice is openly advertised, the risk is that careless use of such tools in operational regimes they are not expected to run, could lead to wrong or distorted estimation of network performance.

For the sake of illustration, we consider a single cache scenario, in which the request arrival rate is dimensioned so that the respective data arrival rate can significantly exceed the link capacity: the main purpose is, indeed, not the realism of the scenario, as the workload rather represents a stress test. Clearly, while the capacity parameter is taken into account in ndnSIM (or CCNPL-Sim), it is ignored in ccnSim (or Icarus). Due to the same operational choices made by *ccnSim* and *Icarus* from one side, and from *ndnSIM* and *CCNPL-Sim* of the other side, only *ccnSim* is considered for the comparison against *ndnSIM* in this section.

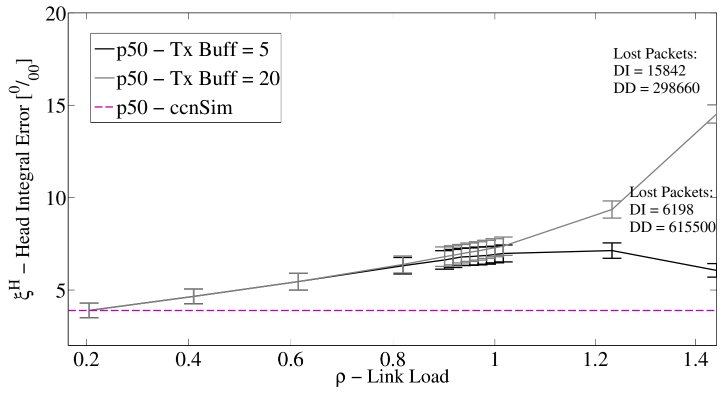
Fixing $\alpha = 1$ and $R = 10^7$ (that yield accurate results as reported in Sec. 6.3), link capacity is set to $Bw = 1\,Mbps$, link delay to $d = 10\,ms$, while two sizes of the link buffer $B \in \{5, 20\}$ packets are considered to possibly expose different loss patterns. Requests follow a Poisson process with average inter-arrival time $\lambda \in [25, 180]$ Interest/s, that, considering an average data size of $L = 1kB$, corresponds to a link load $\rho = \lambda \cdot L/Bw$ varying in the range $\rho \in [0.2, 1.4]$. For each value of buffer size B and λ, mean values are reported in Fig. 8, along with their 95% confidence intervals, by averaging ten different simulation runs. Furthermore, retransmissions are disabled in ndnSIM.

8.2. Simulation results

For each load ρ and buffer size B, the number of dropped Data and Interest packets is reported in Fig. 8-(a), while the ξ^H for the $50th$ percentile is reported in Fig. 8-(b) to assess the corresponding discrepancies with respect to Che's approximation in the lossless case. Observe that, for load $\rho < 1$, losses due to transient congestion phenomena imply no noticeable impact on the accuracy, validating the choice made by ccnSim and Icarus of preferring scalability in this operational regime. Conversely, for $\rho > 1$ (which can happen because of an incorrect dimensioning of the network, for example), non-trivial phenomena arise, with furthermore opposite effects depending on the buffer size parameter, that are captured by ndnSIM, while silently ignored by ccnSim.



(a)



(b)

Figure 8: Single Cache scenario - link with Bw = 1Mbps and d = 10ms, B = 5, 20: (a) Dropped Data and Interest packets; (b) ξ^H for the 50th percentile.

Aside the underlying reasons of the observed dynamics[8], what is worth to remark here is that, with complex scenarios and larger topologies, a wrong design of the network scenario could produce significant distortions in the evaluation of the system performance (e.g., considering the hit ratio).

While in the current state of affairs the burden of carefully engineering the simulation scenario to avoid inconsistent results is left to users (e.g., as ccnSim and Icarus results should be corrected by scaling down request arrival rate), a better practice would be to automate and generalize consistency check, to minimize sources of human errors in the scenario design, that possibly lead to

[8]Specifically, when the buffer is small and the load high, losses of Data and Interest packets are uncorrelated, so that while Data loss impact LRU timers, Interest loss is independently distributed over all contents, which has not impact on the content popularity. Large buffers, instead, correlate losses, and more prominent Interest losses significantly affect the arrival process, which explains larger discrepancies with respect to Che's approximation.

inference of wrong conclusions. For this specific example, ccnSim and Icarus could detect inconsistence, without incurring in the scalability penalty of fully simulating link capacities, by evaluating the data load over arbitrary windows of time, and comparing it against the configured user capacity (e.g., as done by Mini-HiFi[89], that checks incoherence from measures of expected delays).

8.3. Summary of findings

Our findings in this section are twofold. On the one hand, we confirm that simplified simulation assumptions are not to be neglected, as they may hide effects with significant impact – which validates the choice of ndnSIM to maintain fidelity at the expense of scalability, as this reduces the risk of careless evaluation. On the other hand, we find that in an operational regime lower than (even close to) congestion, losses due to transient congestion phenomena imply no noticeable impact on the accuracy – which validates the choice made by ccnSim and Icarus of preferring scalability in this operational regime.

Overall, we see each of the above tools has its own merit (e.g., scalability over large-scale scenario vs fidelity of fine-grained events), and we believe the ICN community can benefit from better exposing when to use which.

9. Related work

Related research closest to this manuscript is represented by works sharing the same methodology and spirit. A list of the most relevant cross comparison of software tools, and especially simulators, that are used for the performance evaluation in the networking community is shown in Tab. 5.

Table 5: Related work summary.

Ref	Year	Domain	#Sim.	Focus
[90]	2003	TCP	3	Scalability
[87]	2006	TCP	2	Accuracy
[91]	2009	TCP	6	Scalability
[92]	2008	WSN	4	User experience
[93]	2010	WSN	14	Scalability
[88]	2011	WSN	4	Accuracy
[35]	2007	P2P	9	Survey
[6]	2013	P2P	9	Survey

In the TCP/IP domain, [91] (and [90]), in particular, implement a grid (dumbell) topology and test 6 (3) simulators, with a special focus on scalability. In [91], to test the consistency of all the implementations, the authors firstly run simulations by varying the drop probabilities and the network size, and monitoring the end-to-end packet loss; then, after having verified that the results are comparable with each other, they run experiments to test the performance of the five simulators in terms of simulation runtime and memory usage. For each set of experiment they vary both the drop probability and the network size. The conducted study is useful to highlight the strengths of each simulator according to the priorities of the users (i.e., scalability, simulation runtime, ease of use). In [90], instead, results concerning the execution speed, the TCP RTT value, and the memory usage against the number of parallel connections are shown; due to the significant differences between the examined simulators, the authors put accent on the importance of understanding peculiarities and limitations of each simulator before effectively running large scale and computational demanding simulations. Fewer work instead tackle the challenge of comparing accuracy, as for instance [87] that thoroughly compares ns2 and NSC TCP implementations against real-world networking stacks (FreeBSD, OpenBSD and Linux). Thanks to efforts such as [87], the networking community is able to perform simulation of TCP/IP networks, knowing the expected level of fidelity.

For what concerns wireless sensor networks, authors in [92] consider four simulators, and compare them according to the amount of effort needed for installation, familiarization, implementation of a specific protocol, and visualization – though no actual comparison of simulation results is attempted. In [93], an extensive survey and taxonomy of fourteen simulators is carried on, after which a quantitative comparison limited to simple scalability test is performed over three of the surveyed simulators, monitoring the computational time and the memory consumption – confirming a tradeoff between scalability and fidelity. Consistency and accuracy is the focus of [88], that employs four WSN simulators and tunes several simulation models (like radio propagation, noise, and energy consumption) calibrated according to a set of experiments on MICAz sensor nodes in indoor and outdoor environments. Authors attempt at performing the same simulations (homogeneous comparison in our terminology) over the four simulators, obtaining inconsistent results concerning the energy consumption and the packet delivery probability – a rather undesirable situation, opposite to what happens in CCN/NDN networks as we have seen in terms of consistency of simulation results.

In the P2P context, authors in [35] present a taxonomy of 9 P2P simulators (according to certain criteria, like simulation architecture, usability, scalability, statistics and underlying network simulation), and surveys over 280 P2P papers. They discover that 62% of them claimed to use a custom proprietary simulator instead of an open source one, with simulation results generally reported in the literature in a fashion that precludes any reproduction. In 2013 a journal extension [6] of the original 2007 editorial note [35] appeared, allowing to take several interesting lessons of the evolutions in the P2P domain. Specifically, quoting [6] while "*it was hoped that the findings of the survey would lead to closer collaboration between the designers of P2P simulators and academics conducting P2P research. A survey of papers that cites this work shows that the opposite has occurred and that there has been further fragmentation in the domain of P2P simulators*". Casting the lesson of P2P domain to ICN, we see that while the ICN situation is currently rather favorable, with a rich ecosystem of consistent tools covering different aspects, there is a risk that a proliferation of custom, proprietary and non validated simulators will dramatically make the picture worse in few years.

10. Discussion and conclusions

This paper first overviews the ICN software ecosystem, finding that despite a number of software tools are available (i) for most architectures only prototypes are available, (ii) about half of the ICN software has been developed around the CCN/NDN architecture, that thus exhibits a larger diversity in terms of prototypes, emulators and (especially) simulators.

The focus is then narrowed down to the largest class of ICN software, namely conducting a rigorous cross-comparison of four CCN/NDN simulators, that we contrast in terms of accuracy, consistency, scalability and fidelity. From the analysis it emerges that, (i) provided simulation scenarios are properly dimensioned, CCN/NDN simulators are able to produce accurate results. What is even more important is that (ii) the examined CCN/NDN simulators yield coherent results even in complex scenarios, which is a rather positive finding. Finally (iii) scalability results exhibit the expected tradeoff between fidelity of the implementation vs memory/computational complexity of the execution, and (iv) fidelity analysis can hopefully guide the community in selecting the right tool, each of which has been found to have merit on its own, for evaluation of different aspects (or specific conditions).

While this paper does provide a useful snapshot of ICN software, it is clearly a starting point more than the end of a journey: indeed, not only the snapshot is incomplete, but it would also need to be regularly updated to reflect the ICN ecosystem evolution. Albeit our comparison methodology is general, and our comparison thorough, it appears clearly, from Fig. 3, that the work that remains to be done is far larger than what has been achieved so far to complete the comparison *within* and *across* categories (e.g., all simulators of an architecture, w.r.t. simulator vs emulator vs prototype of the same architecture and so on).

Henceforth, by raising awareness in the ICN community, our hope is that further effort in widely adopting good practices can not only broaden the knowledge brought by our investigation, but also contribute to avoid pitfalls experienced in other research domains [6]. Possible directions can be identified in:

- fidelity assessment of simulators vs emulators vs prototypes;
- implementation of some reference scenarios, beyond naive ones, in all tools in order to facilitate result comparison;
- making source code available for verifiability, and performing a cross-calibration on baseline scenarios for any new tool;

Clearly, we do not expect the community to align to the point of having a single, common, reference ICN software evaluation suite[9]. At the same time,

[9]Quoting again lesson learned from the P2P community [6], "*In hindsight, it was somewhat naive of us to assume that the P2P academic community would pool their efforts into the design and use of a single P2P simulator*".

we believe that some of these practices are simple enough to be worth sharing (and enforcing) to attain the goal of repeatability and verifiability.

As a final word, we believe that enforcement of these good practices should be done at multiple levels, like at IRTF research group ICNRG with Internet drafts [10, 11, 12], or at academic venues through "challenges" promoting cross-comparison (e.g., Special Interest Groups (SIG) of the Association for Computing Machinery (ACM) do organize both long-standing challenges[10], as well as more temporally scoped contests[11]) in areas other than networking.

Acknowledgements

This work has been carried out at LINCS *http://www.lincs.fr*.

We want to thank the Editor in Chief, the Area Editor, and all the Reviewers, whose guidance through all the steps of the reviewing process helped us improving this work.

References

[1] B. Ahlgren et al., A survey of information-centric networking, Communications Magazine, IEEE 50 (7) (2012) 26 –36.

[2] G. Xylomenos et al., A survey of information-centric networking research, Communication Surveys and Tutorials, IEEE 16 (2) (2014) 1024–1049.

[3] C. Perkins, IP Mobility Support for IPv4, Internet Standard (Aug. 2002). URL `https://tools.ietf.org/html/rfc3344`

[4] N. B. Melazzi, L. Chiariglione, The Potential of Information Centric Networking in Two Illustrative Use Scenarios: Mobile Video Delivery and Network Management in Disaster Situations, IEEE COMSOC MMTC E-Letter 8 (4) (2013) 25–28.

[5] M. Tortelli, D. Rossi, G. Boggia, L. Grieco, Pedestrian Crossing: The Long and Winding Road toward Fair Cross-comparison of ICN Quality, in: Proc. of International Workshop on Quality, Reliability, and Security in Information-Centric Networking, Q-ICN, Rhodes, Greece, 2014.

[6] A. Basu et al., The state of peer-to-peer network simulators, ACM Comput. Surv. 45 (4) (2013) 46:1–46:25.

[7] V. Jacobson et al., Networking Named Content, in: Proc. of ACM CoNEXT, 2009.

[10] Such as the Sort Benchmark in the SIGMOD community (`http://sortbenchmark.org/`) or the SIGKDD (`http://www.kdd.org/kddcup/index.php`) and SIGSPATIAL (`http://sigspatial2014.sigspatial.org/sigspatial-cup`) cups

[11] Such as SIGGRAPH contest for International Collegiate Virtual Reality Contest (IVRC), Immersive Realities and Augmented/Virtual Reality held in recent years

[8] M. Bari, S. Chowdhury, R. Ahmed, R. Boutaba, B. Mathieu, A survey of naming and routing in information-centric networks, Communications Magazine, IEEE 50 (12) (2012) 44–53.

[9] G. Carofiglio et al., From content delivery today to information centric networking, Comput. Netw. 57 (16) (2013) 3116–3127.

[10] D. Kutscher et al., ICN Research Challenges, Internet Draft - `https://tools.ietf.org/html/draft-irtf-icnrg-challenges-02` (Sep. 2015).

[11] K. Pentikousis et al., Information-centric networking: Evaluation methodology, Internet Draft - `https://tools.ietf.org/html/draft-irtf-icnrg-evaluation-methodology-01` (Jul. 2015).

[12] K. Pentikousis et al., Information-centric Networking: Baseline Scenarios, Internet Draft - `https://datatracker.ietf.org/doc/draft-irtf-icnrg-scenarios/` (Mar. 2015).

[13] G. Tyson et al., A survey of mobility in information-centric networks: Challenges and research directions, in: Proc. of the ACM NoM Workshop, 2012.

[14] R. Ravindran, S. Lo, Z. Xinwen, W. Guoqiang, Supporting seamless mobility in named data networking, in: IEEE ICC, 2012.

[15] G. Piro, L. A. Grieco, G. Boggia, P. Chatzimisios, Information-centric networking and multimedia services: present and future challenges, ETT, Transactions on Emerging Telecommunications Technologies, 25 (5) (2014) 392–406.

[16] L. Grieco et al., Iot-aided robotics applications: Technological implications, target domains and open issues, Elsevier Computer Communications 54 (2014) 32 – 47.

[17] L. Grieco, M. Alaya, T. Monteil, K. Drira, Architecting Information Centric ETSI-M2M systems, in: Proc. of IEEE PerCom, 2014.

[18] L. Zhang et al., Named Data Networking, ACM SIGCOMM Computer Communication Review (CCR) 44 (3) (2014) 66–73.

[19] M. Gritter, D. Cheriton, An architecture for content routing support in the internet, in: Proc. of USITS Conference, 2001.

[20] Online website, `http://gregorio.stanford.edu/triad/` (2001).

[21] A. Carzaniga, A. Wolf, Content-based networking: A new communication infrastructure, in: Proc. of IMWS NSF Workshop, 2001.

[22] I. Stoica et al., Internet indirection infrastructure, in: In Proceedings of ACM SIGCOMM, 2002.

[23] A. Carzaniga, M. Rutherford, A. Wolf, A routing scheme for content-based networking, in: Proc of IEEE INFOCOM, 2004.

[24] T. Koponen et al., A data-oriented (and beyond) network architecture, SIGCOMM Comput. Commun. Rev. 37 (4) (2007) 181–192.

[25] N. Fotiou, P. Nikander, D. Trossen, G. Polyzos, Developing Information Networking Further: From PSIRP to PURSUIT, Broadband Communications, Networks, and Systems 66 (2012) 1–13.

[26] C. Dannewitz et al., Network of information (netinf) - an information-centric networking architecture, Comput. Commun. 36 (7) (2013) 721–735.

[27] Online website, `http://www.sail-project.eu/` (2010).

[28] G. Garcia et al., COMET: Content mediator architecture for content-aware networks, in: Future Network Mobile Summit (FutureNetw), 2011.

[29] A. Detti, N. B. Melazzi, S. Salsano, M. Pomposini, CONET: A Content Centric Inter-networking Architecture, in: Proc of the ACM SIGCOMM ICN Workshop, 2011.

[30] I. Seskar, K. Nagaraja, S. Nelson, D. Raychaudhuri, Mobilityfirst future internet architecture project, in: Proc of the 7th Asian Internet Engineering Conference (AINTEC), 2011.

[31] D. Han et al., XIA: Efficient Support for Evolvable Internetworking, in: Proc. of USENIX NSDI, 2012.

[32] Online website, `http://www.greenicn.org/` (2013).

[33] D. Naylor et al., XIA: architecting a more trustworthy and evolvable internet, SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 50–57.

[34] C. Yi, A. Afanasyev, L. Wang, B. Zhang, L. Zhang, Adaptive forwarding in named data networking, SIGCOMM Comput. Commun. Rev. 42 (3) (2012) 62–67.

[35] S. Naicken et al., The state of peer-to-peer simulators and simulations, SIGCOMM Comput. Commun. Rev. 37 (2) (2007) 95–98.

[36] Online website, `http://palproject.org.uk` (2009).

[37] Online website, `http://www.psirp.org/` (2011).

[38] Online website, `http://www.fp7-pursuit.eu/` (2013).

[39] Online website, `http://users.piuha.net/blackhawk/0.3/` (2010).

[40] Simplified Wrapper and Interface Generator (SWIG) online website, `http://swig.org/` (1996).

[41] Online website, https://github.com/fp7-pursuit/blackadder (2013).

[42] D. Trossen et al., IP over ICN - The better IP?, in: Proc. of EuCNC, 2015.

[43] Online website, http://rife-project.eu (2015).

[44] Online website, http://www.read.cs.ucla.edu/click/ (2009).

[45] N. Vastardis, A. Bontozoglou, K. Yang, M. Reed, Simulation Tools Enabling Research on Information-centric Networks, in: IEEE ICC 2012 Workshop on the Network of the Future (FutureNet V), 2012.

[46] Online website, http://privatewww.essex.ac.uk/~nvasta/ICNSim.htm (2014).

[47] H. Casanova et al., Versatile, scalable, and accurate simulation of distributed applications and platforms, Journal of Parallel and Distributed Computing 74 (10) (2014) 2899 – 2917.

[48] Online website, http://www.4ward-project.eu/ (2008).

[49] Online website, http://code.google.com/p/opennetinf/ (2012).

[50] Online website, http://sourceforge.net/projects/netinf/files/nilib/ (2012).

[51] Online website, http://gin.ngnet.it/ (2012).

[52] Online website, http://www.ict-convergence.eu/ (2011).

[53] Online website, http://www.ict-convergence.eu/demodownloads (2011).

[54] Online website, http://netgroup.uniroma2.it/CONET/ (2012).

[55] Online website, https://github.com/XIA-Project/xia-core/wiki (2015).

[56] Online website, http://mobilityfirst.orbit-lab.org/wiki/Proto (2014).

[57] Online website, http://icarus-sim.github.io (2015).

[58] L. Saino, I. Psaras, G. Pavlou, Icarus: a Caching Simulator for Information Centric Networking (ICN), in: Proc. of ICST Valuetools, 2014.

[59] Online website, http://www.ccnx.org (2015).

[60] Online website, http://www.ccnx.org/specifications/ (2015).

[61] Online website, http://www.named-data.net/ (2015).

[62] Online website, http://named-data.net/doc/NFD (2015).

[63] C. Cabral, C. Rothenberg, M. Magalhes, Reproducible network experiments using container-based emulation, in: Proc. of IEEE ISCC, 2014.

[64] Online website, `https://github.com/carlosmscabral/mn-ccnx/wiki` (2014).

[65] Online website, `http://telematics.poliba.it/index.php/en/ccn-joker` (2012).

[66] I. Cianci, L. Grieco, G. Boggia, CCN - Java Opensource Kit EmulatoR for Wireless Ad Hoc Networks, in: Proc. of 7th ACM Int. Conf. on Future Internet Technologies(CFI), 2012.

[67] Online website, `http://www.ccn-lite.net` (2015).

[68] H. Tazaki et al., Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments, in: Proc. of ACM CoNEXT, 2013.

[69] Online website, `http://www-sop.inria.fr/members/Frederic.Urbani/ns3dceccnx/getting-started.html` (2012).

[70] Online website, `http://systemx.enst.fr/ccnpl-sim` (2014).

[71] G. Carofiglio, M. Gallo, L. Muscariello, Joint hop-by-hop and receiver-driven interest control protocol for content-centric networks, in: Proc. of ACM SIGCOMM ICN Workshop, 2012.

[72] G. Carofiglio, M. Gallo, L. Muscariello, ICP: Design and evaluation of an interest control protocol for content-centric networking, in: Proc. of IEEE INFOCOM NOMEN Workshop, Orlando, Florida, 2012.

[73] Online website, `http://ndnsim.net` (2015).

[74] A. Afanasyev, I. Moiseenko, L. Zhang, ndnSIM 2.0: A new version of the NDN simulator for NS-3, in: Tech. Rep. NDN-0028, 2015.
URL `http://named-data.net/techreport/ndn-0028-1-ndnsim-v2.pdf`

[75] A. Afanasyev, I. Moiseenko, L. Zhang, ndnSIM: A ns-3 Based NDN Simulator, in: CCNxCon'12, Sophia Antipolis, France, 2012.

[76] Online website, `http://www.enst.fr/~drossi/ccnSim` (2015).

[77] G. Rossini, D. Rossi, ccnSim: a highly scalable CCN simulator, in: Proc. of IEEE ICC, 2013.

[78] I. Psaras, W. Chai, G. Pavlou, Probabilistic in-network caching for information-centric networks, in: Proc. of ACM ICN Workshop, 2012.

[79] L. Saino, I. Psaras, G. Pavlou, Hash-routing schemes for information centric networking, in: Proc. of ACM ICN Workshop, 2013.

[80] S. Fayazbakhsh et al., Less Pain, Most of the Gain: Incrementally Deployable ICN, in: Proc. of ACM SIGCOMM, 2013.

[81] H. Che, Z. Wang, Y. Tung, Analysis and design of hierarchical web caching systems, in: Proc of IEEE INFOCOM, 2001.

[82] V. Martina, M. Garetto, E. Leonardi, A unified approach to the performance analysis of caching systems, in: Proc. of IEEE Infocom, 2014.

[83] S. Arianfar, P. Nikander, Packet-level Caching for Information-centric Networking, in: Proc. of ACM SIGCOMM, ReArch Workshop, 2010.

[84] A. Hoque et al., NLSR: Named-data Link State Routing Protocol, in: Proc. of ACM SIGCOMM ICN Workshop, 2013.

[85] M. Tortelli, D. Rossi, G. Boggia, L. A. Grieco, Ccn simulators: Analysis and cross-comparison, `http://perso.telecom-paristech.fr/~drossi/index.php?n=Software.ICN14Demo` (2014).

[86] G. Rossini, D. Rossi, Evaluating CCN Multi-path Interest Forwarding Strategies, Comput. Commun. 36 (7) (2013) 771–778.

[87] S. Jansen, A. McGregor, Performance, validation and testing with the network simulation cradle, in: IEEE MASCOTS, 2006, pp. 355–362. doi:10.1109/MASCOTS.2006.40.

[88] A. Stetsko, M. Stehlik, V. Matyas, Calibrating And Comparing Simulators for Wireless Sensor Networks, in: Proc. of IEEE MASS, 2011.

[89] N. Handigol et al., Reproducible network experiments using container-based emulation, in: Proc. of ACM CoNEXT, 2012.

[90] D. Nicol, Scalability of Network Simulators Revisited, in: Proc. of the CNDS Conference, 2003.

[91] E. Weingartner, H. vom Lehn, K. Wehrle, A performance comparison of recent network simulators, in: Proc. of IEEE ICC, 2009.

[92] J. Lessmann et al., Comparative study of wireless network simulators, in: Proc. of IEEE ICN, 2008.

[93] H. Sundani et al., Wireless sensor network simulators a survey and comparison, Int. Journal of Computer Networks 2 (5) (2010) 249–265.