Cache Freshness in Named Data Networking for the Internet of Things

Maroua Meddeb[1,2], Amine Dhraief[1], Abdelfettah Belghith[4,5], Thierry Monteil[2,3], Khalil Drira[2], Saad AlAhmadi[4]

[1] *HANA Lab, Univeristy of Manouba, Tunisia*
[2] *LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France*
[3] *Univ de Toulouse, INSA, F-31400 Toulouse, France*
[4] *College of Computer and Information Sciences, King Saud University, Saudi Arabia*
[5] *Corresponding Author*
Email: abelghith@ksu.edu.sa

The Information-Centric Networking (ICN) paradigm is shaping the foreseen future Internet architecture by focusing on the data itself rather than its hosting location. It is a shift from a host-centric communication model to a content-centric model supporting among others unique and location-independent content names, in-network caching and name-based routing. By leveraging the easy data access, and reducing both the retrieval delay and the load on the data producer, the ICN can be a viable framework to support the Internet of Things (IoT), interconnecting billions of heterogeneous constrained objects. Among several ICN architectures, the Named Data Networking (NDN) is considered as a suitable ICN architecture for IoT systems. However, its default caching approach lacks a data freshness mechanism, while IoT data are transient and frequently updated by the producer which imposes stringent requirements in terms of information freshness. Furthermore, IoT devices are usually resource-constrained with harsh limitations on energy, memory and processing power. We propose in this paper a caching strategy and a novel cache freshness mechanism to monitor the validity of cached contents in an IoT environment while minimizing the caching process cost. We compared our solution to several relevant schemes using the ccnSim simulator. Our solution exhibits the best system performances in terms of hop reduction ratio, server hit reduction ratio and response latency, yet it provides the lowest cache cost and significantly improves the content validity.

Keywords: ICN; NDN; IoT; In-network caching; Freshness; Cache cost

1. INTRODUCTION

The rapid rise in the number of small embedded devices along with the improvement of wireless technologies enabling objects to communicate, compute and coordinate form the foundation of the so-called Internet of Things (IoT). Despite the diversity of the research on IoT, its definition remains fuzzy [1]. Most of the definitions agree on the fact that it can be defined as a global network that interconnects smart devices able to sense and react according to their environmental situations. This innovation is expected to significantly increase the number of contents carried by the network. However, unlike traditional Internet hosts, IoT devices are resource-constrained in terms of memory, energy and processing power that require a scalable architecture to support data dissemination in the network.

IoT data are usually small and frequently transient compared to Internet contents which are large and time-invariant[1]. IoT applications impose stringent requirements on some critical domain application such as Smart Home. Furthermore, they are looking for information rather than point-to-point communications. Facing these distinct particularities of IoT systems, the research community has been actively working on the adoption of a new networking protocol stack that effectively supports IoT communications.

The current Internet communication model is built according to a host-centric view of the network. Whenever a source node requests a specific content, it has first to fully locate and identify the node on which this content is stored. It is worth noting that this paradigm has survived for the last four decades. Nowadays, it has inexorably reached its limits. Actually, current nodes are equipped with several network interfaces bound to distinct access technologies. With the democratization of the IPv6 protocol, each interface has its own global network address. Besides, the Internet is no more an

interconnection/concatenation of disjoint networks, but has become an overlay network where users share contents. Internet traffic pattern has mutated from the classical "one-to-one" to the dominant "one-to-many" and "many-to-many". In addition to the Internet evolution, its current architecture has some limits towards efficient and scalable content distribution, mobility and security. Regarding these problems, the Internet requires, nowadays, to be redesigned for its healthy and sustainable future growth.

This issue has attracted the attention of many researchers [1]. For instance, Van Jacobson et al. have taken the initiative in [2] to address this subject. They have introduced a novel paradigm, namely Information-Centric Networking (ICN). The latter constitutes a shift from a host-centric view of the network to a content-centric one. According to this new paradigm, the user requests a content by its name instead of using its network localization. In ICN, every content is identified by a unique, persistent and location-independent name. ICN provides native multicast support, in-network caching, name-based routing and easy data access. As a result, ICN has the potential to become the key technology for data dissemination in IoT networks.

Such issue has already been investigated by several studies [3], [4], [5], [6], [7], [8] and [9]. In particular, the in-network caching enhances the producer data access since data can be provided by any intermediate node. Consequently, data caching speeds up the data retrieval, and reduces the network traffic. Hence, ICN leads to a low dissemination latency, a considerable reduction of the bandwidth as well as a decrease in the energy consumed by producers which are resource-constrained IoT devices.

Many test beds were proposed in the literature to evaluate ICN solutions. We cite Open Network Lab, PlanetLab, Emulab, Deter, NEPI, PURSUIT Blackadder test bed and JGN-X [10]. Other test beds are currently used for ICN experiments on IoT, which are CCN-lite on top of the RIOT operating system [4] and IoT-LAB [11].

The caching concept, which is not a revolutionary term, has been widely used on the Web, P2P systems and Content Distribution Networks (CDNs). However, in ICN, the in-network caching is more prominent and more challenging than the already existing caching systems. First, it is transparent and does not require any specific application to cache content. Second, it is ubiquitous since any ICN nodes can be a cache[12]. It is worth-noticing that in-network caching only provides the last version of sensed data so it may be unsuitable for applications requiring previous measurements as is in the case of Health applications. In our study, we rather consider the Smart Home scenario and we use real smart building data.

Several studies have already addressed the in-network caching in ICN. However, we claim that the proposed caching strategies are not suitable for IoT systems due to the stringent requirement in term of data freshness. Since cached copies can be out of date, a freshness mechanism must be adopted to check the validity of the retrieved data. This problem is called the cache freshness (coherence/validity) [13]. We notice that only few work addressed the in-network caching in IoT[14], [4].

Several ICN approaches have been proposed, such as DONA, NDN, NetInf, COMET, CONVERGENCE, Mobility First and PRISP. We refer the reader to [15] and [16] for a general survey on ICN. The so-called Named Data Networking (NDN) is considered as the most suitable ICN architecture for IoT systems [17], [8], [4], [6], [18], [19], [20] and [21]. It defines a receiver-driven, pull-based, robust connection-less communication model. These features are beneficial for the IoT systems in terms of easy and scalable data access, energy efficiency, security and mobility support[22]. We adopt, in our study, the NDN approach.

The use of NDN in an IoT environment must take into consideration two major issues. Firstly, IoT devices are heavily constrained in terms of memory, battery and computation power, and as such we must introduce non-costly caching strategies while maintaining good system performances. Secondly, data freshness is paramount since IoT contents are transient and frequently updated.

The contribution of this paper is twofold. We first propose a caching strategy, named *consumer-cache* that caches content close to consumers. Second, we design a freshness mechanism termed *event-based freshness* which is based on the time prediction of IoT traffic. In this context, we aim to improve existing in-network caching strategies, including our proposed strategy, in order to support event-based freshness mechanism. By comparing the latter to the existing freshness mechanisms, we could demonstrate that our freshness mechanism significantly improves the content validity with all caching strategies.

The remainder of the paper is organized as follows: we overview, in section 2, the in-network caching in ICN. Furthermore, we discuss the principal studies of the research community in ICN approaches that targeted IoT systems. Section 3 describes our proposed scheme and presents a concrete use case. We present and analyze simulation results in section 4, before discussing the results and concluding the paper in section 5 and 6.

2. BACKGROUND

2.1. ICN in-network caching mechanisms

In this section, we will first focus on in-network caching features, namely, in-network caching strategies end freshness mechanisms. Then, we will introduce some related work about in-network caching in ICN-IoT systems.

A major building block of ICN concerns the in-network caching. The latter is the most common and important feature of ICN architectures. It was introduced to alleviate the pressure on the network bandwidth and consequently improves the efficiency of contents dissemination[12]. However, the in-network caching decision is affected by various aspects including cache size, cache decision, cache eviction policies, cache freshness and availability of cached contents. The cache eviction is performed once the cache is full. Old contents are deleted, according to a policy, to allow storing new ones. It is also worth noticing that the cost of in-network caching is somewhat high in terms of larger overhead and complexity, which is unsuitable for IoT systems in the presence of resource-constrained devices.

The in-network caching is classified into two scenarios: *On-path* and *Off-path* caching. With the On-path scenario, cache nodes belong to the request path. When the requested content is forwarded to the consumer, nodes on the requested path may choose to cache the content according to their specific caching policy. We will detail different on-path caching strategies in the following subsection. In the Off-path caching scenario, cache nodes are rather fixed in advance within the topology. In fact, each producer has an associated cache element known in advance. Producer forward the content to both the consumer and the associated cache [23]. In this paper, we solely focus on the on-path caching.

2.1.1. Existing on-path caching strategies

In order to improve the cache diversity, and reduce cached contents redundancy, several caching strategies have been proposed[12]. At a given node, the decision to cache or not a content is made based on a caching strategy. We quote some of the proposed in-network caching strategies. The most known is Leave Copy Everywhere (LCE) [24] in which each nodes, belonging to the path towards the consumer, keeps a copy of the content. While, with Leave Copy Down (LCD) [24], only the node on one level down in the reverse path towards the consumer stores a copy. Probabilistic cache (ProbCache) [25] privileges caching close to consumers. In fact, the caching process is performed with a changing probability inversely proportional to the distance between the consumer and the producer. The Betweenness Centrality (Btw) [26] strategy is based on a pre-calculated parameter for each node. This parameter measures the number of times that a node belongs to a path between all pairs of nodes in a network topology. When the request is sent along a path, the greatest betweenness centrality of the traversed nodes is stored. Then, it is appended to the data message. When the response is sent towards the consumer, a copy of the content is only cached in nodes having the highest value. Finally, the Edge-caching strategy[27] is proposed for hierarchical topology and caches contents at the topology leaves.

2.1.2. Existing freshness mechanisms

Cache freshness maintains the validity of the shared contents stored in multiple caches. As an immediate consequence, consumers can trust copies in caches. A copy is considered valid when it has the same version as the source, or when the difference between the data in the source and its copy in the cache is not important. For example, for a home automation usage, a cached temperature value of 23° is considered valid when the ambient temperature is to 24°.

In [13], Dingle et al., have focused on the web cache coherence. The cache freshness problem in both the Web and ICN remains the same. This issue is also addressed in the area of distributed systems. Maintaining freshness is easier in Web caches than in distributed files because these latter support distributed writes. Whereas, updating Web contents is performed only by the content's producer. In the sequel, we introduce the representative freshness mechanisms used by popular distributed file systems. These mechanisms can be classified into two major classes. First, the *Validation check* [28] mechanism in which the validity of the requested content is checked with the time-stamp (caching time). The time-stamp is sent to the producer to check if the data has not been modified since it was cached. Second, the *Callback* mechanism [29], in which all cache nodes are notified by the producer whenever an update occurs. In this case, the producer must have the list of all content's caches, which is unsuitable.

Concerning the web-freshness mechanisms, there are four major approaches. First, the HTTP Headers[30] mechanism, in which an If-Modified-Since header is appended to the request in order to check if the last modification time of the content is greater than the If-Modified-Since time in the request header. This is called a conditional GET. Second, the "Naive" coherence[30] which is proposed for hierarchical topology. This mechanism is a particular case of HTTP headers. In fact, to check the content validity in a cache, the latter sends a conditional GET message only to the next higher cache or server. Third, the Expiration-based freshness[31] mechanism. In this one, each content is marked with an expiration time. The content is assumed valid as long as the expiration time has not elapsed. Fourth, the pre-fetching[32] mechanism consists in periodically refreshing the cached contents. This is performed by sending conditional GET requests to check if the content has been changed. The main disadvantage of this mechanism is the decision of contents refreshment time. This decision depends on the content age and popularity. In some cases, it may be useless to refresh it.

2.2. Related Work

The deployment of IoT systems enabling ICN features has started to gain momentum within the research community. An interesting contribution in this field was proposed by Pentikousis et al., in [3]. The authors presented several ICN baseline scenarios including IoT applications.

In [4], authors showed that ICN can be applicable in IoT, and is efficient in terms of energy consumption and memory costs. The results presented in [4] showed that the number of radio transmissions is drastically reduced thanks ti in-network caching. This work constitutes the first study of ICN in a real IoT deployment. The authors defended the NDN architecture that is very effective in IoT scenarios. They also proved that this architecture satisfies IoT requirements.

The beneficial effect of the named data on IoT is identified in [5] by Heidemann et al. Authors discussed that content naming is considered as the building block of ICN by making the information easily and uniquely identified in the network. Moreover, they showed that naming, as one of the ICN features, makes the deployment of IoT in ICN more effective.

Authors in [6], defined NDOMUS (Named Data netwOrking for sMart home aUtomation) as a framework based on the NDN-IoT architecture tailored to the smart home domain. They presented several use-cases. They also evaluated the proposed framework by calculating the transmitted packets number.

As discussed in [7], using the already existing ICN caching mechanisms leads to a considerable reduction in the consumed energy and bandwidth. Authors showed that as the available caching storage capacity increases, the consumption of both energy and bandwidth decreases.

Hail et al., in [8] proposed a novel distributed probabilistic caching strategy named probabilistic CAching STrategy for the INternet of thinGs (pCASTING) that considers data freshness and potentially constrained capabilities of IoT devices. The pCASTING strategy adjusts the caching probability by considering the battery energy level, the cache occupancy and the data freshness. In such a way, fresher data have priority in the caching decision. The freshness is evaluated by a parameter that measures how many seconds the content is considered as *valid* in the cache. Results show the effectiveness of the proposed scheme in terms of energy consumption and content retrieval delays.

The work presented in [9] addresses the information freshness in cache nodes. Authors proposed a freshness parameter. Unlike the defined parameter in [8], the suggested one is not variable. It can be adjusted by consumers to specify their particular freshness requirements. Simulation results show that the received data validity has been improved.

The work of [14] is the first study addressing the in-network caching of IoT data at content routers on the Internet. Authors of this work discussed the trade-off between the multi-hop communication cost and the freshness of a requested data. In this context, they introduced a distributed probabilistic caching strategy in which cache nodes dynamically update their caching probability. This probability depends on the distance between the producer and the consumer as well as on the data freshness in a way that the closer the caching location is to the producer, the fresher the retrieved data packet is. Two results were observed. First, the more popular the content is, the higher the cost reduction will be. Second, when caching nodes are closer to consumers, IoT data put a lower workload on the network.

To the best of our knowledge, [33] was the first attempt studying the in-network caching of transient data in IoT systems. In this study, the authors aimed to prove the efficiency of in-network caching in IoT even with transient data. To this end, they quantified caching gains with transient items by introducing an analytical model to capture the trade-off between communication cost and data freshness. In their simulations, they assumed that each data has a known lifetime. Through simulations, they showed the benefit of using Internet routers to cache transient data generated by IoT applications.

Few studies considered the ICN approach in specific IoT application domains such as Vehicle-to-Vehicle (V2V) communications[34], Wireless sensors networks [35],[36],[37],[38], Smart Home [6] and Smart Grid [39].

3. A COHERENT CACHING STRATEGY FOR A NAMED DATA IOT NETWORKS

In this section, we present our proposed freshness-aware caching strategy for NDN IoT networks introduced in our previous work[21]. Our main objective is, first, to reduce the caching cost while maintaining the system performances in terms of hop reduction and server hit reduction ratios. Second, we aim to enhance the percentage of the freshness of the requested content. For this purpose, we suggest an in-network caching strategy that privileges to cache at gateways close to the consumers, and a cache freshness mechanism dedicated to the IoT traffic pattern.

Before going into the details of our contributions, we need first to understand the IoT architecture. Until now, there is still no consensus on the physical model of the IoT architecture to adopt. Two divergent visions are tacitly used in various IoT experiments. The first considers the IoT as a collection of sensors/actuators interacting with each other without any intermediate equipment. The second vision assumes that actuators and sensors communicate through their respective gateways. In an IoT scenario where actuators and sensors directly communicate with each other, an actuator directly requests a sensor to get the measured value by the latter. This scenario assumes that the

actuators and sensors are provided with an operating system, software applications and memory. In the second model, gateways play a key role. In fact, actuators, as well as sensors, are directly connected thereto. Contrary to the first view, the measured value by a sensor is no longer stored in the memory of the latter but in the gateway to which it is attached.

Starting from the premise that IoT devices are resource-constrained, caching data, as small as they are, in a sensor memory, will be the determinant of embedded applications and operating systems. In addition, the caching decision is made by the CPU of the sensor which decreases the battery lifetime. Finally, the cached data retrieval will consume valuable sensor's radio resources. For these reasons, we propose to adopt, in this paper, an NDN IoT architecture where data are cached on the gateways and sensors represent the producers.

3.1. Consumer-cache strategy

In this section, we detail our first contribution: *consumer-cache*. Consumer-cache is an (i) on-path caching strategy (ii) which targets to reduce the number of caches nodes and (iii) privileges gateways directly linked to consumers.

In an IoT context, we believe that it is preferable to distribute caches on the network topology (on-path) rather than centralizing them on fixed caching nodes (off-path). With an important number of equipment and large data flows, specifying particular gateways to handle caching (even locally) would have as an immediate consequence the creation of multiple congestions within the network. In addition, the off-path caching increases the signaling overhead since it requires, in the caching process, additional packet transmissions to the cache node.

W.K.Chai et al., showed in [26] that caching in fewer nodes can perform better results in terms of producer hits and number of the traversed hops to reach a content. Even caching in one randomly selected cache can be better than the LCE strategy. With LCE caches fill up very quickly because the cache size is very limited compared to the number of the contents to be cached. Once the cache is full, a replacement process is triggered to allow caching new items. The more frequent is the replacement process, the least efficient is the caching, as contents may be cached and then deleted without ever being used which induces an increased caching cost. In addition, IoT nodes are memory and energy-constrained, consequently, it is more advantageous to minimize the resource consumption. As a consequence, the second directive aims to reduce the number of caches. It should be emphasized that this does not mean that lowering the number of cache nodes provides a better strategy. It is a trade-off between the number of evictions, the cache cost and the data availability.

From this point of view, we need to know what is the selection criterion of the placement of cache nodes in a topology. The results obtained in [27] show that edge-caching saves in-network caching benefits. Recall here that the edge-caching strategy is originally conceived for hierarchical topologies. Under this strategy, caches are located at the leaves of the topology. Results in [27] proved that the most efficient nodes for caching are located at the edges of the topology. Since consumers are generally located at network edges and in some cases in the middle of paths, we propose as the third directive to cache at nodes close to consumers.

Based on these three directives, our caching strategy, termed *consumer-cache*, proposes to store a copy of contents on on-path nodes which are connected to a consumer. We can say that if all nodes in the topology are connected to consumers, consumer-cache strategy tends towards LCE strategy. Moreover, if consumers are only connected to edges, our strategy will be similar to edge-caching strategy. Our strategy strictly depends on the number of consumers and their locations. We present below the algorithm of the consumer-cache strategy (Algorithm 1). It is assumed that the request has already arrived at the producer, which will send the desired content to the consumer.

Data:
$Data$ to cache
$Prod$ = the *producer* node
$Cons$ = the *consumer* node
$G = (V, E)$ a graph that represents the network
$Path = (v_1, v_2, ...v_n) \in V * V * ... * V$ s.a:
v_i adjacent to v_{i+1} for $1 \leq i \leq n$ and
$v_1 \longleftarrow Prod$
$v_n \longleftarrow Cons$
Result:
The set of selected *nodes* as a cache
$Caches = (v_c, ..., v_p) \in V * V * ... * V$ s.a:
$1 \leq c \leq p \leq n$
$Caches \subseteq Path$
for *Each* $nodev_i \in Path$ **do**
  if $\exists$ *a consumer connected to* $nodev_i$ **then**
    Caches = v_i Cache($Data$,v_i)
  end
end

Algorithm 1: *Consumer-cache strategy*

Let us take the example of the topology presented in Figure.1 where $Consumer1$ requests a content named (c). Initially, all caches are empty. So, the request is forwarded to the producer. $Producer1$ holds the content (c). The latter sends the response via the path from $n11$ to $n0$. While forwarding the response, each of the nodes in the response path has to decide whether to cache a copy of (c) according to its own caching strategy. With LCE, all the 10 nodes in the path store a copy. Other strategies require that the content should be stored in only a single node. For LCD, only $n11$ retains a copy; whereas with Btw the cache is $n6$. However,

under consumer-cache, caches are $n7$ and $n0$. For ProbCache, $n0$ has the greatest probability to be the cache. Let us now assume that $Consumer4$ requests the same content (c). It sends a request towards $Producer1$. With Btw and probably ProbCache, we have a cache miss. Under the ProbCache strategy, the probability to find the content in $n7$ is very low as this node is far from Consumer1. Using consumer-cache and LCE the request is satisfied by $n7$.

Using edge cache, the request is satisified by a leave router (n1). With Edge caching strategy, $n0$, $n1$ and $n2$ are selected as caches. Requests issued by $Consumer1$, $Consumer2$ or $Consumer3$ are directly satisfied. However, requests issued from $Consumer4$ are satisfied by $n1$ which is 5 hops away from $Consumer4$. In the case of LCE, the cached copies at $n0$, $n3$, $n4$, $n5$, $n6$, $n8$, $n9$, $n10$ and $n11$ are redundant. It is therefore clear that caching content close to consumers is sufficient to avoid redundancy in other caches.

3.2. Event-based freshness mechanism

To address the cache coherence problem related to different in-network caching strategies, we conceive a freshness mechanism which we call *event-based* freshness. The mechanism process depends on the data flows categories. Being in the case of IoT networks, we first detail the different IoT traffic patterns.

3.2.1. The IoT traffic patterns

IoT traffic is usually classified into four categories: continuous, periodic, OnOff and request-response transmissions[40]. With the continuous transmission, the data are transmitted in a continuous manner like video streaming. Under periodic transmission, the source sends a data at every fixed period of time. For example, with a temperature sensor, after the elapse of each period (e.g: 1 hour), a new value is recorded. The OnOff transmission mode stipulates that the content is updated as soon as new data is sent. Let us consider the example of a presence sensor. The value 0 of the sensor indicates the absence of persons. Once someone is in the room, the value is updated to 1. Finally, in the request-response transmission, as its name indicates, the consumer sends directly a request to get the current value of the sensor. By considering these transmission modes, we can assume that sensors can be passive and do not support any behavior as it is the case in the request-response transmission. Otherwise, sensors can be active with a periodic, OnOff or continuous behavior. Considering the continuous transmission as a periodic one with a tiny period(ϵ), IoT events are summed up to periodic and OnOff mode.

3.2.2. Event-based freshness algorithm

The event-based freshness algorithm is an expiration-based freshness with a variable expiration time. It depends on producer behaviors, the instant of storing a content in a cache called cache-time, the period in the case of periodic mode, and past events in the case of OnOff mode. The pseudo-code for forwarding the request and checking the data freshness is given in Algorithm 2.

Data:
$Data$ to retrieve
$Prod$ = the *producer* node
$Cons$ = the *consumer* node
$G = (V, E)$ a graph that represents the network
$Path = (v_1, v_2, ...v_n) \in V * V * ... * V$ s.a:
v_i adjacent to v_{i+1} for $1 \leq i \leq n$ and
$v_1 \longleftarrow Cons$
$v_n \longleftarrow Prod$
Result:
The Fresh $Data$ to retrieve
for *Each node* $\in Path$ **do**
  if *Data in cache* **then**
    if $Data.flow =$ "$Periodic$" **then**
      Calculate the lifetime of the last version in the source
      $T_{life} = Exp(1/T)$
      if $T_{life} > Now - cache_time$ **then**
        Get($Data$)
      end
      else
        Delete($Data$)
      end
    end
    else if $data.flow =$ "$OnOff$" **then**
      if $T_{event} > Now$ **then**
        Get($Data$)
      end
      else
        Delete($Data$)
      end
    end
  end
end

Algorithm 2: *Event-based freshness*

As we can see in the algorithm, the check process is performed when a request reaches a cache. To verify the content validity, we compare the lifetime of the content in the cache with its lifetime in the producer to check if the cached content has been modified in its source. To calculate the lifetime of a cached content, we use the *cache_time* value. The difference between the current time Now and the *cache_time* measures how long a content was cached. On the other hand, to calculate the lifetime of the last version in the source, we focus on the transmission mode, namely periodic or OnOff flow. In fact, in the case of periodic transmissions with a period T, to calculate the lifetime T_{life} in the source, we use the residual life paradox. The residual life of an exponentially distributed variable is also exponential with the same rate. The period T must be then

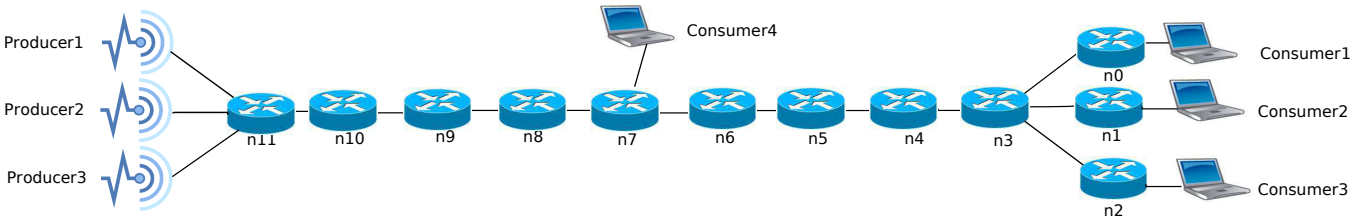


FIGURE 1. An example of a Transit-Stub path

appended to the content. If T_{life} is greater than the difference between the current time and the *cache_time*, the content is returned to the consumer.

With an OnOff transmission, the procedure is different because the updating time in the source is not predicted, and every producer has a specific behavior. To face this problem, we propose to use a prediction method to estimate the time of the occurred events T_{event}. Prediction process is based on past events. For this reason, we propose to store, in the source, all past events in a list *lastEvents*. Using our mechanism of validity, producers do not need to be synchronized. In fact, in our algorithm, we proposed to save at the producer level the time elapsed between two successive events instead of saving the time when the event took place. We take the example of a producer who had two events at 10 am then at 11 am. In this case, the time elapsed between these two successive events is 1 hour. Therefore, we save this value (1 hour) in this producer's *lastEvents* list.

When a request arrives at the producer, we calculate an estimate of the time for the next event to happen based on the values in the *lastEvents* list (for example the calculated value indicates that the next event will take place in 30 minutes). This estimated value is added as *metadata* to the response to say that this content is only valid for 30 minutes. On the return path, content can be cached in any node. The metadata is also saved in the cache node with the copy of the content as T_{event}. T_{event} is the *cache_time* plus the *metadata* value.

When a copy is found in a cache, if T_{event} is greater than the current time, then we can presume that the content has been updated in the source, and this copy is considered invalid. Such decision is not always right given that it is founded on estimations.

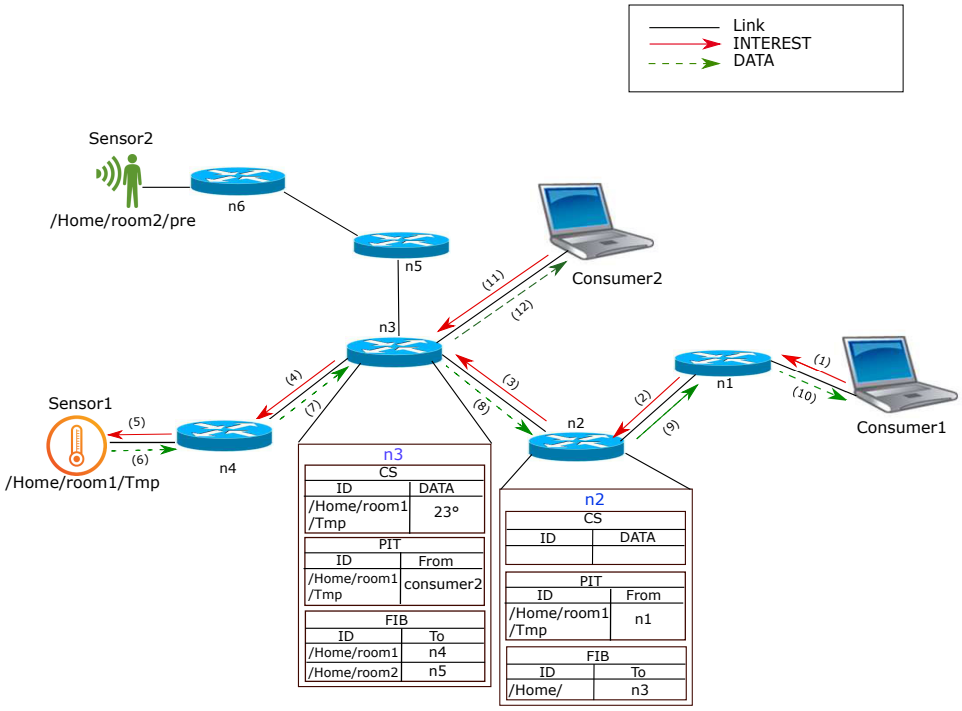
We use time series for the prediction process. Time series data is analyzed in order to extract a meaningful statistic model used to predict future values based on the ones previously observed. Several time series models exist such as the ARMA (Autoregressive Moving Average) model or the exponential smoothing. The exponential smoothing uses all past events to make the prediction, while ARMA utilizes only the k past data points. In an IoT context, we always prefer to put less information in equipment. As a consequence, in our event-based freshness mechanism, we choose to use the ARMA model [41].

3.3. A scenario of Named Data IoT Networks

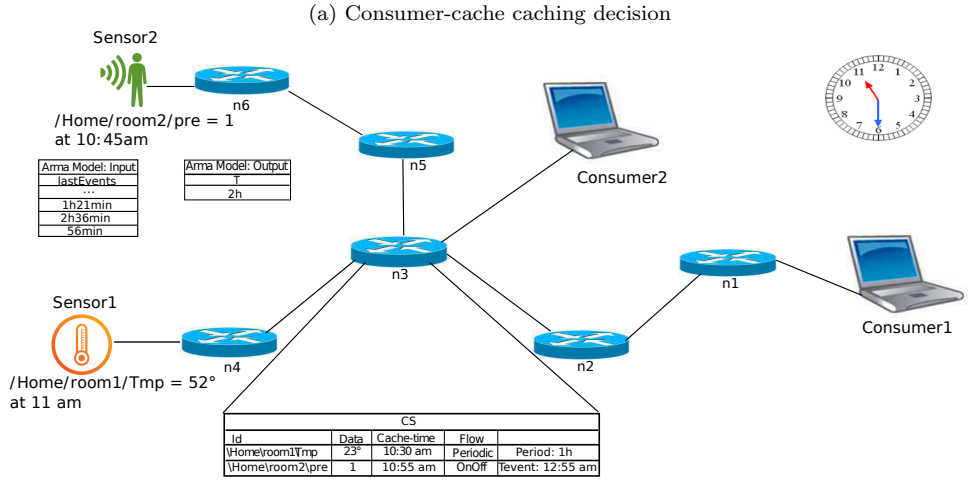
We describe in Figure.2 a scenario under the Named Data IoT Networks using consumer-cache caching strategy and event-based freshness mechanism. NDN is based on three system elements: Content Store (CS), Pending Interest Table (PIT) and Forwarding Information Base (FIB) table. Consumers issue *Interest* messages to request information objects. The request is routed by name. CS acts as a cache in which the received data are cached. When a request hits a cache, a CS lookup is performed. In the case of a cache miss, the *Interest* packet is sent to the next hop according to the FIB, and the interface of the incoming *Interest* is appended to a set of interfaces interested by that chunk in the PIT. The FIB contains, for each content, an entry that points to the right output interface. In the case of a cache hit, the corresponding *Data* is sent back according to the list of interfaces stored in the PIT. When the request is satisfied, the corresponding entry in the PIT is then removed. NDN approach natively supports on-path caching using the CS. It also supports multicast channels by allowing *Interest* messages to be cached in intermediate hosts. If a new *Interest* message, requesting the same name, reaches a host, it will be pending until receiving the *Data* packet. Then, the host will directly return the *Data* to both requesters.

In Figure.2a, $Consumer1$ sends an *Interest* packet for the content named $/Home/room1/Tmp$ (arrows 1-5). When the node $n2$ receives the *Interest* message, it first performs a lookup in its CS. If there is no entry with this name, the request is forwarded to the next hop according to the node's FIB. This base contains an entry which points to node $n3$. The router records the request's incoming interface $n1$ in the PIT. Then, it sends the *Interest* to $n3$ and so on. In the case of a cache miss, the producer satisfies the request. A *Data* message is then returned. This packet is forwarded based on states stored in PITs (arrows 6-10). Under the consumer-cache strategy, only $n3$ and $n1$ store the object in their CSs. Now, we suppose that $Consumer2$ asks for the same content $/Home/room1/Tmp$ (arrow 11). When the *Interest* packet arrives to node $n3$, it matches the information object found at the CS. Then, the *Data* is directly returned (arrow 12).

We show, in Figure.2b, the operation of our freshness mechanism. The event-based mechanism introduces



(a) Consumer-cache caching decision



(b) Event-based freshness mechanism

FIGURE 2. An example with the NDN architecture using consumer-cache caching strategy and event-based freshness mechanism

new parameters which are *cache_time*, T_{event}, the period T and the *flow* (Periodic or OnOff). The *cache_time* parameter is stored in the cache structure in the gateway when a new item is cached. However T_{event} in the case of an OnOff flow and T in the case of a Periodic flow are sent as metadata with the content and then stored in the Content Store structure as shown in figure Fig. 2b. We consider two producers, a temperature sensor and a presence sensor. When the *Interest* packet reaches node $n3$, a cache lookup is performed. Then, the content is found in the CS. The use of the event-based freshness consists in verifying the cached object before returning it. It is assumed to be 11:30 am. The freshness of a data depends on its flow. In the case of periodic transmission, when the period elapses, the content is no longer valid. In our scenario, the caching is performed at 10:30 am. After 11:30 am, the temperature value 23° is considered invalid. Consequently, this entry is removed from the CS, and the request is forwarded to the next hop to look for a valid data. In the case of OnOff transmission, the time of the next event is calculated in the source and is appended in the *Data* packet. As we have already mentioned, the time series model ARMA is used for this prediction based on past events. Calculations assume that the next event will be after 2 hours. When an *Interest* packet reaches the node $n3$, T_{event} is calculated ($T_{event} = cache_time + 2hours = 12:55am$) and compared to the current time. Based on this comparison, we admit that the data is fresh since it

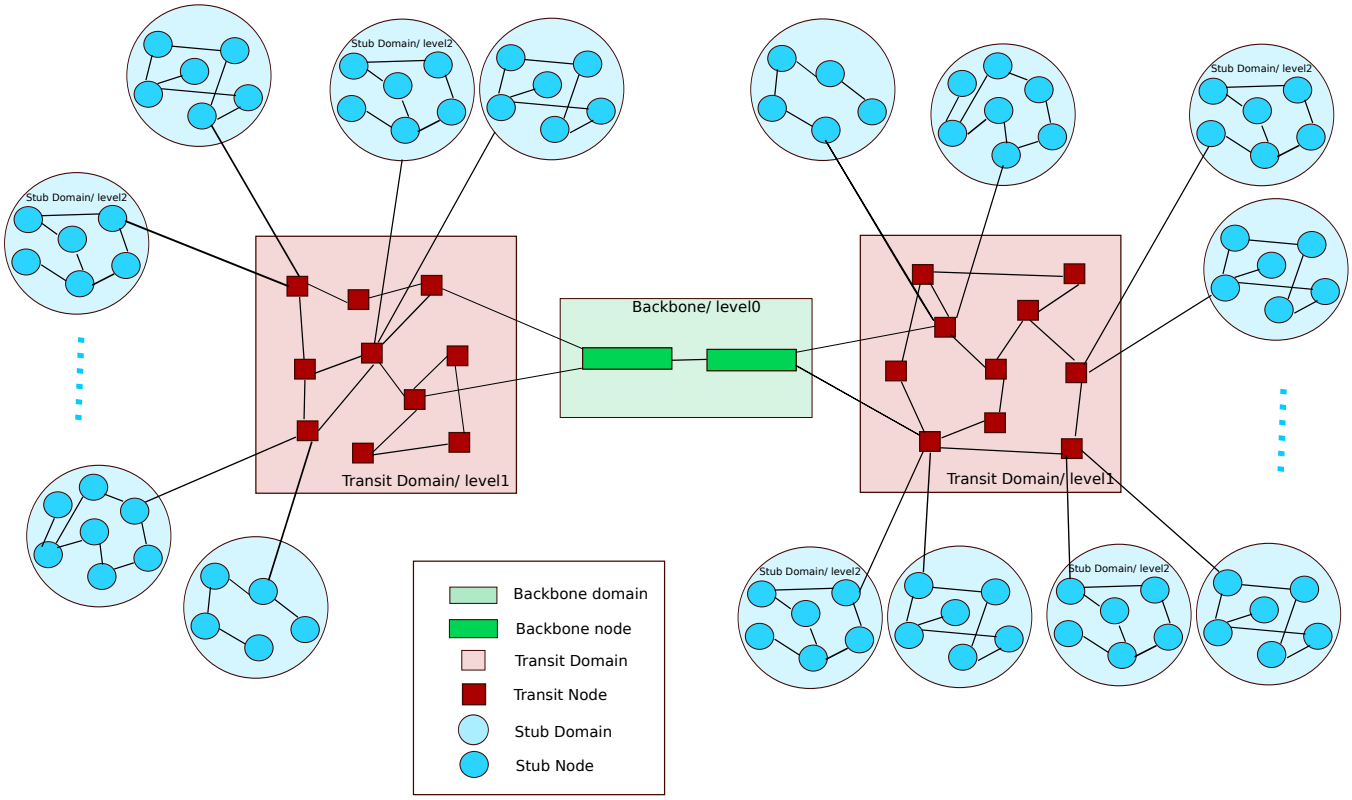


FIGURE 3. Transit-Stub topology

is supposed that the next event has not occurred yet ($T_{event} > Now$). The *Data* is then returned to the consumer.

4. PERFORMANCE EVALUATION

This section details the performance evaluation of our caching strategy and freshness mechanism for information-centric IoT networks. For this purpose, we use the ccnSim simulator [42]. It is a C++ framework under the OMNeT++ discrete-event simulator that implements the routine to simulate the NDN architecture. Every node implements the three system elements mentioned in the subsection 3.3, in form of layers. The first one, called "Core layer" is responsible for both the PIT management and the communication with the different layers. The second one, "the cache layer", represents the CS parts in the NDN structure. It acts according to a caching and replacement strategies. The third element is the strategy layer that takes the decision about *Interest* forwarding. CcnSim uses by default the shortest path forwarding strategy. In the remainder of this section, we describe the simulation scenario, the adopted metrics and the obtained results.

4.1. Simulation scenario

We summarize in table 1 the system parameters used in our simulations. Experiments were run with different caching strategies as LCE, LCD, ProbCache, Btw, edge-caching as well as with our caching strategy named consumer-cache. We use LRU (Least Recently Used) as a cache replacement policy.

Concerning the topology, authors in [10] affirm that there is not a single topology that can be used to evaluate ICN aspects and this choice depends on the focus of evaluation. In this paper, we present results with Transit-Stub (TS) topology whose properties imitate closely the IoT topology. The TS graph is a 3-level hierarchical topology presented in [43]. This model is composed of interconnected stub and transit domains and LANs connected to stub nodes. A stub domain carries only the traffic that originates or terminates in the domain. However, transit domains consider all transmissions and their role consist of efficiently interconnecting stub domains. The TS parameters are T, which is the total number of transit domains, S is the total number of stub domains per transit node. N_T and N_s are the average numbers of nodes per transit and stub domain respectively. L is the average number of LANs per stub node, and N_L is the average number of hosts per LAN. We set $T = 2$ with $N_T = 10$, $S = 2$ with $N_s = 6$, $L = 1$ and $N_L = 1$. The total number of nodes is $N = TN_T(1 + SN_S) = 260$ nodes and the total number of hosts $N_H = TN_TSN_SLN_L = 240$ hosts. Figure.3 presents our TS topology. Producers and consumers can be connected to 240 hosts. We choose to distribute them in such a way the producer

and its consumer do not belong to the same transit domain, however, a host can connect a consumer and a producer at the same time. We generate the topology with the GT-ITM tools [1] (Georgia Tech Internetwork Topology Models) using the parameters fixed above. GT-ITM is a complete set of tools for the conversion of network topologies that support NED language used in OMNET++.

Transmission delays are set by the GT-ITM. Values are within the range of $[2; 78]\,ms$. They are set so that transmissions in the third level are faster than the second level and the same with the second and the first level.

Analyzing the request popularity distribution in different geographical locations, S. K. Fayazbakhsh et al., [27] stated that the web distribution, used by a vast majority of previous work on ICN, behaves as a Zipfian distribution. Indeed, the majority of ICN studies use the Zipf distribution which stipulates that some popular contents have a high probability to be requested (e.g., new films, news, today's weather, etc.). However, under IoT, we do not refer to this distribution since it is rather devised for web-based contents and Internet applications. In IoT, contents have close request probabilities. As a result, we assume in this work that Interest packets are uniformly distributed as was done in other work such as [9, 8].

We note that each producer provides a single content as a single chunk. The number of producers is then equal to the number of contents $\mid F \mid$ in the network and the file size F is equal to 1 chunk. In [44], D. Rossi and G. Rossini showed that the ratio $\frac{C}{F|F|}$ of the cache size C over the catalogue size $F \mid F \mid$, is such that $\frac{C}{F|F|} \in \left[10^{-5}; 10^{-1}\right]$. In our simulation, we set $\frac{C}{F|F|} = 10^{-3}$.

Taking into consideration this ratio, we set the file number $\mid F \mid = 4000$ files and the cache size $C = 4$ chunks. The 4000 sensors are connected to 40 Gateways. We consider a variable number of consumers ranging from 20 to 30 and we suppose that all consumers are already connected at the beginning of the simulation. Clients ask for files following the arrival rate of the Poisson process with $\lambda = 1$. We use SPR (Shortest Path Routing) as a forwarding strategy.

Our simulations were carried out with a real IoT data extracted from the ADREAM [45] building in LAAS-CNRS laboratory which is a smart building. The 4000 sensors are scattered in the building. It is about periodic sensors (temperature, humidity and light) with different period T and OnOff sensors (presence, vibration and fall).

4.2. System performances

We start with evaluating different caching strategies and compare them to our consumer-cache strategy in term of system performances. We consider the *hop reduction ratio*, the *server hit reduction ratio* and the *response latency* metrics to evaluate a caching scheme.

The *hop reduction ratio* α measures the reduction of the number of hops traversed to satisfy a request compared to the number of hops required to retrieve the content from the server. α is analytically represented by Eq. 1

$$\alpha = 1 - \frac{\sum_{i=1}^{N} \frac{\sum_{r=1}^{R} \frac{h_{ir}}{H_{ir}}}{R}}{N} \tag{1}$$

Where N is the number of consumers, and R is the number of requests created per consumer. α represents the average over N consumers of averages over R requests per consumer of the hop reduction ratio of the request r sent by consumer i; $\frac{h_{ir}}{H_{ir}}$. The h_{ir} parameter is the number of hops from i to the cache that satisfies r, and H_{ir} is the number of hops from i to the producer.

β represents the *server hit reduction ratio*, the second metric that measures the reduction of the rate of access to the server. In other words, the alleviation rate of the server load. Eq. 2 calculates this metric, where $serverhit_i$ is the number of requests sent by i and satisfied by the server (producer) and $totalReq_i$ is the total number of requests, satisfied by both the server and the cache, sent by i.

$$\beta = 1 - \frac{\sum_{i=1}^{N} serverhit_i}{\sum_{i=1}^{N} totalReq_i} \tag{2}$$

The third metric is the *response latency*. It is the duration between the delivery of a request and its response. In Eq. 3, we calculate γ, the average of the response latency T_{ir} over N consumers and each one sends R requests.

$$\gamma = \frac{\sum_{i=1}^{N} \frac{\sum_{r=1}^{R} T_{ir}}{R}}{N} \quad (ms) \tag{3}$$

Figure. 4 portrays the system performances: the Server hit reduction ratio (Figure 4a), the number of evictions (Figure 4b) and the hop reduction ratio (Figure 4c), using 25 consumers. Figure.4d portrays the response latency as a function of the number of consumers (varying from 20 to 30). From the latter, we notice that the response latency gets better as the number of consumers increases. Indeed, increasing the number of consumers leads to an increase in the number of requests, which in turn augments the contents availability inside the topology.

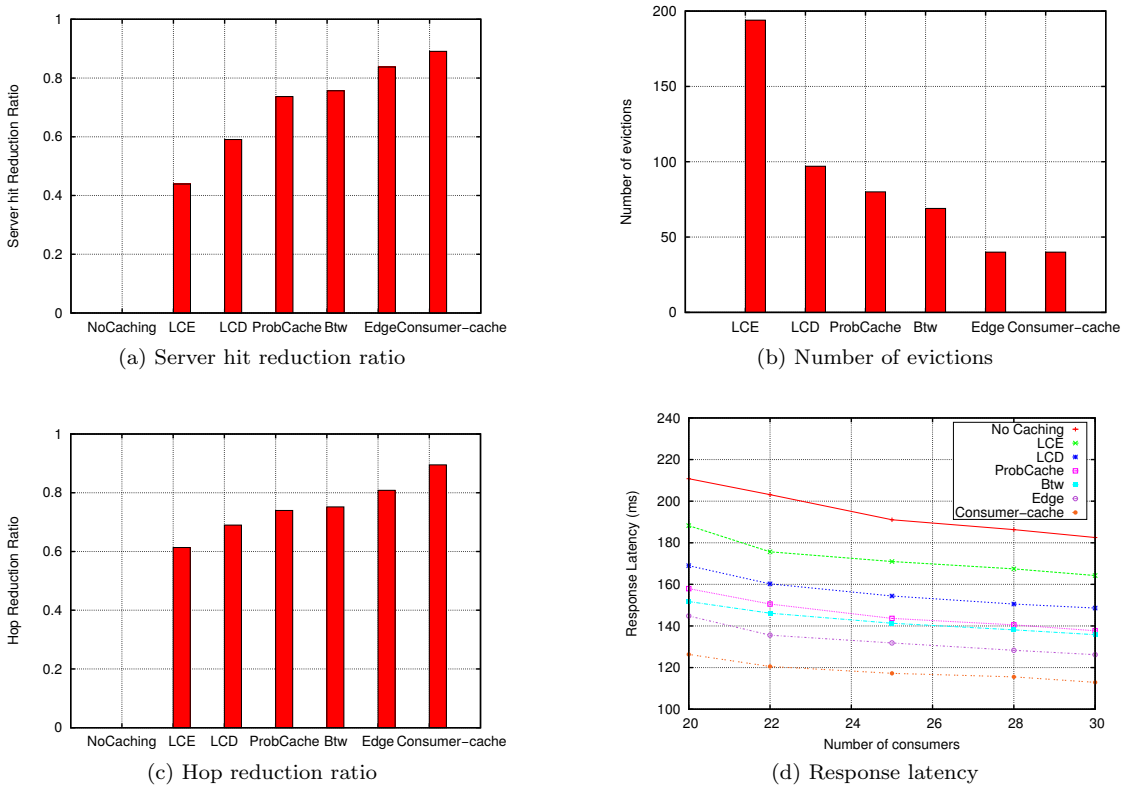
Without a caching strategy the Server hit Reduction Ratio and the Hop Reduction Ratio are equal to zero as all requests are satisfied by the producers.

The LCE strategy stores copies everywhere, which make content available at every node. However, caches fill up quickly and consequently, old contents at the bottom of the stack are rapidly evicted which increases the number of evictions and leads to cache misses. This explains the fact that this strategy performs the

[1] http://www.cc.gatech.edu/projects/gtitm/

TABLE 1. System parameters

Parameter	Meaning	Values
C	Cache size	4 chunks
$\mid F \mid$	Producers	4000 sensors
F	File size	1 chunk
$\frac{C}{F \mid F \mid}$	Cache size and Catalogue size ratio	10^{-3}
$Cons$	Consumers	$[20; 30]$ consumers
λ	Arrival rate	1
R	Replicas	1
RS	Replacement strategy	LRU
FS	Forwarding strategy	SPR
$transmission_delay$	Transmission Delay	$[2; 78]\, ms$
$simulation_time$	Simulation Time	$200s$
$expiration_time$	Expiration Time	$20s$
$lastEvents$	Time events	ADREAM DATA



(a) Server hit reduction ratio

(b) Number of evictions

(c) Hop reduction ratio

(d) Response latency

FIGURE 4. System performances

worst results in this scenario. To clarify the Server Hit Reduction Ratio results, we plot the average number of evictions for each caching strategy. Figure. 4b shows that LCE has the highest number of evictions as it was expected. Also with ProbCache, contents can probably be cached on more than one node within the request path which increases the number of evictions. However, with LCD and Btw there is one cache node in each request path. As a consequence, the number of evictions is lower compared to LCE and ProbCache. Finally, the edge-caching and consumer-cache have almost the same number of evictions. With LCE caching strategy, The Server Hit Reduction Ratio (Figure. 4a) is 0.43. This value means that only 43% of requests are satisfied from cache nodes. The Hop Reduction Ratio (Figure. 4c) for this strategy is about 0.61. Which means that paths are reduced by 61% in term of the number of hops. Concerning the third metric, the response latency (Figure. 4d) is about 182ms to 210ms.

The caching strategy LCD decides to cache contents at the node on one level down from the response source (Producer/cache node). After a certain number of

requests, it tends to LCE and all path nodes become caches. For this reason, LCD results, are not so good as LCE. Figure. 4c shows that LCD records 69% of hop reduction, and requests take about 164ms to 188ms of response latency (Figure. 4d). Its server hit reduction is about 0.59.

Concerning ProbCache and Btw, cache nodes are selected in the middle of the request path and probably more close to consumers, in the case of ProbCache. Simulation results of these two strategies are medium comparing to other caching strategies. The hop reduction ratio using ProbCache and Btw is respectively about 0.73 and 0.75. The same for response latency, ProbCache and Btw reports respectively about 137ms to 157ms and 135ms to 151ms. Figure. 4a shows 0.73 of the server hit reduction under ProbCache and 0.75 for Btw.

We remind that the third hypothesis on which we are based was that the edge nodes are the best placement for cache nodes. Our findings confirm the results presented in [27]. In fact, edge-caching reports good results. Under this strategy, we measured 0.83 in server hit reduction, 0.80 of hop reduction ratio, and 126ms to 144ms as response latency.

We detail now the results of our consumer-cache strategy. This latter stores copies in nodes attached to consumers which allow these consumers to easily reach requested contents. Consumer-cache has the best simulation results because requests are, in most cases, satisfied by the first hop node. We report for our strategy 0.89 of the server hit reduction. The hop reduction ratio is about 0.89, this implies that requests only cross 11% of hops on the path towards the producer. Finally, with our strategy, the response latency varies from 112ms to 126ms.

Since caching strategies can be quite costly in term of used resources, we calculate in the next section the cache cost of all caching strategies.

4.3. Cache cost

The *cache cost* is a compromise between data availability and caching overhead. It captures the trade-off between the average delay to receive a content and the corresponding caching cost[46]. The delay to obtain information is considered as a cost because, in the case of a cache miss, the time of waiting for the response by searching the caches is longer than the time of directly looking for the content from the remote server. As we have already mentioned in section 2.1, the decision whether to cache a content or not in each node depends on the adopted caching strategy. So, the caching cost is related to this decision; the more we have "yes" decisions, the higher the global caching cost is. Although caching improves content availability, it increases redundancy, and causes network overhead due to the existing stores in the cache nodes. In addition, some strategies, as LCE, come to fill up caches faster than other ones. In this case, we may remove still valid contents. As a consequence, the number of cache misses increases, and the cache is no longer effective. From this point of view, we consider the cache eviction as a cost; the more we evict contents, the higher the global caching cost is.

To calculate the cache cost, we were inspired by the cache cost function (Eq. 4) proposed in [46].

$$Totalcost = P_{hit} * C_{hit} + (1 - P_{hit}) * C_{miss} + N(S) * C_{cache} \quad (4)$$

Where P_{hit} is the probability to have a cache hit. It is calculated by the ratio of the number of requests satisfied by caches over the total number of requests. C_{hit} and C_{miss} are the average delays experienced by a consumer in order to receive its requested information from a cache and the producer, respectively. $N(S)$ is the number of caches in the topology. Finally, C_{cache} is the cost to cache items which is defined by Eq. 5.

$$C^j_{cache} = \frac{a}{1 - \rho^j_{util}} \quad (5)$$

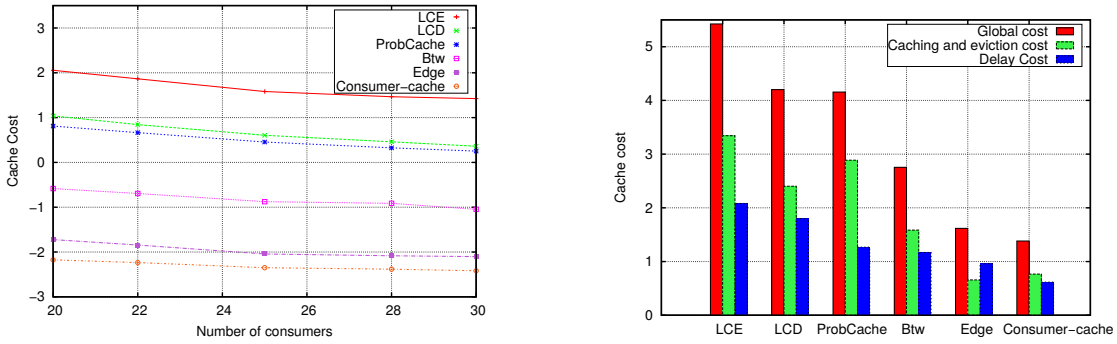
Where a is the cost coefficient that is set to 2[46], and ρ^j_{util} is the cache utilization. The cache utilization at a node j is the ratio of the number of times a content is cached in j ($Ncaching$) over the number of times a content passes through j ($Ndecisions$). C_{cache} calculates the average of C^j_{cache} over j cache nodes. The proposed cost function (Eq. 4) captures the trade-off between the average delays for a consumer to receive an information item, given by the first and the second terms of Eq. 4, and the corresponding caching cost given by the third term of Eq. 4.

To calculate the eviction cost, we should add, to Eq. 4, a fourth term which is similar to the third one. However, the calculation of ρ_{util} differs from that calculated in the third term. In fact, in the cache eviction function, the utilization is the ratio of the number of times an eviction is performed in node j ($Nevictions$) over the number of time a content is cached in j ($Ncaching$). This is because the eviction takes place when a new item will be cached and the cache is full.

To summarize, we define our cache cost metric in Eq. 6.

$$Totalcost = P_{hit} * C_{hit} + (1 - P_{hit}) * C_{miss} + N(S) * \frac{2}{1 - \frac{Ncaching}{Ndecisions}} + N(S) * \frac{2}{1 - \frac{Nevictions}{Ncaching}} \quad (6)$$

We evaluate the cache cost consumed by different caching strategies. Figure. 5 illustrates the cache cost function. Collected values show that the sum of the third and fourth parts in the cost function, which represent caching and eviction cost, is notably higher than the sum of the first and the second parts which illustrate the delay. For this reason, we use



(a) Cache cost

(b) Caching cost and delay cost

FIGURE 5. Global cache cost

normalization method to adjust the two values. The normalization method, named "reduced-centred", is calculated as shown in Eq. 7, where μ is the average of the distribution, and σ represents the standard deviation of the distribution.

$$Z = \frac{X - \mu}{\sigma} \tag{7}$$

Figure. 5a plots values of the reduced centered normal distribution Z for the calculated values of the cache cost function X. Negative values are the result of the normalization method. Simulation results are obviously positive.

We illustrate, in Figure. 5a, the cache cost with different caching strategies. We notice that the increasing of the number of consumers leads to a decrease in the cache cost. Caching strategies have different cache costs. In fact, we expect that LCE has the higher caching and eviction costs seen that it stores copies everywhere and cache nodes are more numerous. In addition, LCE has the worst system performances. From this view, LCE will have the worst global cache cost. In our previous results, LCD strategy doesn't perform good system performances. Furthermore, this strategy has many cache nodes because for each request the content is solved at one level down. As a consequence, LCD, in our scenario, is a costly strategy. Also with the ProbCache strategy, cache cost is assumed to be high. This is due to the fact that system performances are medium and content can probably be cached in different nodes for each request. Under Btw strategy, the caching and eviction costs are expected to be minimal, because it is always the same node having the highest betweenness centrality that keeps a copy of a content. On the other hand, system performances with Btw strategy are medium. This makes its cache cost medium. The edge-caching strategy had good system performance results as consumer-cache, in addition, the number of cache nodes in the topology are constant and not very high. Consequently, the cache cost under this strategy is not very high. Concerning, our caching strategy, consumer-cache, it performs the best system performances, and moreover, cache nodes are limited as with edge-caching, which makes it the least costly strategy.

As we have discussed, we report, in Figure. 5a, from 2 to 1.4 cache cost as the highest cost with LCE. The minimum cost is calculated with our consumer-cache strategy with -2.4 to -2.1. The second least costly strategy, edge-cache, reports -2.1 to -1.7. Then, Btw, has -1 to -0.5 of global cache cost. At the last, we found that ProbCache and LCD have similar results with 1 to 0.25.

For better understand of the trade-off between the caching cost and delay cost, we separately depict in Figure. 5b the delay cost and the caching and eviction cost with 25 consumers. The objective is to show which of the different components building up the Global Cost metric dominate.

Figure. 5b shows that the delay cost is less than the caching and eviction cost. Furthermore, we remark that with LCE, the cache cost is very important. In the other side, the edge-caching strategy has the least caching and eviction cost since it has the least number of cache nodes. Consumer-cache strategy has also a very low caching and eviction cost and its low delay cost makes it the least costly caching strategy. We can also remark that although ProbCache system performance results were better than LCD results, these two strategies have similar cache cost results. With Figure. 5b, we can understand that ProbCache has well a low delay cost, however its caching and eviction cost is higher than the LCD one which makes them equal in term of global cost.

In the following section, we evaluate our second contribution, which is the freshness mechanism.

4.4. IoT data freshness

To evaluate the freshness mechanism, we use the following metric presented in Eq. 8. *Validity* is the percentage of the valid contents received by N consumers against the total number of received content including valid and invalid ones. In Eq. 8, we respectively note $valid_i$ and $invalid_i$ as the number of valid and invalid content received by consumer i.

$$Validity(\%) = \frac{\sum_{i=1}^{N} valid_i * 100}{\sum_{i=1}^{N} valid_i + invalid_i} \quad (8)$$

With the aim of maximizing the content validity percentage to meet the IoT coherence requirement, we propose to integrate our freshness mechanism to several caching strategies. Event-based freshness mechanism tries to predict the exact times of updates in order to eliminate copies supposed to be invalid. Figure.6 depicts the percentage of fresh content with or without freshness mechanisms using different caching strategies.

Without freshness mechanism, cached copies are never checked before being sent. After a certain amount of time, all copies will be deprecated. As it is shown in Figure.6, the percentage of content validity for all caching strategies, without the use of freshness mechanism, do not exceed 20%. We report, 2% using the LCE, LCD, Btw and ProbCache strategies, 9% with edge-caching and 19% under consumer-cache. We notice that the consumer-cache strategy inherently maintains data validity. This result depends on the number of evictions (Figure.4b). In fact, strategies that have a high number of evictions may remove from caches still valid contents.

Expiration-based freshness mechanism comes to deal with this problem, but it is difficult to fix the right *expiration_time*. We propose to put the average of events period in order to cover the maximum of content updates. We choose $expiration_time = 20s$. Under Expiration-based freshness mechanism, the improvements are not very impressive. The graph portrays 9% using LCE, LCD and Btw, 5% with ProbCache, 34% under edge-caching and 40% using consumer-cache.

Our freshness mechanism has proven that it can significantly improve the content validity percentage. Figure.6 shows that this percentage can reach 98% with different strategies. We conclude, from this figure, that event prediction is a good solution to increase the content validity, especially with steady systems.

We infer that even if the expiration-based freshness mechanism combined with our consumer-cache strategy improved the validity percentage, the event-based freshness still performs better results. Consequently, we deduce that our proposal is a good solution for IoT data freshness.

5. DISCUSSION

As it was shown in the previous subsection, in-network caching efficiency strictly depends on the number of cache evictions. In fact, this parameter impacts the system performances since the cache replacement may cause cache misses. We remark that the closer the cache nodes are to the producer the higher is the number of evictions. This is because nodes close to the producers belongs to many request paths by against nodes close to consumers belongs only to request paths starting from this consumer. However, the impact of the number of evictions on the performance results is not proportional. For instance, edge-caching and consumer-cache have the same number of evictions but consumer-cache outperforms edge-caching. In fact, under edge-caching, *Interest* packets sent by consumers in the middle of paths will be satisfied by the producer since there is no cache node within the request path. In this case, we have a cache miss without any cache eviction. However, consumer-cache makes contents available to consumers, just in one hop.

Furthermore, we may assert that, in high traffic environment like IoT, we should minimize packets transmission within the first (backbone) and the second topology levels. Our proposed consumer-cache strategy, as well as the edge-cache strategy, significantly relieve these two levels as most of the requests are satisfied within the stub domain.

In an IoT environment, system performances, as well as cache cost, need to be closely considered. Our proposed consumer-cache performs the best trade-off between content availability and caching cost. As such, consumer-cache strategy stands out as a viable solution in a such IoT environment.

Finally, we showed that with an integrated event-based freshness mechanism, caching strategies perform much better and attain a high percentage of content validity up to 96% of fresh contents. Without any freshness mechanism, the number of evictions influences the data freshness due essentially to adopted replacement strategy which may delete still fresh contents.

6. CONCLUSION

In the quest to exhibit the potential of Information-Centric Networking (ICN) as a solution to adequately support Internet-of-Things (IoT) systems, we focused on Named Data Networking (NDN). IoT data are mostly transient and frequently updated by their producers which imposes stringent requirements in terms of cache replacement and information freshness. We investigated the most relevant in-network caching mechanisms and proposed a caching strategy called *consumer-cache* along with a freshness mechanism called *event-based freshness*. *consumer-cache* stores a copy of contents on on-path nodes directly connected consumers in a way to minimize the response latency and maximize the hop reduction ration.

We carried out extensive experiments using the ccnSim on a real IoT scenario. The obtained results indicated that our two proposals outperform other schemes and satisfy IoT systems requirements in terms of data freshness and cache cost, with a notable improvement in content validity and content availability.

As IoT scenarios and applications are very diverse in terms of nature and requirements, further experiments

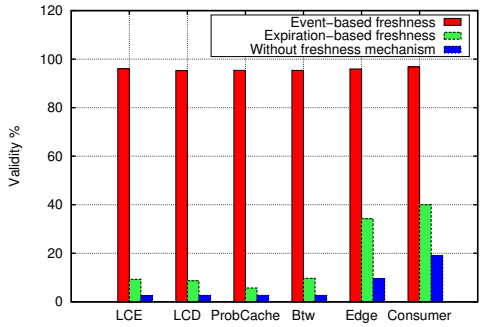


FIGURE 6. Validity %

are underway to properly ascertain the adequacy and relevance of our proposed strategy on various IoT scenarios. Furthermore, as IoT devices can be mobile, there is certainly a need to study the effect of both consumer and producer mobility on our proposed caching strategy.

7. ACKNOWLEDGEMENT

The authors extend their appreciations to the Deanship of Scientific Research at King Saud University for funding this work through Research Group No. RGP-1439-023.

REFERENCES

[1] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013) Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, **29**, 1645 – 1660.

[2] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L. (2009) Networking named content. *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies CoNEXT '09*, Rome, Italy, 1-4 December, pp. 1–12. ACM.

[3] Pentikousis, K., Ohlman, B., Corujo, D., Boggia, G., Tyson, G., Davies, E. B., Molinaro, A., and Eum, S. (2015). Information-Centric Networking: Baseline Scenarios. RFC 7476.

[4] Baccelli, E., Mehlis, C., Hahm, O., Schmidt, T. C., and Wählisch, M. (2014) Information centric networking in the iot: Experiments with ndn in the wild. *Proceedings of the 1st International Conference on Information-centric Networking ICN '14*, Paris, France, 25-26 September, pp. 77–86. ACM.

[5] Heidemann, J., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D., and Ganesan, D. (2001) Building efficient wireless sensor networks with low-level naming. *SIGOPS Oper. Syst. Rev.*, **35**, 146–159.

[6] Amadeo, M., Campolo, C., Iera, A., and Molinaro, A. (2015) Information centric networking in iot scenarios: The case of a smart home. *2015 IEEE International Conference on Communications (ICC'15)*, London, UK, 8-12 June, pp. 648–653. IEEE.

[7] Quevedo, J., Corujo, D., and Aguiar, R. (2014) A case for icn usage in iot environments. *IEEE Global Communications Conference (GLOBECOM'14)*, Austin, Texas, USA, 8-12 December, pp. 2770–2775. IEEE.

[8] Hail, M., Amadeo, M., Molinaro, A., and Fischer, S. (2015) Caching in named data networking for the wireless internet of things. *International Conference on Recent Advances in Internet of Things (RIoT'15)*, Singapore, 7-9 April, pp. 1–6. IEEE.

[9] Quevedo, J., Corujo, D., and Aguiar, R. (2014) Consumer driven information freshness approach for content centric networking. *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS'14)*, Toronto, Canada, 27 April- 2 May, pp. 482–487. IEEE.

[10] Pentikousis, K., Ohlman, B., Davies, E. B., Boggia, G., and Spirou, S. (2016). Information-Centric Networking: Evaluation and Security Considerations. RFC 7945.

[11] Adjih, C., Baccelli, E., and Fleury, E. (2015) Fit iot-lab: A large scale open experimental iot testbed. *IEEE 2nd World Forum on Internet of Things (WF-IoT'15)*, Milan, Italy, 14-16 December. IEEE.

[12] Zhang, G., Li, Y., and Lin, T. (2013) Caching in information centric networking: A survey. *Computer Networks*, **57**, 3128 – 3141.

[13] Dingle, A. and Pártl, T. (1996) Web cache coherence. *Comput. Netw. ISDN Syst.*, **28**, 907–920.

[14] Vural, S., Navaratnam, P., Wang, N., Wang, C., Dong, L., and Tafazolli, R. (2014) In-network caching of internet-of-things data. *IEEE International Conference on Communications (ICC'14),*, Sydney, Australia, 10-14 June, pp. 3185–3190. IEEE.

[15] Ahlgren, B., Dannewitz, C., Imbrenda, C., Kutscher, D., and Ohlman, B. (2012) A survey of information-centric networking. *IEEE Communications Magazine*, **50**, 26–36.

[16] Xylomenos, G., Ververidis, C., Siris, V., Fotiou, N., Tsilopoulos, C., Vasilakos, X., Katsaros, K., and Polyzos, G. (2014) A survey of information-centric networking research. *IEEE Communications Surveys and Tutorials*, **16**, 1024–1049.

[17] Amadeo, M., Campolo, C., Iera, A., and Molinaro, A. (2014) Named data networking for iot: An architectural perspective. *European Conference on Networks and Communications (EuCNC'14)*, Bologna, Italy, 23-26 June, pp. 1–5. IEEE.

[18] Amadeo, M., Briante, O., Campolo, C., Molinaro, A., and Ruggeri, G. (2016) Information-centric networking for {M2M} communications: Design and deployment. *Computer Communications*, **89-90**, 105 – 116.

[19] Amadeo, M., Campolo, C., Molinaro, A., and Ruggeri, G. (2014) Content-centric wireless networking: A survey. *Computer Networks*, **72**, 1 – 13.

[20] Franois, J., Cholez, T., and Engel, T. (2013) Ccn traffic optimization for iot. *Fourth International Conference on the Network of the Future (NOF'13)*, Pohang, South Korea, 23-25 October, pp. 1–5. IEEE.

[21] Meddeb, M., Dhraief, A., Belghith, A., Monteil, T., and Drira, K. (2015) Cache coherence in machine-to-machine information centric networks. *IEEE 40th Conference on Local Computer Networks (LCN'15)*, ClearWater Beach, Florida, USA, 26-29 October, pp. 430–433. IEEE.

[22] Amadeo, M., Campolo, C., and Molinaro, A. (2014) Internet of things via named data networking: The support of push traffic. *International Conference and Workshop on the Network of the Future (NOF'14)*, Paris, France, 3-5 December, pp. 1–5. IEEE.

[23] Xylomenos, G., Vasilakos, X., Tsilopoulos, C., Siris, V., and Polyzos, G. (2012) Caching and mobility support in a publish-subscribe internet architecture. *IEEE Communications Magazine*, **50**, 52–58.

[24] Saino, L., Psaras, I., and Pavlou, G. (2014) Icarus: a caching simulator for information centric networking (icn). *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques (SIMUTOOLS'14)*, Lisbon, Portugal, 17-19 March. ICST.

[25] Psaras, I., Chai, W. K., and Pavlou, G. (2012) Probabilistic in-network caching for information-centric networks. *Proceedings of the Second ICN Workshop on Information-centric Networking ICN('12)*, Helsinki, Finland, 17 August, pp. 55–60. ACM.

[26] Chai, W. K., He, D., Psaras, I., and Pavlou, G. (2012) Cache "less for more" in information-centric networks. *Proceedings of the 11th International IFIP TC 6 Conference on Networking - Volume Part I*, Prague, Czech Republic, 21-25 May, pp. 27–40. Springer-Verlag.

[27] Fayazbakhsh, S. e. a. (2013) Less pain, most of the gain: Incrementally deployable icn. *SIGCOMM Comput. Commun. Rev.*, **43**, 147–158.

[28] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B. (1985) Design and implementation of the sun network filesystem. *USENIX*.

[29] Kazar, M. L. (1988) Synchronization and caching issues in the andrew file system. *USENIX*, pp. 27–36.

[30] World wide web consortium, hypertext transfer protocol. http://www.w3.org/pub/WWW/Protocols/.

[31] World wide web consortium, w3c httpd. http://www.w3.org/pub/WWW/Daemon/.

[32] Berners-Lee, T. (2007). Analysis of caching and replication strategies for web applications.

[33] Vural, S., Wang, N., Navaratnam, P., and Tafazolli, R. (2016) Caching transient data in internet content routers. *IEEE/ACM Transactions on Networking*, **25**, 1048 – 1061.

[34] Wang, L., Wakikawa, R., Kuntz, R., Vuyyuru, R., and Zhang, L. (2012) Data naming in vehicle-to-vehicle communications. *In Proceedings of INFOCOM 2012 Workshop on Emerging Design Choices in Name-Oriented Networking*, Orlando, Florida, USA, 25-30 March. IEEE.

[35] Ngoc-Thanh, D. and Younghan, K. (2013) Potential of information-centric wireless sensor and actor networking. *International Conference on Computing, Management and Telecommunications (ComManTel'13)*, Ho Chi Minh City, Vietnam, 12-14 January, pp. 163–168. IEEE.

[36] Ren, Z., Hail, M., and Hellbruck, H. (2013) Ccn-wsn - a lightweight, flexible content-centric networking protocol for wireless sensor networks. *IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP'13)*, Melbourne, Australia, 2-5 April. IEEE.

[37] Teubler, T., Hail, M. A., and Hellbrck, H. (2013) Efficient data aggregation with ccnx in wireless sensor networks. *19th EUNICE Workshop on Advances in Communication Networking (EUNICE'13)*, hemnitz, Germany, 28-30 August. Springer.

[38] Amadeo, M., Campolo, C., Molinaro, A., and Mitton, N. (2013) Named data networking: A natural design for data collection in wireless sensor networks. *IFIP Wireless Days (WD'13)*, Valencia, Spain, 13-15 November. IEEE/IFIP.

[39] Katsaros, K., Chai, W., Wang, N., Pavlou, G., Bontius, H., and Paolone, M. (2014) Information-centric networking for machine-to-machine data delivery: a case study in smart grid applications. *IEEE Network*, **28**, 58–64.

[40] Liu, R., Wu, W., Zhu, H., and Yang, D. (2011) M2m-oriented qos categorization in cellular network. *7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM'11)*, Wuhan, China, 23-25 September. IEEE.

[41] Makridakis, S. and Hibon, M. (1997) Arma models and the boxjenkins methodology. *Journal of Forecasting*, **16**, 147–163.

[42] Chiocchetti, R., Rossi, D., and Rossini, G. (2013) ccn-sim: An highly scalable ccn simulator. *IEEE International Conference on Communications (ICC'13)*, Budapest, Hungry, 9-13 June. IEEE.

[43] Calvert, K. L., Doar, M. B., and Zegura, E. W. (1997) Modeling internet topology. *IEEE Communications Magazine*, **35**, 160–163.

[44] Rossi, D. and Rossini, G. (2011) Caching performance of content centric networks under multi-path routing. Technical report. Telecom ParisTech, Paris, France.

[45] LAAS-CNRS (2013). Adream. http://www.laas.fr/1-32329-Le-batiment-intelligent-Adream-instrumente-et-econome-en-energie.php.

[46] Vasilakos, X., Siris, V. A., Polyzos, G. C., and Pomonis, M. (2012) Proactive selective neighbor caching for enhancing mobility support in information-centric networks. *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking (ICN '12)*, Helsinki, Finland, 17 August, pp. 61–66. ACM.