A Model for Named Data Networking Inspired by Nonlinear Dynamical Systems

António D. N. Rodrigues
Faculdade de Engenharia da Universidade do Porto (FEUP), Porto, Portugal
adamiaonr@fe.up.pt

***Abstract*—At the time of its inception, the Internet mostly served the purposes of communication between connected end-hosts. Now, at the World Wide Web era, the Internet is immersed in a content-centric paradigm, more concerned about content generation, sharing and access. Recently, a new research trend — Information Centric Networking (ICN) — started advocating for deep modifications on the Internet's network layer, making it content-centric by design, including the widespread use of in-network caching.**

In this paper, we focus on the analysis of cache behavior in a specific ICN architecture — Named Data Networking (NDN) — under different cache algorithms, network topologies and content usage characteristics. To do so, we specify a simple and but modular NDN router model, loosely inspired in nonlinear dynamical systems. We implement the specified model in MATLAB®, providing some simulation results with three simple cache algorithms, specifically (1) Least Recently Used (LRU), (2) More Recently Used (MRU) and (3) Random caching.

I. Introduction

Departing from its initial model as a network for host-to-host communications, the Internet started moving towards a content-centric model with the advent of the World Wide Web in the 1980s. This model persisted, and with increasingly demanding usage requirements, leading to the development of technologies such as Content Delivery Networks (CDNs) and Peer-to-Peer (P2P) networks [1]. These were built around the architecture's edge, due to the so-called 'ossification' [2] of the Internet's core, leading to inefficiencies in terms of latency, bandwidth usage, among others. Given the widespread adoption of the content-centric model, researchers to think about new and clean-slate designs for the Internet's core, in order for it to natively cope with these issues. Among such efforts [3], the research field of Information Centric Networking (ICN) [4] emerged, advocating the deliberate abolition of network locators, replacing of IP addresses with content identifiers and calling for the widespread use of in-network caching, so that content can be easily served from multiple anywhere in the network [5]–[10]. Here we focus on the aspect of in-network caching in one of such clean slate designs, the Named Data Networking (NDN) architecture [6].

In this paper, we focus on the analysis of cache behavior in NDN networks under different cache algorithms, network topologies and content usage characteristics. To do so, we specify a simple and but modular NDN router model, loosely inspired in nonlinear dynamical systems [11]. We implement the specified model in MATLAB®, providing some simulation results with three simple cache algorithms, specifically (1) Least Recently Used (LRU), (2) More Recently Used (MRU) and (3) Random caching. The main contribution of this work consists in the provision of a framework in MATLAB®, which allows for the simulation of NDN network behavior under different topologies, cache algorithms, NDN router characteristics, etc. without the complexity of more elaborated network simulators.

The remainder of this paper is organized as follows. In Section II we provide an overview over the NDN architecture, focusing on the basic operation of its forwarding engine and the way it involves in-network caching. In Section III, we present the overall methodology followed during this work, including an explanation of the considered NDN router model, network topologies to be considered, cache algorithms, etc. In Section IV we present a set of experiments ran over our model implementation, as well as the respective results. Finally, in Section VI we draw some pertinent conclusions from the presented work.

II. Named Data Networking (NDN)

In the Named Data Networking (NDN) [6] architecture, clients issue subscriptions for content objects by specifying a hierarchical (URL-like) content name, e.g. `/pdeec/mtsp/2014/`, which is directly used in NDN packets. Destination network locators (e.g. IP addresses) are not used in this case, as NDN routers are able to forward such packets towards appropriate content-holding destinations, solely based on such names. NDN contemplates two fundamental types of packets, 'Interest' and 'Data' packets, used for content subscriptions and publications, respectively. Interest packets are originally released into the network by clients willing to access a particular content, addressing it via its content name, while Data packets carry the content itself.

An NDN router is conceptually composed by three main elements: (1) a Forward Information Base (FIB), (2) a Pending Interest Table (PIT) and (3) a Content Store (CS) [6]:

- **Forward Information Base (FIB):** Routing/forwarding table holding entries which relate a name prefix and a list of router interfaces to which Interest packets matching that content name prefix should be forwarded to.
- **Pending Interest Table (PIT):** A table which keeps track of the mapping between arriving Interest packets and the interfaces these have been received from, in order to save a reverse path for Data packets towards one or more

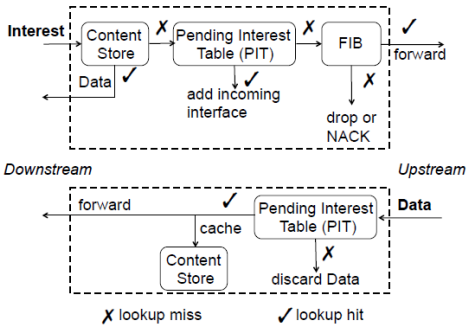


Fig. 1: Interest and Data packet processing according to NDN's forwarding engine [12].

subscribers (this may be a 1:N mapping, as an Interest packet matching the same content may be received in multiple interfaces).

- **Content Store (CS):** A cache for content, indexed by content name or item. This novel element allows for content storage at the network level. In-network caching allows an Interest to be satisfied by a matching Data packet in any location other than the original producer of the content, constituting one of the main content-oriented characteristics of NDN.

In NDN, communication is receiver-driven, i.e. having the desire to fetch a particular content, a client releases an Interest packet into the network so that it is forwarded towards an appropriate content holder. In Figure 1 [12], we provide a graphical description of the mechanics of the forwarding engine of an NDN router, supported by the textual description provided below:

1) An Interest packet arrives on an interface (e.g. `iface0`) of an NDN router.
2) A longest prefix match on the content name specified in the Interest (e.g. `name`) is performed. The NDN router will now look in its CS, PIT and FIB, in that order, in order to resume the forwarding action:
   a) If there's a match in the router's CS, a copy of the respective CS entry will be sent back via `iface0`, the Interest packet is dropped. Depending on the pre-specified caching policy (e.g. MRU, LRU, LFU [1], etc.), the organization of the CS may change at this point. **End.**
   b) Else if there is an (exact) match in the PIT, `iface0` is added to the mapping list on the respective entry. The Interest packet is dropped (as a previous one has already been sent upstream). **End.**
   c) Else if only a matching FIB entry is found, the Interest packet is forwarded upstream, via all remaining interfaces on the list (except `iface0`), towards an eventual content holder. A PIT entry <`name`, `iface0`> is added. **End.**
   d) Else if there is no match at all, the Interest packet is simply discarded. **End.**

[1] http://en.wikipedia.org/wiki/Cache_algorithms

Note that in NDN only Interest packets are forwarded according to the FIB: intermediate NDN routers (i.e. between client and content holder) forward the Interests and have their respective PIT tables updated with Interest-to-interface mappings, pre-establishing a reverse path for Data packets to follow as soon as a content holder is found. When the reverse path is 'followed' (i.e. in the 'downstream' direction, lower part of Figure 1), each intermediate NDN router receiving a Data packet looks in its PIT for <`name`, `iface`> entries, and forwards the Data packet through all matching interfaces. In addition, a CS entry is created to cache the content locally at the router (again, depending on the caching policy, the organization of the CS may change at this point). If a Data packet with no matching PIT entries arrives, it is treated as unsolicited and discarded.

III. Methodology

In this section we describe the proposed NDN model in detail. We start with an overview, setting the notation and crudely establishing its relation with nonlinear dynamical systems. We then continue with a detailed description of each model component and related procedures.

A. Overview

We consider a conceptual NDN network composed by three main types of entities: (1) $|R|$ NDN routers[2] R_r, $r = \{1, 2, ..., |R|\}$, organized in some type of topology (e.g. cascade, tree, etc.); (2) a set of $|C|$ clients C_c, $c = \{1, 2, ..., |C|\}$; and a single content server S, holding $|O|$ different content objects O_o, $o = \{1, 2, ..., |O|\}$ (e.g. $|O|$ different photos).

Clients issue requests for content objects O_o, i.e. Interest packets i_{O_o}, which are propagated through NDN routers towards the content server S, and eventually followed by Data packets d_{O_o}, containing the requested content object. We represent the elementary set of signals fed to/read from the inputs/outputs of the aforementioned basic entities, at some discrete time n, as a $2|O| \times 1$ vector in the form

$$\mathbf{v}[n] = \begin{bmatrix} i_{O_1} \\ i_{O_2} \\ \dots \\ i_{O_{|O|}} \\ d_{O_1} \\ d_{O_2} \\ \dots \\ d_{O_{|O|}} \end{bmatrix} \quad (1)$$

Each component i_{O_o} or d_{O_o} may assume an integer value, i.e. $i_{O_o}, d_{O_o} \in \mathbb{N}_0 = \{0, 1, 2, ...\}$, representing the absence (in case of $i_{O_o}, d_{O_o} = 0$) or presence (in case of $i_{O_o}, d_{O_o} > 0$) of an Interest/Data packet, at a given discrete time n. E.g. considering a setting with $|O| = 2$ content objects, a value of $\mathbf{x}$ corresponding to the presence of two Interests for content

[2] Here we use the notation $|E|$ to represent the number of elements of type E.

O_1 and one Data packet for O_2 (with the absence for the remaining components) would be encoded as

$$\mathbf{v}[n] = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{2}$$

With this representation, we capture situations in which a network entity may simultaneously receive/issue any Interest or Data packet at its input/output. Our models for NDN routers, clients or servers can be seen as nonlinear dynamical subsystems, accepting inputs $\mathbf{u}$ and producing outputs $\mathbf{y}$, in the form shown in eq. 1 and eq. 2. Furthermore, each one of these subsystems exhibits one or more types of state $\mathbf{x}$ (e.g. the composition of a Content Store or a Pending Interest Table), which evolves according to nonlinear dynamics $\mathcal{H}$, driven by $\mathbf{u}$ and the current state $\mathbf{x}$:

$$\mathbf{x}[n+1] = \mathcal{H}(\mathbf{x}[n], \mathbf{u}[n]) \tag{3}$$

Outputs $\mathbf{y}$ are nonlinearly related to some current state $\mathbf{x}$ and input $\mathbf{u}$:

$$\mathbf{y}[n] = \mathcal{G}(\mathbf{x}[n], \mathbf{u}[n]) \tag{4}$$

We also note that in some subsystems, namely at the clients C, the outputs $\mathbf{y}$ may include a stochastic component w (e.g. in cases where Interest signals are randomly generated):

$$\mathbf{y}[n] = \mathcal{G}(\mathbf{x}[n], \mathbf{u}[n]) + w \tag{5}$$

In the next subsections, we describe each one of the nonlinear subsystems in detail, including the types of state and associated nonlinear dynamics.

B. NDN Router Model

An NDN router is the central entity of the presented model, acting as the main agent of NDN's forwarding engine. It is also the more complex entity, including a set of submodules, already mentioned in Section II: (1) the Pending Interest Table (PIT); (2) the Content Store (CS); and (3) the Forward Information Base (FIB). We first provide an overview of our NDN router module, and then proceed with the description of each one of its submodules.

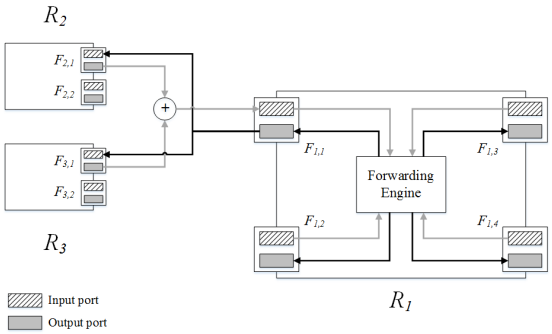


Fig. 2: Graphical depiction of our NDN router model.

Figure 2 provides a graphical description of the proposed router model. A router R contains a set of $|F|$ interfaces F, used to interconnect it to other entities (e.g. some other router R', a client C or a server S). Each interface can subsequently be divided into one input and one output port, which 'cross-connect' with the ports of the attached interfaces (see Figure 2). We use the notation $F_{r,f,in}$ or $F_{r,f,out}$, to refer to the input/output ports of an interface with index f, of a router R_r[3].

Taking Figure 2 as a supporting example, the act of forwarding some set of Interest/Data packets from a router R_1, over some interface F_1, is modeled by having R_1 fill $F_{1,1,out}$ with some set of signals $\mathbf{y}$, following the encoding shown in Section III-B. Conversely, the act of receiving some set of Interest/Data packets is modeled by having routers R_2 and R_3 — connected with $F_{1,1}$ via $F_{2,1}$ and $F_{3,1}$ — fill $F_{2,1,in}$ and $F_{3,1,in}$ with $\mathbf{u}$ [4]. As seen in Figure 2, more than one entity may be connected to some interface, in which case the signals $\mathbf{y}$ originating from the interconnected interfaces' output ports, e.g. $F_{2,1,out}$ and $F_{3,1,out}$, are combined and summed at the other end's input port, e.g. $F_{1,1,in}$. While the use of interfaces and input/output ports may be seen as a case of over engineering, we argue it makes our model robust, highly modular and capable of supporting multiple network topologies.

1) Pending Interest Table (PIT): In the same way that the FIB commands the forwarding of Interests, the PIT commands the forwarding of Data packets. The composition of the PIT is more dynamic than that of the FIB, being dependent on the flow of Interest and Data signals through the NDN router. We model the PIT as a $|O| \times |F|$ matrix in the form:

$$\mathbf{PIT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{6}$$

PIT[5] entries (o, f) are encoded as 0 or 1: if $(o, f) = 1$, Interests for content object O_o have been previously received in interface $F_{r,f}$, and so Data packets d_{O_o} shall be forwarded via $F_{r,f}$; on the other hand, if $(o, f) = 0$, d_{O_o} should not be forwarded via that interface. The arrival of Interest and Data packets influences the composition of the PIT over time, each triggering special routines — **PIT::updateOnInterest()** and **PIT::updateOnData()** — shown below. We consider special $2\,|O| \times |F|$ matrices $\mathbf{U}$ and $\mathbf{Y}$, corresponding to the concatenation of all the column vectors $\mathbf{u}_{r,f}$ and $\mathbf{y}_{r,f}$, i.e. the contents from the input and output ports of all the interfaces $F_{r,f}$, at some router R_r:

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_{r,1} & \mathbf{u}_{r,2} & \dots & \mathbf{u}_{r,|F|} \end{bmatrix} \tag{7}$$

[3]We adopt a flexible notation, sometimes referring to an interface as $F_{r,f}$ when there is no need to specify a particular port.

[4]In fact, this procedure can be extended to any network entity, let it be a router, a client or a server.

[5]We use the form 'PIT' for general references to the Pending Interest Table, and '**PIT**' when referring to its matrix form, as in eq. 6. This dual representation is extended to the FIB and CS.

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_{r,1} & \mathbf{y}_{r,2} & \cdots & \mathbf{y}_{r,|F|} \end{bmatrix} \quad (8)$$

1: **define** PIT::updateOnInterest(**U**):

2: $\mathbf{U}' \leftarrow \mathbf{U}(1:|O|,:)$
3: $\mathbf{Y}'_i \leftarrow \neg(\mathbf{PIT} \times \mathbf{1}^{|F|}) \;\&\; (\mathbf{U}' \times \mathbf{1}^{|F|})$
4: $\mathbf{Y}'_i \leftarrow \mathbf{FIB} \;\&\; (\mathbf{Y}'_i \times \mathbf{1}^{|F|^T})$
5: $\mathbf{Y}_i \leftarrow \begin{bmatrix} \mathbf{Y}'_i \\ \mathbf{0}^{|O|\times|F|} \end{bmatrix}$
6: $\mathbf{PIT} \leftarrow \mathbf{PIT} \mid \mathbf{U}'$
7: **return** $\mathbf{Y}_i$

Upon the reception of Interest signals, i.e. $\mathbf{U}'$ (the first $|O|$ rows of **U**), we first identify the content items O_o, or the row indexes o of the **PIT**, for which there are **no** pending Interests (line 3). For convenience, we often recur to binary operations (negation '$\neg$', conjunction '&', disjunction '|') over matrices: e.g. in line 3, after summing all columns of the **PIT**[6], we negate the result, obtaining a binary encoded $|O| \times 1$ column vector which indicates the absence (encoded as '1') and presence (encoded as '0') of pending Interests for some content object O_o. Still in line 3, we obtain a $|O| \times 1$ column vector, $\mathbf{Y}'_i$, which encodes Interest signals which are not already stored in the PIT, and that the NDN router needs to forward upstream. This is accomplished by taking the logic AND, '&', between $\neg(\mathbf{PIT} \times \mathbf{1}^{|F|})$ and $\mathbf{U}'$. Note that even if $\mathbf{U}'$ includes some $i_{O_o} > 1$, we only need to forward one Interest over the interfaces specified in the FIB, and so the binary encoding of $\mathbf{Y}'_i$, resulting from the use of binary operations, neatly serves our purposes. The operation in line 4 results in a final form of $\mathbf{Y}'_i$, which assigns the Interest signals to be forwarded upstream to the appropriate interfaces, according to the FIB. In line 5 we finally obtain $\mathbf{Y}_i$, a $2|O| \times |F|$ matrix compliant with the format shown in eq. 8. In line 6, the contents of the **PIT** are updated by performing a logic OR, '|', with $\mathbf{U}'$, so that it registers all the newly received Interest signals and their correspondence to interfaces. This last step is important, as it allows future Data packets to be forwarded downstream over the requesting interfaces.

1: **define** PIT::updateOnData(**U**):

2: $\mathbf{U}' \leftarrow \mathbf{U}(|O|+1:2|O|,:)$
3: $\mathbf{G} \leftarrow (\mathbf{U}' \times \mathbf{1}^{|F|}) \times \mathbf{1}^{|F|^T}$
4: $\mathbf{Y}'_d \leftarrow \mathbf{PIT} \;\&\; \mathbf{G}$
5: $\mathbf{Y}_d \leftarrow \begin{bmatrix} \mathbf{0}^{|O|\times|F|} \\ \mathbf{Y}'_d \end{bmatrix}$
6: $\mathbf{PIT} \leftarrow \mathbf{PIT} \;\&\; \neg\mathbf{G}$
7: **return** $\mathbf{Y}_d$

Note that the objective of PIT::updateOnData() is the reverse operation of PIT::updateOnInterest(), i.e. forwarding Data packets over the interfaces for which there is a registered Interest in the **PIT**, with the result encoded in matrix $\mathbf{Y}_d$. **G** consists in a $|O| \times |F|$ matrix, which expands the single-column vector $\mathbf{U}' \times \mathbf{1}^{|F|}$ to $|F|$ columns (line 3), making it ready for the subsequent AND ('&') operations (lines 4 and 6). The final step of the operation (line 6) consists in erasing all Interest registrations which have been successfully attended.

As a final remark, we can now identify the **PIT** as one form of state **x**, which is driven by the nonlinear dynamics specified on the PIT::updateOnInterest() and PIT::updateOnData() routines (line 6, in both cases). Furthermore, both routines also contribute to the generation of outputs, in the form of matrices $\mathbf{Y}_i$ and $\mathbf{Y}_d$.

2) Content Store (CS): We model the Content Store (CS), i.e. the NDN router's cache, as an $|O| \times |P|$ matrix, in which $|P|$ is the size of the CS, i.e. maximum number of content objects it is able to accommodate at any given point in time. We use P to represent a position or slot in the CS, which is able to hold a single content object O_o (here we do not consider a notion of content object size, an object always fits in a slot P). E.g. in eq. 9 we show the encoding of the **CS** for R_1 in Figure 3, with $|P| = 2$, which is shown to contain content objects O_2 and O_4[7].

$$\mathbf{CS} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad (9)$$

Again, we follow a binary encoding, using $(o,p) = 1$ to indicate the presence of content O_o at slot P_p, and $(o,p) = 0$ as an indication of its absence. Each slot is assigned a different integer index, i.e. P_p, with $p = \{1, 2, ..., |P|\}$, which express the idea of 'cache levels': O_o, occupying slot P_2, is at the 2nd (highest) position of the **CS**. P_1 is usually interpreted as the 'highest' level, and $P_{|P|}$ the 'lowest', nevertheless the meaning is of the 'levels' is often dependent on the cache algorithm implemented by the CS.

Note that **CS** must obey a set of constraints, since (1) each slot can only hold a single content object, and (2) it is inefficient for the **CS** to hold multiple copies of a content O_o. Specifically, the conditions for the validity of **CS** are:

$$\sum_{o=1}^{|O|} \mathbf{CS}_{o,p} \leq 1 \quad \forall\; p \in 1, 2, ..., |P| \quad (10)$$

$$\sum_{p=1}^{|P|} \mathbf{CS}_{o,p} \leq 1 \quad \forall\; o \in 1, 2, ..., |O| \quad (11)$$

Similarly to the PIT, the operations related to the CS are implemented by two routines: **CS::updateOnInterest()** and **CS::updateOnData()**. While the behavior of CS::updateOnData() is specific to a particular cache algorithm (see Section III-D), that of CS::updateOnInterest() is rather general, consisting in the generation of a pair of $|O| \times |F|$ matrices, $[\mathbf{Y}_h, \mathbf{R}]$. In short, $\mathbf{Y}_h$ encodes the Data packets for which there are cache hits, already assigned to the appropriate

[6]We use the notation $\mathbf{1}^m$ for the $m \times 1$ sum vector, i.e. $\mathbf{1}^m = [1\,1\,...\,1]^T$, and $\mathbf{1}^{m\times n}$ for an $m \times n$ matrix solely composed by '1'. Equivalently, we also use $\mathbf{0}^m$ and $\mathbf{0}^{m\times n}$.

[7]Usually, we consider $|P| << |O|$.

interface columns; and **R** encodes all the Interest signals for which there were no cache hits[8].

Again, **CS** may be seen as the other form of state in the NDN router subsystem, driven by the nonlinear dynamics specified by each different cache algorithm. Furthermore, the CS::updateOnInterest() routine, common to all caching policies, contributes to the generation of the output matrix **Y** via $\mathbf{Y}_h$.

3) Forwarding Information Base (FIB): The FIB is important for the act of forwarding Interest packets towards appropriate content sources, by indicating the interfaces F over which such sources are reachable. For each NDN router R, we model the FIB as a simple $|O| \times |F|$ matrix in the form

$$\textbf{FIB} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{12}$$

The **FIB** encoding shown in eq. 12 would be held by router R_1 in the simple topology shown in Figure 3. **FIB** entries (o, f) are encoded as 0 or 1: if $(o, f) = 1$, content objects of type O_o are accessible via interface F_f, meaning that Interests i_{O_o} shall be forwarded via F_f; on the other hand, if $(o, f) = 0$, i_{O_o} should not be forwarded via that interface. Following the same type of operations described for the PIT in Section III-B1, we now present the algorithm followed by NDN routers to forward Interest signals, which uses PIT::updateOnInterest() as well as CS::updateOnInterest():

1: **define** Router::forward(**U**):

2: $[\mathbf{Y}_h, \mathbf{R}] \leftarrow$ CS::updateOnInterest(**U**)
3: $\mathbf{Y}_i \leftarrow$ PIT::updateOnInterest(**R**)
4: $\mathbf{Y}_d \leftarrow$ PIT::updateOnData(**U**)
5: $\mathbf{Y} \leftarrow (\mathbf{Y}_i + \mathbf{Y}_h + \mathbf{Y}_d)$
6: **return Y**

From CS::updateOnInterest(), we obtain $\mathbf{Y}_h$, the output Data signals resulting from cache hits, and **R**, the Interest signals for which there were no cache hits. We use **R** as an input to PIT::updateOnInterest(), ultimately obtaining $\mathbf{Y}_i$, the output Interest signals. Finally, the router processes the Data input signals on **U**, generating the Data output signals, $\mathbf{Y}_d$, according to the behavior of PIT::updateOnData(). The forward operation finally returns a complete output matrix **Y**, consisting in the combination of $\mathbf{Y}_i$, $\mathbf{Y}_h$ and $\mathbf{Y}_d$.

In our model, the composition of the **FIB** is established at an initial phase, not suffering any further alterations (more details in Section III-F). Note that the FIB only commands Interest forwarding actions, not participating in the forwarding of Data packets.

[8]Due to space constraints, we do not show the algorithm of CS::updateOnInterest(), which can nevertheless be consulted in github.com/adamiaonr/mtsp-project.

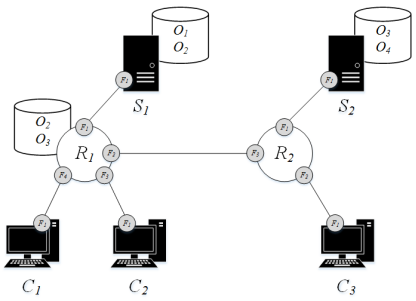


Fig. 3: Simple example of an NDN network topology.

C. Endpoints

In our model we consider two forms of endpoints: clients C and servers S. Clients generate Interests for content, i.e. vectors in the form of **v** in eq. 1, while servers hold caches in which content objects persist and are always available.

The operation of servers is simple, basically consisting in mirroring the Interest signals received as input, according to the contents of their persistent CS:

1: **define** Server::mirror(**U**):

2: $\mathbf{U}' \leftarrow$ **CS** & **U**

3: $\mathbf{Y}_h \leftarrow \begin{bmatrix} \mathbf{0}^{|O| \times |F|} \\ \mathbf{U}'(1 : |O|, :) \end{bmatrix}$

4: **return** $\mathbf{Y}_h$

Clients generate Interest signals according to some popularity distribution such as Zipf [13]. In our model, each client C independently takes a $|O| \times 1$ vector **q**, containing the Interest generation probabilities for each content object O_o, q_o. During a simulation, at some discrete time instant n, a client C generates an Interest for content object O_o with probability q_o.

D. Cache Algorithms

In order to evaluate the feasibility of the proposed model, we consider two well known cache algorithms, LRU [14] and MRU [15], in addition to a Random caching policy, which randomly evicts a content object cached at the CS, in the presence of a new object to be cached. In practice, each different caching algorithm constitutes a different implementation of the CS::updateOnData() routine, mentioned in Section III-B2, and basically defines the 'type' of CS for an NDN router as an input parameter.

As the design of cache replacement policies has not been the main purpose of this work, we do not explain the aforementioned algorithms in detail. Their implementation has nevertheless been part of this work, and can be consulted in github.com/adamiaonr/mtsp-project.

E. Network Topologies

We represent a network topology using a square matrix **T**, with dimension $|R| + |C| + |S|$. For representation purposes,

we attribute an integer index to each one of the network nodes, and so each elements (i, j) of the matrix corresponds to interconnections between network entities with indexes i and j. The values at each (i, j) position identify the **near end interface** of the connection, i.e. the interface at entity i. E.g. the matrix **T** shown in eq. 13 encodes the topology shown in Figure 3. In this case, we attribute the integers 1 to 7 to the network entities, starting with the clients C_1 to C_3, then routers R_1 and R_2 and finally the servers S_1 and S_2.

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 4 & 3 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 2 & 3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{13}$$

F. 'Connecting the Dots'

Given all the 'pieces of the puzzle', we can now describe how the proposed model behaves under simulation. A simulation is composed by two distinct phases: (1) a setup phase, and (2) a round phase. During phase 1, the simulation engine interprets (a) the topology matrix **T**, (b) the content parameters (number of contents $|O|$ and popularity distribution **q**) and (c) the CS parameters (size $|P|$ and type), creating the overall network entities — routers, clients and servers — as well as the appropriate interconnections.

In phase 2, given a number of rounds $|N|$, the simulation engine iteratively executes the following algorithm:

```
1:  define SimEngine::run():
2:
3:  for (N ← 1; N < |N|; N++) do

4:      for all C_c do
5:          U ← C_c.requestContent()
6:          C_c.putOutPorts(U)
7:      end for

8:      for all R_r do
9:          U ← R_r.getInPorts()
10:         Y ← R_r.forward(U)
11:         R_r.putOutPorts(Y)
12:     end for

13:     for all S_s do
14:         U ← S_s.getInPorts()
15:         Y_h ← S_s.mirror(U)
16:         S_s.putOutPorts(Y_h)
17:     end for

18: end for
```

Each simulation round N consists in a simple sequence of steps: (1) having clients generate Interest signals and pushing them to the output ports of their interfaces; (2) having routers fetch their interfaces' input ports, process Interest and Data packets, according to the Router::forward() routine, defined in Section III-B3, and pushing the result to their interfaces' output ports; and finally (3) having servers load their interfaces' input ports, perform the mirroring process specified in Section III-C, and pushing the result to their interfaces' output ports. The topology setup performed during phase 1 guarantees the appropriate flow of Interest and Data signals between the input and output ports of the interfaces of the various network components.

G. Implementation

The NDN network model described above has been implemented in MATLAB®, taking advantage of its Object Oriented Programming (OOP) capabilities. Most of the object types are independent of particular cache algorithms, nevertheless CS objects are defined as abstract classes, allowing for specific implementations of cache policies such as LRU, MRU and Random caching. The MATLAB® code — along with additional documentation — is freely available at github.com/adamiaonr/mtsp-project.

IV. Experiments

This section describes a set of experiments conducted to evaluate the behavior of our NDN model, under different cache conditions. In Section IV-A, we start with a brief description of the setup of our experiments, including relevant parameters (e.g. number of content objects, popularity distribution, etc.), network topologies, cache characteristics, as well as the metrics to be considered. Later, in Section IV-B, we present the results of several experiment runs, according to the metrics specified in Section IV-A.

A. Experimental Setup

The conducted experiments can be characterized along three main dimensions: (1) content object characteristics; (2) network topologies; and (3) cache characteristics.

1) Content Object Characteristics: For all experiments we consider a content object space of $|O| = 100$, with decreasing popularity and following a Zipf distribution: each content object O_o, with $o = \{1, 2, ..., |O|\}$, is requested by a client C with probability $q_o = \frac{c}{o^\alpha}$, with $c = 0.8$ and $\alpha \in \{0.25, 0.5, 1, 2\}$, depicted in Figure 4.

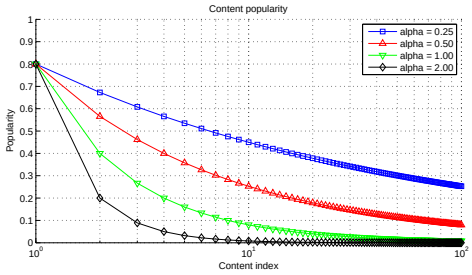
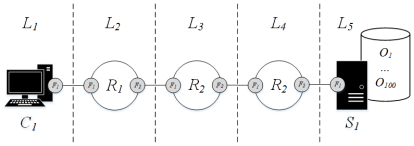


Fig. 4: Content popularity distribution, for $c = 0.8$ and $\alpha \in \{0.25, 0.5, 1, 2\}$.

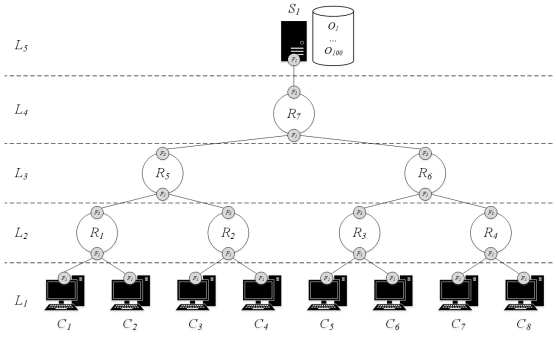
At each experiment round (see Section III-F for more details on rounds), each client C generates a signal for content object

O_o with probability q_o. We consider a number of rounds $|N| = 10000$ for all experiment runs.

2) Network Topologies: We consider two types of topologies: (1) a cascade topology with $|L| = 5$ levels ($|C| = 1$, $|R| = 1$ and $|S| = 1$), shown in Figure 5(a); (2) a binary tree topology, also with $|L| = 5$ levels ($|C| = 8$, $|R| = 7$ and $|S| = 1$), as shown in Figure 5(b).



(a)



(b)

Fig. 5: Graphical depiction of the topologies considered for our experiments: (a) cascade topology ($|L| = 5$, $|C| = 1$, $|R| = 3$ and $|S| = 1$); (b) binary tree topology ($|L| = 5$, $|C| = 8$, $|R| = 7$ and $|S| = 1$).

The definition of 'topology level' is straightforward and depicted in Figure 5. Furthermore, we set the FIBs of every NDN router to forward Interests over the upstream interfaces, i.e. $F_{r,2}$, resulting in the form:

$$\textbf{FIB} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 1 & 0 & \dots & 0 \end{bmatrix} \tag{14}$$

3) Cache Characteristics: We evaluate the performance of our model under three different types of cache algorithms, already described in Section III-D: (1) Least Recently Used (LRU); (2) More Recently Used (MRU); and (3) Random caching. We consider different cache sizes, specifically $|P| = \{10, 25, 50, 75\}$.

4) Metrics: For all our experiments we collect and evaluate the following metrics:

1) Total number of Interests and Data packets received/sent per network node/level;
2) Cache hit/miss rate per content object and network level;
3) Ratio of received Interests to original requests, per content object and network level (including server(s) S);
4) Relative time spent at cache per content object.

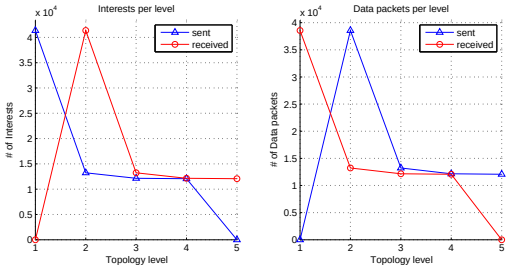
Regarding metric 2, we first provide our definition of 'cache hit': a cache hit happens when an Interest signal for some content object O_o, arriving at some router R_r, finds a cached copy of O_o at the CS of R_r. Therefore, to compute metric 2, we consider the Interest signals arriving at all NDN routers of some level L, discriminated by content object O_o, for all simulation rounds N. We then find the ratio between Interest 'hits' for O_o, i'_{O_o}, and the total value of arriving Interests for O_o at that level, i_{O_o}:

$$\frac{\sum_{n=1}^{|N|} i'_{O_o}}{\sum_{n=1}^{|N|} i_{O_o}} \quad \forall\ R_r \text{ in level } L \tag{15}$$

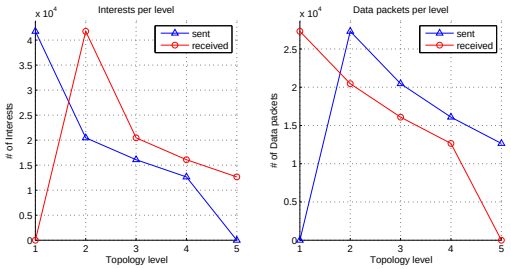
For metric 4, we simply find the average number of rounds some content object O_o spends at the caches of NDN routers of some level L, and find the ratio vs. the total number of simulation rounds.

B. Experimental Results

Here we present a selection of experimental results, following the metrics introduced in Section IV-A[9]. The captions on the figures enclose the respective experimental parameters.



(a)



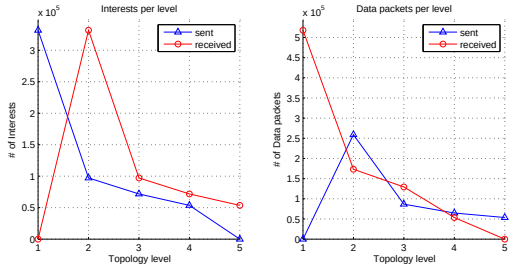
(b)

Fig. 6: Number of Interests/Data packets sent/received per topology level, for different cache algorithms: LRU (a), MRU (b). Cascade topology, $|P| = 25$, $\alpha = 1$, LRU caching.

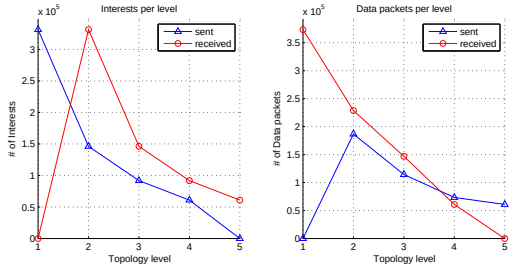
V. Discussion

Figures 6 and 7 show a comparison of the quantity of Interest and Data packets received/sent per topology level, for both the cascade and binary tree topologies in Figure 5. Both topologies clearly show a significantly larger number of received Interest signals at level 2 (in fact, level 2 receives all the Interest signals generated by the clients at level 1), compared

[9]A complete collection of results can be found in github.com/adamiaonr/mtsp-project/tree/master/report/figures/experiments.



(a)

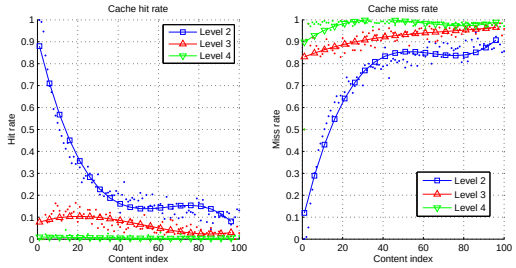


(b)

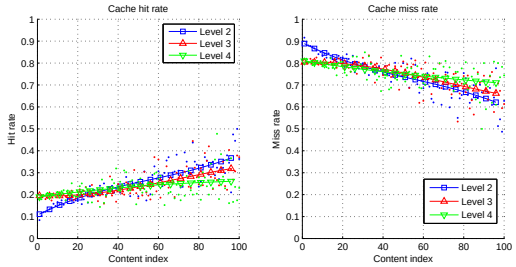
Fig. 7: Number of Interests/Data packets sent/received per topology level, for different cache algorithms: LRU (a), MRU (b). Binary tree topology, $|P| = 25$, $\alpha = 1$, MRU caching.

to a much smaller number of Interest signals relayed to the subsequent 'upstream' levels. This is a result of two features of NDN: (1) the aggregation of Interest signals in PITs; and (2) cache hits in the CSs. Also note the difference in nearly one order of magnitude of the Interests forwarded 'upstream' for both topologies, e.g. at level 2, $\sim 2 \times 10^4$ Interest signals are forwarded for the cascade case, vs. $\sim 1.5 \times 10^5$ in the binary tree topology case. This is obviously a consequence of the larger number of clients contained in the binary tree topology. The quantity of Interests forwarded 'upstream' is generally larger in the case for the MRU cache algorithm, due to the smaller contribution of cache hits (e.g. as seen in Figure 8). The differences between cache algorithms are accentuated in Data packet statistics: e.g. focusing on Figure 6, while for LRU the quantity of Data packets sent by level 2 is practically equal to the number of received Interests, the same does not happen with MRU, with a difference of 4×10^4 to $\sim 2.5 \times 10^4$. This is mostly due to the fact that MRU does not favor the most popular content, whose Interests have to be forwarded much further 'upstream' to be satisfied, resulting in larger accumulations of pending Interests at lower levels. Another interesting phenomena is verified in Figure 7(a), specifically regarding the binary tree topology: note how the quantity of Data packets received at level 1 far exceeds the generated Interest signals. This happens due to the sharing of the interfaces $F_{r,1}$ of the routers at level 1 per each pair of clients, and subsequent duplication of Data signals. Again, the difference is not exactly a 2:1 ratio due to the action of Interest signal aggregation at PITs.

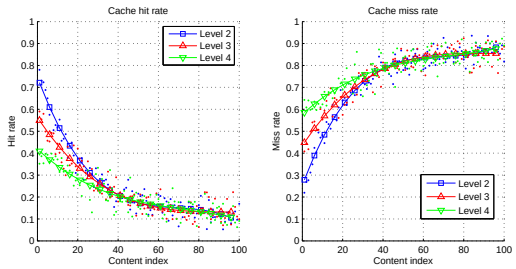
Figures 8 and 9 show the cache hit/miss rates for the different topologies and all considered caching algorithms. We can not a similarity in the trends of the LRU and Random caching



(a)



(b)



(c)

Fig. 8: Cache hit/miss rates for different cache algorithms: LRU (a), MRU (b) and Random caching (c). Cascade topology, $|L| = 5$, $|C| = 1$, $|R| = 3$ and $|S| = 1$, $|P| = 25$, $\alpha = 1$.

algorithms, favoring the most popular content, nevertheless with LRU favoring it in a larger degree. For both topology types, level 2 (the first level after the 'layer' of clients) exhibits significantly larger cache hit rate values for the most popular content than for the subsequent 'upstream' levels. This indicates that CSs at level 2 become mostly occupied by popular content objects, hence explaining some of the results previously analyzed in Figures 6 and 7. Regarding topologies, the differences between Figures 8(a) and 9(a) seem to indicate a larger 'caching capacity' for the binary tree topology, as it presents overall larger hit rate values across all content objects. This is not surprising, due to the larger number of routers (and hence cache capacity) per level. Note, however, that such differences are only verified with the LRU algorithm, being mitigated with Random caching. The MRU algorithm does the opposite, favoring the less popular content and 'flattening' the overall hit rates (at lower values). In fact, these results somehow show the unsuitability of the MRU cache algorithm for the model of Interest signal generation we consider here, i.e. with independent inter-arrival times. The MRU algorithm is mostly indicated for situations in which the inter-arrival times of events are not independent between each other, e.g.

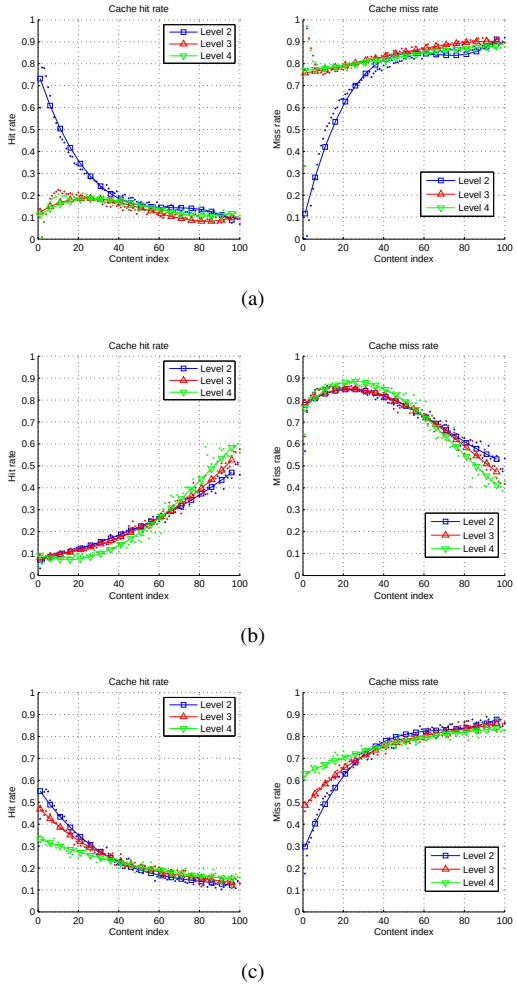


Fig. 9: Cache hit/miss rates for different cache algorithms: LRU (a), MRU (b) and Random caching (c). Binary tree topology, $|L| = 5$, $|C| = 8$, $|R| = 7$ and $|S| = 1$, $|P| = 25$, $\alpha = 1$.

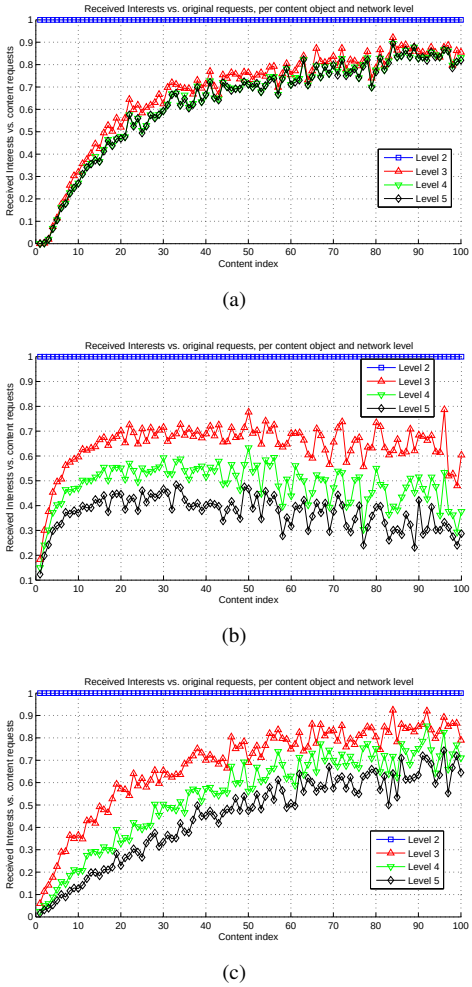


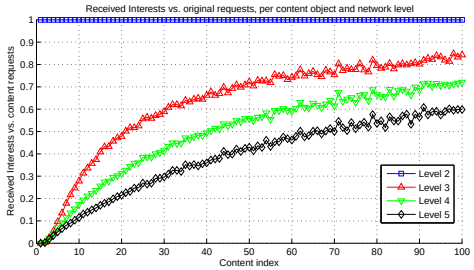
Fig. 10: Received Interests to original requests ratio, per content object and network level. Different cache algorithms — LRU (a), MRU (b) and random caching (c) — cascade topology, $|L| = 5$, $|C| = 1$, $|R| = 3$ and $|S| = 1$, $|P| = 25$, $\alpha = 1$.

in cases where the occurrence of some event e at some point in time indicates that the same event is unlikely to occur in the near future. On the other hand, the LRU algorithm, designed to favor events which are likely to be occur in 'bursts' and concentrated in specific time periods, favors popular content (whose Interest generation occurs more often and therefore exhibiting smaller intervals between occurrences), despite our usage of independent inter-arrival times.

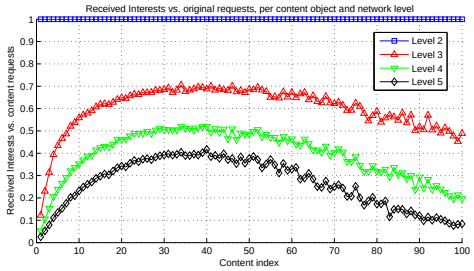
Figures 10 and 11 show the ratio of received Interests to original requests, per content object and network level (including server(s) S), for both topology types considered in the experiments. These charts is provide an idea of the 'latency' a client can expect for the satisfaction of an Interest for some content object, in the sense that it shows how far (i.e. to which level) are Interests forwarded 'upstream'. Overall, and for both topologies, the following aspects can be highlighted:

- All original Interest requests are received by level 2 (the first layer of routers after the clients), as expected;
- For the LRU and Random cache algorithms, the ratio is greatly reduced for the most popular content objects, with values ~ 0 for the LRU algorithm, indicating that these content instances are mostly cached and served at lower levels of the topologies, and therefore accessed with lower latencies;
- For both LRU and Random algorithms, the ratio values increase with the index of content objects, i.e. with the reduction of their popularity, indicating that Interests for less popular content tend to cross more topology levels and are therefore accessed with larger latencies (e.g. in the case of O_{100} the ratio value of level 5 — of the server S — is contained in the interval $[0.5, 0.8]$, meaning that more that 50% of the Interest signals for O_{100} need to actually reach S to be satisfied);
- The ratio values for a given content object tend to decrease with the topology levels, both as a result of (1) aggregation of pending Interests at PITs and (2) cache hits.

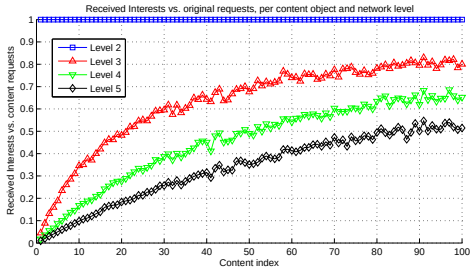
One can note a difference between Figures 10(a) and 11(a), both pertaining to the LRU cache algorithm, but under different topologies: in the former, the differences in ratio values



(a)

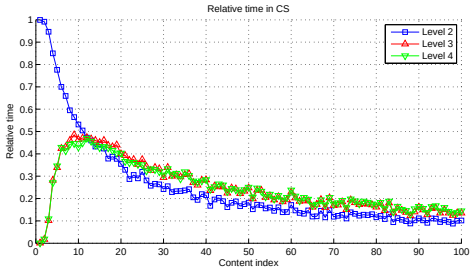


(b)

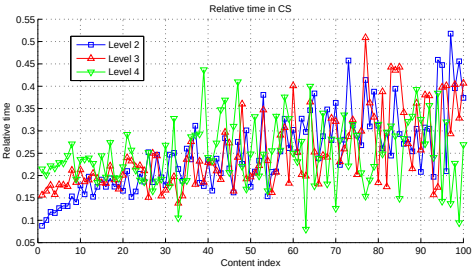


(c)

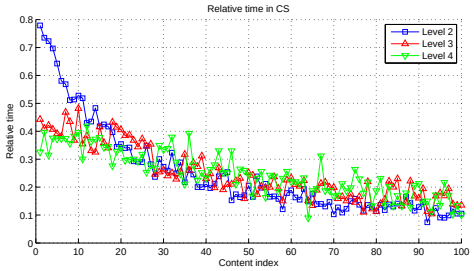
Fig. 11: Received Interests to original requests ratio, per content object and network level. Different cache algorithms — LRU (a), MRU (b) and random caching (c) — binary tree topology, $|L| = 5$, $|C| = 8$, $|R| = 7$ and $|S| = 1$, $|P| = 25$, $\alpha = 1$.



(a)



(b)



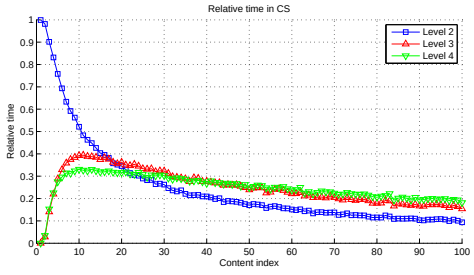
(c)

Fig. 12: Relative time spent at cache per content object. Different cache algorithms — LRU (a), MRU (b) and random caching (c) — cascade topology, $|L| = 5$, $|C| = 1$, $|R| = 3$ and $|S| = 1$, $|P| = 25$, $\alpha = 1$.

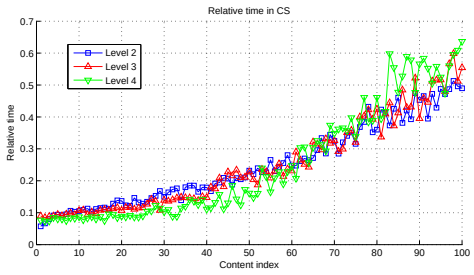
between levels are absent, while in the latter, a clear difference can be identified. The same pattern is evident in Figures 6(a) and 7(a). This behavior can be explained by the larger 'caching capacity' of the binary tree topology.

The results for the MRU cache algorithm (Figures 10(b) and 11(b)) show a trend for benefiting less popular content. The ratio values for the most popular content (O_1 to O_2) are low due to the number of Interests aggregated at PITs at level 2, and not to a high cache hit rate, as previously identified in Figures 8(b) and 9(b). As MRU favors the caching of less popular content, the ratio values decrease with the index of content objects. Also note how the results for the cascade topology are more 'unstable' that those of the binary tree case: this is due to difference in nearly one order of magnitude of the Interests forwarded 'upstream', e.g. at level 2, $\sim 2 \times 10^4$ Interest signals are forwarded for the cascade case, vs. $\sim 1.5 \times 10^5$ in the binary tree topology case.

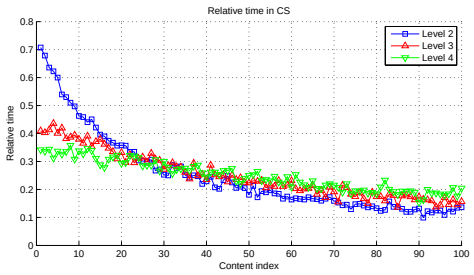
Finally, in Figures 12 and 13, we present the results for our last metric, the relative time spent at CSs, per content object and topology level. Here the differences between cache algorithms are much more evident than those between topologies. In the case of LRU, Figures 12(a) and 13(a), one can verify that content objects O_1 to O_{10} are located in caches for $> 50\%$ of the simulation rounds in level 2, and nearly absent in levels 3 and 4. This seems to accentuate the idea that the exchanges of popular content are concentrated between levels 1 and 2. Objects O_o with $o > 10$ spend more time cached at the subsequent levels, with relative times in the interval $[0.4, 0.1]$. Note how the binary tree's larger 'caching capacity' rises the relative caching time of objects O_o with $o > 10$, as well as a reduction for $1 \leq o \leq 10$ in levels 3 and 4. The results for the Random cache algorithm, Figures 12(c) and 13(c), follow a similar trend to that of LRU, nevertheless with larger relative caching times for content objects O_1 to O_{10} on levels 3 and 4, and a subsequent reduction for level 2. Regarding the MRU caching policy, the benefit of less popular content is also verified in Figures 12(b) and 13(b), as one notes the increase in relative caching time for larger content indexes, without major differences between levels. Furthermore, the results for the binary tree topology seem more 'stable', once again probably due to the due to difference in nearly one order of magnitude of the Interests forwarded 'upstream'.



(a)



(b)



(c)

Fig. 13: Relative time spent at cache per content object. Different cache algorithms — LRU (a), MRU (b) and random caching (c) — binary tree topology, $|L| = 5$, $|C| = 8$, $|R| = 7$ and $|S| = 1$, $|P| = 25$, $\alpha = 1$.

VI. Conclusions

In this work, we have presented the specification details of an Named Data Networking (NDN) [6] simulation model, inspired by the field of nonlinear dynamical systems, which allows the evaluation of different caching policies under different topologies. To do so, we have understood the mechanics of NDN's forwarding engine and proposed models for each of its components, identifying the relevant types of inputs, outputs, state and associated nonlinear dynamics.

To demonstrate the feasibility of the proposed model, we implemented the given specifications in MATLAB®, providing simulation results with three simple cache algorithms, specifically (1) Least Recently Used (LRU), (2) More Recently Used (MRU) and (3) Random caching. The results obtained after simulation allow one to identify some of the key features claimed by the NDN architecture [6] — in a way asserting the validity of the proposed model — as well as specificities of the considered cache algorithms. We believe the proposed model specification and implementation to be of particular interest for designers wishing to perform preliminary evaluations of caching policies under NDN. Even though the purpose of the proposed model is not to strive for a truly realistic scenario, as future enhancements we predict the introduction of packet loss, network congestion, among other parameters, in order to make it more robust.

References

[1] Andrea Passarella. A Survey on Content-Centric Technologies for the Current Internet: CDN and P2P solutions. *Computer Communications*, 35(1):1–32, 2012.

[2] M Handley. Why the Internet Only Just Works. *BT Technology Journal*, 24(3):119–129, 2006.

[3] J Pan, S Paul, and R Jain. A Survey of the Research on Future Internet Architectures. *Communications Magazine, IEEE*, 49(7):26–36, July 2011.

[4] George Xylomenos, Christopher N. Ververidis, Vasilios a. Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V. Katsaros, and George C. Polyzos. A Survey of Information-Centric Networking Research. *IEEE Communications Surveys & Tutorials*, PP(99):1–26, 2013.

[5] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A Data-Oriented (and Beyond) Network Architecture. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 181–193, 2007.

[6] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking Named Content. *Proceedings of the 5th international conference on Emerging networking experiments and technologies CoNEXT 09*, 178(1):1–12, 2009.

[7] Dirk Trossen and George Parisis. Designing and Realizing an Information-Centric Internet. *IEEE Communications Magazine*, 50(7):60–67, 2012.

[8] Dipankar Raychaudhuri, Kiran Nagaraja, and Arun Venkataramani. MobilityFirst : A Robust and Trustworthy Mobility- Centric Architecture for the Future Internet. *ACM SIGMOBILE Mobile Computing and Communications Review*, 16(3):2–13, 2012.

[9] Dongsu Han, Peter Steenkiste, Michel Machado, and David G Andersen. XIA : Efficient Support for Evolvable Internetworking. In *The 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, pages 1–14, 2012.

[10] Christian Dannewitz, Dirk Kutscher, BöRje Ohlman, Stephen Farrell, Bengt Ahlgren, and Holger Karl. Network of Information (NetInf) - An Information-Centric Networking Architecture. *Comput. Commun.*, 36(7):721–735, April 2013.

[11] J. K. Hedrick and A. Gira. *Control of Nonlinear Dynamic Systems: Theory and Applications*. 2010.

[12] Cheng Yi, Alexander Afanasyev, Lan Wang, Beichuan Zhang, and Lixia Zhang. Adaptive Forwarding in Named Data Networking. *ACM SIGCOMM Computer Communication Review*, 42(3):62–67, June 2012.

[13] G Carofiglio, M Gallo, L Muscariello, and D Perino. Modeling Data Transfer in Content-Centric Networking. In *Teletraffic Congress (ITC), 2011 23rd International*, pages 111–118, September 2011.

[14] Theodore Johnson and Dennis Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 439–450, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[15] Hong-Tai Chou and David J DeWitt. An Evaluation of Buffer Management Strategies for Relational Database Systems. In *Proceedings of the 11th International Conference on Very Large Data Bases - Volume 11*, VLDB '85, pages 127–141. VLDB Endowment, 1985.