

# A Keyword-based ICN-IoT Platform

Onur Ascigil\*, Sergi Reñé\*, George Xylomenos†, Ioannis Psaras\*, George Pavlou\*

\*Dept. of Electrical & Electronic Engineering, University College London

Email: {o.ascigil,s.rene,i.pсарas,g.pavlou}@ucl.ac.uk

†Dept. of Informatics, Athens University of Economics and Business, Greece

Email: xgeorge@aueb.gr

**Abstract**—Information-Centric Networking (ICN) has been proposed as a promising solution for the Internet of Things (IoT), due to its focus on naming data, rather than endpoints, which can greatly simplify applications. The hierarchical naming of the Named-Data Networking (NDN) architecture can be used to name groups of data values, for example, all temperature sensors in a building. However, the use of a single naming hierarchy for all kinds of different applications is inflexible. Moreover, IoT data are typically retrieved from multiple sources at the same time, allowing applications to aggregate similar information items, something not natively supported by NDN. To this end, in this paper we propose (a) locating IoT data using (unordered) keywords combined with NDN names and (b) processing multiple such items at the edge of the network with arbitrary functions. We describe and evaluate three different strategies for retrieving data and placing the calculations in the edge IoT network, thus combining connectivity, storage and computing.

**Index Terms**—IoT, ICN, edge computing, NDN

## I. INTRODUCTION

The Internet of Things (IoT) is becoming a reality in smart homes, smart buildings and smart cities [1], producing huge amounts of sensor readings that need to be processed, possibly in order to control actuators. IoT can be applied to multiple scenarios, such as Intelligent Transportation System (ITS), smart grids, smart homes, health care applications or Building Management Systems (BMS).

Multiple application-layer, cloud-based solutions have been proposed to process the huge amounts of data items produced by IoT devices, using IP to remotely manage IoT devices and pull data from them, with powerful cloud servers complementing the resource-constrained IoT edge (e.g., [2], [3]). However, cloud-based IP solutions are associated with long Round-Trip Times (RTTs) and are dependent on uncertain network connectivity.

In our view, localized processing at the edge of the network is the only way to offer adequate QoS for delay-sensitive applications, as well as to support applications where connectivity to the cloud is sporadic or impossible. Local processing decreases bandwidth consumption and provides better security and privacy, by avoiding the storage of sensitive data in the cloud (see Section V-E). Finally, sharing the cost of deploying IoT devices in the field across different applications is very hard when each device is only accessible to a remote cloud server for security reasons. Current cloud-based solutions could lead to tens of identical sensors being deployed side by side, each to serve a different application.

Information Centric Networking (ICN) has been proposed as a promising solution for IoT scenarios, as multiple ICN characteristics fit the IoT space. Naming data rather than endpoints greatly simplifies application design, also easing mobility as data names can be independent of location. Named data can be freely cached in the network, thus allowing resource-constrained IoT devices to be represented by a proxy gateway so as to reduce their communication needs. Different applications can share data based on their names, by using a security association between the application and the data, rather than with the endpoints producing such data. Moreover, named network functions [4] can extend ICN with data processing inside the network, enabling the aggregation of data depending on the data consumer’s request.

The most widely accepted ICN solution, Named-Data Networking (NDN) is based on hierarchical names, which easily lend themselves to naming aggregates of data. For example, in a BMS data names may be organized by location (e.g., /building/...), which simplifies applications needing to gather all data from a building. However, an application that wants to gather all data from temperature sensors, would prefer organizing names by sensor type (e.g., /temperature/...). Ideally, each application could use its own naming scheme, but that would require maintaining different routing state per application, which is currently infeasible [5].

Instead of defining multiple hierarchical names for the same data, we propose using a keyword-based naming system to define IoT data available within an IoT domain (e.g., a company, a building, a city). Data within that domain is defined by an (unordered) set of keywords such as #temperature, #RobertsBuilding, #Floor1, allowing each application to ask for a set of keywords that best describes the appropriate data. However, keywords cannot be aggregated as easily as hierarchical names, therefore we propose using names that consist of a hierarchical prefix, allowing NDN routing to be used to reach an IoT domain, and a set of keywords as a suffix, indicating specific data within that domain. To exploit available computing resources at the edge for data processing and aggregation, we propose adding a function name between these two elements to show how the gathered data should be processed, so as to return a single data item to the requester.

The contributions of this paper are as follows:

- We propose a flexible naming scheme combining NDN names as domain identifiers, keywords to indicate data sets and function names to process these data.

- We propose a hybrid NDN and keyword-based routing ICN solution for IoT, combining the scalability of hierarchical naming with the flexibility of keywords.
- We propose and evaluate three different strategies for locating data and performing computations within the edge IoT domain, offering some preliminary results.

The rest of the paper is organized as follows. In Section II we discuss the tag-based ICN routing scheme upon which our work is based, while in Section III we describe our proposed IoT platform. Preliminary performance results about three proposed function placement strategies are presented in Section IV. We consider various extensions for further research in Section V. We discuss related work in Section VI and conclude and discuss future work in Section VII.

## II. TAG-BASED ICN ROUTING

Our keyword-based platform has been inspired by TagNet, an ICN architecture using tags to name content items at the global level, albeit without function execution facilities [6]. In this section we present TagNet and explain how our platform diverges from it, due to its focus on IoT and edge computing.

In TagNet, data are identified by *content descriptors*, while nodes are identified by *host locators*. A content descriptor is a set of location-independent tags, chosen by the application to describe a content item. Each content descriptor is represented as a Bloom filter. In [7], Papalini et al. use 192-bit Bloom filters with seven hash functions to store content descriptors with no more than 15 tags in order to ensure that false positives (in lookups) are extremely rare.

The matching relation in TagNet is defined as a subset relation between sets of tags. A descriptor  $R$  in a content request matches a content item  $C$  if the request includes all tags (and maybe more) of the publication ( $R \supseteq C$ ). For example, a request with descriptor  $\{\text{temperature}, \text{Floor1}\}$  would match an item with descriptor  $\{\text{Floor1}\}$ . In our platform, we use the reverse relation, that is, matching implies that ( $C \supseteq R$ ), as we want a request for  $\{\text{temperature}\}$  to match a content item  $\{\text{temperature}, \text{Floor1}\}$ , so as to allow groups of content items to be expressed in a compact manner.

TagNet uses trees for routing, but since using a single tree for all routing can lead to stretched routes between some nodes, multiple trees are used, so packets need to include a tree identifier in their header. For each tree, a node maintains a list of the descriptors announced by all the nodes reachable through that neighbor, that is, a list of Bloom filters. When a router receives a packet with a request descriptor  $R$ , it forwards the packet towards each neighbor that has announced a matching descriptor, making sure that the total fanout of the request will not exceed a limiting factor  $k$ . The goal is to find a single item matching the request. In our platform, we expect to receive multiple content items with a single request for further processing, therefore matching uses the reverse rule than TagNet, which leads to more matches, and we do not limit the fanout - we can potentially reach the entire tree.

When a content request reaches a matching data item, the item needs to be returned to the requester. While in NDN requests leave breadcrumbs behind them, in TagNet

they are forwarded over the same tree used for the request. To achieve this, the nodes in each tree are labeled using the TZ algorithm [8], which allows any node to forward a packet using only the TZ-label of that node and the TZ-label of the destination. In TagNet then, each request carries the TZ-label of the requester, called the host locator of that node, to allow data items to be returned without maintaining per request state in the network. In our platform, we only use a single tree rooted at the border router between the IoT domain and the Internet, with requests traveling down the tree and responses traveling up the tree, leading to simplified routing and no need to carry TZ-labels.

## III. KEYWORD-BASED ICN-IoT

In this section we provide a description of our keyword-based ICN-IoT platform. We describe the naming scheme developed for our platform, the forwarding operation for requests and responses, three different strategies for data retrieval and processing, and provide a step-by-step example.

### A. Naming scheme

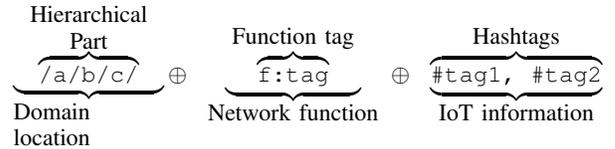


Fig. 1: Keyword-based Names

Our naming scheme needs to (a) locate an IoT domain across the Internet, (b) identify (individual or groups of) data values within that domain and (c) express processing requirements over these values. Taking these features into account, names are composed of three parts (see Figure 1):

- **Hierarchical Part.** It uniquely names the IoT domain that the information belongs to, using the hierarchical naming scheme of NDN; as NDN matches prefixes, it offers compatibility with NDN outside the IoT domain. Network administrators can define their own hierarchical name spaces, enabling communication between different campuses or networking domains *e.g.*, `/ucl/london/bloomsbury/`.
- **Function tag.** It consists of a single (mandatory) function tag, which describes the function that should process the set of IoT data described by the hashtags. If no processing is desired, an identity function tag is used.
- **Hashtags.** It comprises a set of hashtag-like keywords, which are used to describe the IoT data that need to be retrieved for processing by the function. The semantics of the hashtags depend on the data provided by the local sensors; we can even have tags describing processed data, rather than the raw data produced by the sensors.

In the example of the BMS application, the complete name included in a user request can have the fixed hierarchical part `/ucl/london/bloomsbury` naming the UCL London campus, the function name `f:average` and the hashtags `#RobertsBuilding, #Temperature`; these indicate that

we want to get the average of the temperature sensors in the Roberts Building. The corresponding data items are gathered and processed by the network, and the requester eventually receives the average value. A similar naming scheme with hierarchical and tag-based components has been used in the context of mobile computing for naming applications and their parameters [9].

### B. Request and content routing

The routing operation in our platform is a hybrid, based on a modified version of the tag-based routing of TagNet [7] for the local IoT domain and regular NDN forwarding [10] outside that domain. The modifications assume that the goal of the system is processing a set of IoT data in the local domain, before a result is sent to the Internet, using a flexible and scalable keyword-based naming scheme.

Using the hybrid naming scheme introduced above, outside an IoT domain, requests for content, called Interests in NDN, are routed using normal NDN forwarding, using the hierarchical prefix of the full name; the hierarchical part is advertised by the border router between the IoT domain and the Internet. Once an Interest reaches the border router, the hierarchical part is ignored and the Interest is routed based on the tags included in the name after the hierarchical prefix. If no function tag is included in the request, the border router could use a default function such as  $f: any$  (*i.e.*, return any data matching the requested tags). If no hashtags are included in the request, or there are no matching data in the IoT domain, the border router will send a negative acknowledgment towards the consumer.

Routing takes place over a single tree, with its root being the border router. If the physical topology is not a tree, a spanning tree algorithm is used to create a tree on top of the actual topology. Furthermore, we assume that all content requests originate from the root (border router) and, therefore, all final responses need to be returned to the root. This means that each node only needs to know which of its links lead to the root in order to return any results.

To locate content items, we reuse the TagNet scheme, that is, each content item is labeled with a Bloom filter that indicates the keywords it is associated with, and each node in the tree maintains a predicate for each of its downstream links, that is, a list of Bloom filters, indicating what content items are available via this link. Unlike TagNet, a request in our scheme matches a content item if its Bloom filter is a subset of the item's Bloom filter. This allows merging Bloom filters in the upstream direction (they become more permissive), at the cost of introducing false positives. In contrast, if Bloom filters were merged in TagNet, they could lead to false negatives (they would become more restrictive). Any content item matched, whether at its origin node or an intermediate cache, is returned towards the root, by following the links leading to the root at each node. Note that we use neither breadcrumbs nor TZ-labels for this. In contrast to [11] which proposes an elaborate scheme for gathering multiple responses to a single query, since we do not maintain per request state in the IoT domain, we can gather any number of results after a query; this is sensible in a system where sensors may fail or get

disconnected. By implication, functions need to be able to process an arbitrary number of arguments.

Note that we do not maintain the one Interest-one Data (*i.e.*, hop-by-hop flow balance) principle of NDN within the IoT domain, where only keyword-based routing is used. However, a single data packet is returned from the IoT domain for each arriving Interest, maintaining the one-in, one-out principle for the NDN-based portion of the network.

Data can be cached anywhere in the tree of the IoT domain, both in processed and raw data form, to satisfy further requests for the same information. However, for processed data, the matching should be exact and include *all* hashtags and the same function tag, since in this case we should send back a single, specific data item and not a set of matching data to be processed.

### C. Function placement and execution

In current cloud-based systems, all data required to execute a function like Average need to be individually fetched from IoT nodes to a cloud server, so that the function may process them. In our platform, the obvious place to execute a function would be the border router, since all data items converge there in our routing scheme; also, this allows returning a single piece of data over the NDN compatible part of the network, rather than a set of values. Therefore the first strategy called *Naive* is the most straightforward one: process everything in the root node of the IoT domain.

However, even in a simple IoT domain, like a university campus, we may have many levels in the routing tree, for example, building, floor, area, gateway, sensor: many simple sensors can connect to a gateway node (e.g., a Raspberry Pi), with the gateway connecting over short range radio (e.g., UWB or Bluetooth) to an area controller, and so on until the root. It therefore makes sense to push calculations closer to the source of the data, to avoid fetching large numbers of content items to the border router. Assuming that all nodes are capable of computing a function, the second strategy named *Minimum transfer* executes the calculation at the lowest level node where requests are branching in multiple directions. Starting from the root, if more than one downstream links match the request, then the calculation will be performed there; if only one link matches though, we repeat this procedure at the child node reachable via this link. This placement decision can be trivially made as the request propagates from the root towards the leaves. The node chosen for the calculation is indicated in the request and as data items are returned up the tree, when they reach that node they are stored temporarily. When the function executes, its results are pushed upstream to the root.

Although pushing calculations downstream reduces the bandwidth consumed by content item transmissions, it is reasonable to expect that nodes closer to the edge will have fewer resources available (e.g., Raspberry Pis at the lowest levels, with OpenWRT access points at higher levels). Such provisioning of hierarchically increasing resources has been proposed for edge-clouds to better cope with fluctuating demand [12]. We could thus consider how loaded a node is, by examining the expected completion time of the desired

function at that node. Therefore, we propose a third strategy named *Least congested* for function placement and execution, as a simple modification of the previous algorithm, that calculates this metric at each candidate node as we go down the tree, including in the request the currently least loaded node. This is done in order to not overload the root node under the *naive* strategy. As a result, the calculation will take place at the least loaded node between the root and the first branching node of the request, reducing completion time but increasing bandwidth overhead. Data and function results are returned upstream as in the previous algorithm.

Similar strategies that consider data transfer [13] and computational congestion [14] costs have been proposed in the context of request dispatching and service placement in edge-clouds.

#### D. An example

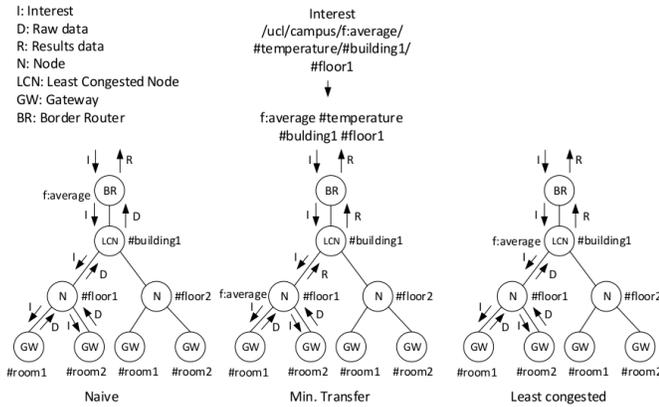


Fig. 2: Keyword-based multi-data retrieval and processing in an IoT domain

We can now provide a complete example of how our system would operate, referring to the simple topology shown in Fig. 2, depicting the three strategies.

- 1) An application issues an interest with the hierarchical prefix of the IoT domain plus a function tag and the data hashtags describing the IoT data, which indicates that it wants the average temperature at the UCL campus of the first floor of a certain building.
- 2) The NDN nodes (not shown) forward the interest towards the IoT domain following the regular NDN longest-prefix match rule towards the /ucl/campus IoT domain.
- 3) The interest reaches the border router of the IoT domain, which keeps the pending interest in its PIT and creates a request including the function tag (f:average) and the content tags (#temperature, #building1, #room1) into the IoT domain.
- 4) As multiple gateways have temperature sensors for the tags included, the request branches at the Floor 1 node. Depending on the evaluation strategy, either the identifier of the root node (Naive strategy), the first branching node (Minimum transfer strategy), or the node that would complete the calculation first between the root

and the first branching node (Least congested strategy) will be entered into each request.

- 5) Each node with data matching the keywords returns its latest temperature value and pushes the request towards the root. Nodes with no data send a negative acknowledgment to avoid using timers. The tags are used to match data in the reverse manner than in TagNet, that is, a less specific request matches a more specific entry (the request's Bloom filter is a subset of the routing entry's Bloom filter).
- 6) All data are gathered for processing at the node included in the request, where the Average function is executed and the result is returned up the tree to the Border router.
- 7) The border router uses its PIT to return the request to the originating node, encapsulating the result in an NDN data packet that includes all the parameters of the request in the hierarchical name.

#### IV. PRELIMINARY RESULTS

We now present a preliminary evaluation of the three mechanisms proposed in Section III-C for the retrieval and processing of IoT data. We evaluate these three strategies using a packet-level, discrete time event simulator based on Icarus [15], which is publicly available.<sup>1</sup>

We simulate the three strategies using a border router node attached to a regular tree topology with a height of three and a branching factor of ten (for example, one building with ten floors, ten areas per floor and ten devices per area). The resulting tree topology contains a total of 1111 nodes. The leaf nodes act as gateways that collect and store data. We assume that all nodes in the topology have computational power to execute functions. We use a betweenness centrality metric to distribute computational resources (for the execution of functions) in terms of number of cores to the internal nodes of the tree, assigning only a single core to the gateway nodes, which are assumed to be low-power devices. With this assignment, the internal nodes that are higher up in the hierarchy obtain larger numbers of cores. More specifically, we initially assign a single core to each node in the topology including the gateway nodes (*i.e.*, leaf nodes), and then distribute a total of 50 additional cores according to the betweenness centrality of the nodes. The root node obtains the maximum number of cores in the topology with a total of 16 cores, and the rest of the internal nodes share the remaining 34 cores. We assume that all cores in the nodes are identical and offer the same execution times for identical tasks.

We assume 100 applications sending requests, each requesting data from up to five randomly selected sets of gateways. We further assume that requests from the same application require the same execution time, and that function executions are not parallelizable; therefore, a single core handles each individual application request. We randomly generate the execution times of each of the 100 applications with a mean of 100 ms. In this simulation, we ignore the data sizes and only consider propagation delays; each link in the topology has a propagation delay of 3 ms. The nodes execute functions using

<sup>1</sup>[https://github.com/oascigil/icarus\\_iot](https://github.com/oascigil/icarus_iot)

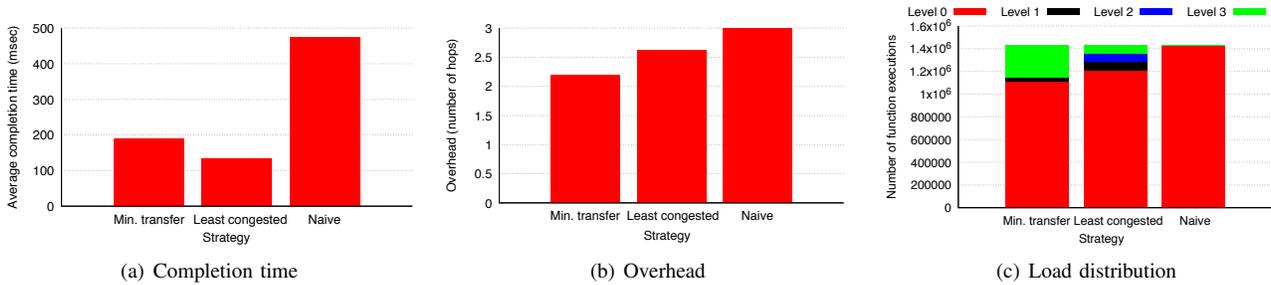


Fig. 3: Simulation topology results

a thread pool model with each core picking up a task from an incoming task queue as it becomes available. The main source of latency in our simulations is the waiting time of requests in the task queues of nodes.

We assume that each application sends requests to the gateway node at the same rate. When evaluating the individual strategies, we set the average aggregate request rate of all the applications to be approximately the same as the average service time of the root node (*i.e.*,  $\approx 160$  requests/sec). This is done in order to not overload the root node under the *naive strategy*, which can only schedule functions in that node. We compare the strategies based on two metrics, namely the *completion time* and the *overhead*. The completion time is the average time it takes for the requests to complete, *i.e.*, fetch data from gateways to one location, execute the function, and return results to the gateway node. The overhead metric is the average number of hops that data travel from gateways to the node executing the function.

In Figure 3, we demonstrate the performance of the three strategies. In Figure 3(a) we can observe the average completion time for each request in ms, in Figure 3(b) the overhead in number of hops, and in Figure 3(c) the load distribution in number of functions processed for each level of the tree. Note that there are 4 levels from the Border router (level 0) to the gateways (level 3). As expected, the completion time and overhead of the naive strategy is the worst, while the least congested strategy achieves slightly higher completion time under the same load. On the other hand, the minimum transfer strategy moves data through the least number of hops, causing the least overhead among the three.

In Figure 3(c) we see that the minimum transfer strategy moves some load from the root node to the gateways, with a very low load in intermediate nodes. This is due to the fact that it is very unlikely to retrieve data only from one subtree (especially at lower levels), when there is more than one randomly chosen gateway involved. The random choice process does not consider locality, therefore it can pick any one of the 1000 gateways. The execution takes place in the gateways only when a single gateway is involved in the request. On the other hand, the least congested node strategy puts more load in the root node than the minimum transfer strategy, since this node is more powerful than the rest and can allocate more processing power. As a result, even when it is possible to process the information closer to the source, the root node pulls most of the load. Depending on the objective

and the load, the network can choose among the minimum transfer and the least congested node strategy. For instance, for lighter loads, the minimum transfer strategy can achieve sufficiently good performance. We plan to extend this work in the future with an optimal strategy, which will jointly optimize the completion time and overhead.

## V. DISCUSSION

In the following, we include a discussion of some aspects that, although not specifically addressed in this paper, are relevant for our scheme, and could be the target of further research.

### A. IoT domain sizing

Although we have not made any specific assumptions on the size of the IoT domains so far, for security and administrative purposes it would not make sense for IoT domains to span the entire Internet. Normally, we would expect domains to belong to the same operator or owner, for example, a single organization, which would also ease the management of the keyword name space used, as well as simplify the mapping of domains to NDN names. The other limiting factor is the scalability of the routing system and, in particular, the information that needs to be maintained in order to perform matching based on keywords. Due to the tree topology used, nodes at higher levels would have to maintain more state. In [7], it is shown that the much more complex TagNet scheme with its multiple trees can scale across very large networks. In addition, while in TagNet each node must maintain all the individual content descriptors available over each outgoing link, as, if these descriptors were aggregated, it could lead to false negatives, in our scheme we can merge descriptors, at the cost of introducing some false positives. This allows our scheme to do limited merging of content descriptors (for example, as long as no more than a specific number of 1's is in each Bloom filter). Also, false positives do not make our scheme fail, but just perform suboptimally: we may send requests down redundant paths and branch at a higher level than needed, thus pushing calculations closer to the root.

### B. Routing scheme limitations

The tree-based routing scheme we have adopted, and in particular the use of a single tree, means that in a non-tree topology some routing paths may be stretched due to the

need to follow the tree. In practice however, IoT deployments do have the form of a tree, that is, there are no cross links between IoT gateways, therefore using a single tree would not affect performance. As we do not use the TZ-labels of TagNet, packets can only travel from or to the root. In the schemes we have devised for function execution placement, we always use an ancestor of the nodes providing the actual data, so returning data naturally converge there. A more advanced placement scheme could choose other nodes, both for complexity and transport efficiency reasons. For example, if most of the data points in a query come from a single building, it would make more sense to gather there data points from the other buildings and execute the function, rather than to an ancestor node, to save on transmission bandwidth. To achieve this, we could adopt the TZ-labeling scheme to allow data to be routed to arbitrary tree nodes, with minimal cost in state (just the TZ-label of the current node).

### C. Information time tags

How to deal with time when processing IoT information is a complex issue that, to the best of our knowledge, has not been finally settled in the ICN literature. Some solutions name processed data by using the timestamp of the oldest data value in the time interval [16]. In the IoT context and for our specific scheme, a reasonable means for the retrieval of processed information based on time would be to add more hashtags to the raw data items, reflecting the time of measurement. For instance, a raw data value could include multiple timing-related tags, such as #year, #month, #day, #hour, #minute and so on, including as many tags as required for a specific timing granularity. In the request, a consumer could include as many tags as required to process the required information (*e.g.*, by month, by day or by our) and the processed information would be cached including the timing tags included in the request. This solution limits the interval times of the processed information that can be requested, but can easily name and match timing information in both raw data and processed form. Note that it is still possible to issue multiple requests to retrieve data over multiple time intervals and combine them, as NDN would do to in order retrieve multiple IoT information items, even for a single interval.

### D. In-network data aggregation

Although our scheme is relatively simple to implement, we do not expect it to be used in very limited IoT devices. We instead expect designated network nodes (*i.e.*, gateways, access points, routers) close to the sensors and actuators within an IoT domain to act as data aggregation points. Our tag-based routing is not used beyond those data aggregation points; this is why we assumed that all nodes are capable of executing functions. Sensor data can either be pushed to or can be periodically pulled by the aggregation points, using protocols specifically designed for constrained devices. In particular, we envision special network functions to be used to fetch sensor data and make them available in advance (*e.g.*, for commonly requested information) or to activate actuators,

instead of processing/aggregating data on-demand. Such network functions can subscribe to the raw data (*e.g.*, required for an actuator decision) using a pub/sub model. For example, a network function could subscribe to any information items containing #BuildingName, #SmokeDetector, in order to automatically activate the fire extinguishers and trigger the fire alarm when necessary.

### E. Privacy and security considerations

We argue that local processing of data within an edge IoT domain can provide better privacy and security compared to cloud-based systems that perform both storage and processing in the cloud. To that end, we envision a proxy re-encryption (PRE) based access control scheme similar to the one proposed by Fotiou et al. [17], which would allow encryption of processed IoT data before sending it to untrusted parties for better privacy and security. Using the proxy re-encryption mechanism, a delegator (*e.g.*, local processing node) can enforce the data access control policies of the IoT domain at an untrusted delegatee (*e.g.*, remote storage services) when encrypted data are shared. More specifically, the delegator first encrypts the data using her public key and then provides both the ciphertext and a set of re-encryption keys to the delegatee. The delegatee uses the re-encryption keys to transform the ciphertext in a way that only authorised users can decrypt using their private keys. A downside of this scheme is the encryption cost placed at the edge, although not at end IoT devices. We leave a detailed investigation of an edge IoT domain access control scheme for future work.

## VI. RELATED WORK

Many cloud-based solutions for IoT and smart cities exist, for example, AWS-IoT [2], [18] is a solution based on Amazon Web Services. IBM has its own cloud-based solution for IoT using the IBM Watson IoT platform [3]. ICN has received more attention in recent years, as it offers interesting features for IoT solutions due to its in-network caching and computing, de-centralization or energy efficiency (*e.g.*, [19], [20], [21]).

In particular, NDN has been studied as a feasible solution to develop IoT solutions in [22], [23]. Implementing NDN for constrained devices can be a challenge, so multiple projects focused on implementing IoT solutions and prototypes based on NDN [24] for sensors and IoT devices, also providing libraries to easily deploy NDN solutions in wireless sensors [25], [26]. Existing work in NDN for IoT includes different IoT scenarios and experiments, such as Building Management Systems (BMS) and Smart Homes [16], [27], [28], [29], [30]. Some effort has been also focused on enabling multi-source data retrieval in NDN [11], an interesting feature for IoT communications, and also to enable push communications natively in NDN [31], [32], which are suitable for IoT scenarios based on producer-driven events.

Mobile Edge computing on the other hand, has been studied for IoT and smart cities [1] aiming at improved QoS, reduced latency and offloading traffic, pushing computing at the edge of the network. However, there is also some work on enabling

edge computing for ICN, via the Named Function Networking (NFN) scheme [4], with recent work focusing on using NFN over IoT scenarios [33].

Nevertheless, there has been no previous attempt to combine the benefits of edge computing and keyword-based ICN for IoT, using tag-based routing and function execution at the edge to deal locally with the huge amounts of IoT data, offering high flexibility and improving overall resource utilization and efficiency.

## VII. CONCLUSIONS AND FUTURE WORK

The ICN community has recently been exploring the feasibility of ICN solutions, especially those based on NDN, for IoT. Some of these works have focused on multi-data retrieval, breaking the one interest - one data rule, or deploying NDN in resource-constrained devices. However, there is no existing solution for the limitations placed by NDN hierarchical names to IoT applications, so as to allow more flexibility in terms of naming. In this paper we have presented an ICN solution based on keywords and network functions that targets IoT data retrieval and local processing. We presented a BMS use case and we evaluated three different strategies to fetch and process IoT data using our ICN-IoT platform. Our preliminary results show that a *least congested node* strategy is better in terms of latency, but a *minimum transfer* strategy creates less overhead.

This work is a first step in the investigation of keyword-based and edge computing ICN solutions for IoT. We have started the discussion on multiple areas upon which our future work will focus, such as timing issues for network functions, plus routing, scalability and security issues. We also plan to extend our work with further investigation on strategies to optimize both the completion time and overhead when processing data in the IoT domain.

## ACKNOWLEDGMENT

This work has been supported by the EC H2020 UMOBILE project (GA no. 645124), the EC H2020 ICN2020 project (GA no. 723014) and the EPSRC INSP fellowship (EP/M003787/1).

## REFERENCES

- [1] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 38–43, March 2017.
- [2] "AWS internet of things - securely connecting to the cloud." <https://aws.amazon.com/iot/>.
- [3] "ibm.com - harness the power of iot. - ibm watson IoT," <https://www.ibm.com/internet-of-things/>.
- [4] M. Sifalakis, B. Kohler, C. Scherb, and C. Tschudin, "An information centric network for computing the distribution of computations," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*, ser. ACM-ICN '14. New York, NY, USA: ACM, 2014, pp. 137–146. [Online]. Available: <http://doi.acm.org/10.1145/2660129.2660150>
- [5] S. S. Adhatarao, J. Chen, M. Arumaiturai, X. Fu, and K. K. Ramakrishnan, "Comparison of naming schema in ICN," in *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, June 2016, pp. 1–6.
- [6] M. Papalini, "Tagnet: A scalable tag-based information-centric network," Ph.D. dissertation, Faculty of Informatics of the Università della Svizzera Italiana, 2015.
- [7] M. Papalini, A. Carzaniga, K. Khazaeni, and A. L. Wolf, "Scalable routing for tag-based information-centric networking," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*, ser. ACM-ICN '14. New York, NY, USA: ACM, 2014, pp. 17–26. [Online]. Available: <http://doi.acm.org/10.1145/2660129.2660155>
- [8] M. Thorup and U. Zwick, "Compact routing schemes," in *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, ser. SPAA '01. New York, NY, USA: ACM, 2001, pp. 1–10. [Online]. Available: <http://doi.acm.org/10.1145/378580.378581>
- [9] I. Psaras, S. Reñé, K. Katsaro, V. Sourlas, G. Pavlou, N. Bezirgiannidis, S. Diamantopoulos, I. Komnios, and V. Tsaoussidis, "Keyword-based mobile application sharing," in *Proceedings of the Workshop on Mobility in the Evolving Internet Architecture*. ACM, 2016, pp. 1–6.
- [10] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '09. New York, NY, USA: ACM, 2009, pp. 1–12. [Online]. Available: <http://doi.acm.org/10.1145/1658939.1658941>
- [11] M. Amadeo, C. Campolo, and A. Molinaro, "Multi-source data retrieval in IoT via named data networking," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*, ser. ACM-ICN '14. New York, NY, USA: ACM, 2014, pp. 67–76. [Online]. Available: <http://doi.acm.org/10.1145/2660129.2660148>
- [12] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.
- [13] I.-H. Hou, T. Zhao, S. Wang, K. Chan *et al.*, "Asymptotically optimal algorithm for online reconfiguration of edge-clouds," in *MobiHoc*, 2016, pp. 291–300.
- [14] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2017, pp. 1–9.
- [15] L. Saino, I. Psaras, and G. Pavlou, "Icarus: a caching simulator for information centric networking (ICN)," in *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques (SIMUTOOLS '14)*. ICST, Brussels, Belgium, Belgium: ICST, 2014.
- [16] W. Shang, Q. Ding, A. Marianantoni, J. Burke, and L. Zhang, "Securing building management systems using named data networking," *IEEE Network*, vol. 28, no. 3, pp. 50–56, May 2014.
- [17] N. Fotiou and G. C. Polyzos, "Securing content sharing over ICN," in *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*. ACM, 2016, pp. 176–185.
- [18] W. Tärneberg, V. Chandrasekaran, and M. Humphrey, "Experiences creating a framework for smart traffic control using aws iot," in *Proceedings of the 9th International Conference on Utility and Cloud Computing*, ser. UCC '16. New York, NY, USA: ACM, 2016, pp. 63–69. [Online]. Available: <http://doi.acm.org/10.1145/2996890.2996911>
- [19] M. Amadeo, C. Campolo, J. Quevedo, D. Corujo, A. Molinaro, A. Iera, R. L. Aguiar, and A. V. Vasilakos, "Information-centric networking for the internet of things: challenges and opportunities," *IEEE Network*, vol. 30, no. 2, pp. 92–100, March 2016.
- [20] A. Lindgren, F. B. Abdesslem, B. Ahlgren, O. Schelen, and A. M. Malik, "Proposed Design Choices for IoT over Information Centric Networking," Internet Engineering Task Force, Internet-Draft draft-lindgren-icnrg-designchoices-00, Nov. 2015, work in Progress. [Online]. Available: <https://datatracker.ietf.org/doc/html/draft-lindgren-icnrg-designchoices-00>
- [21] Y. Zhang, D. Raychadhuri, L. A. Grieco, E. Baccelli, J. Burke, R. Ravindran, G. Wang, B. Ahlgren, and O. Schelen, "Requirements and Challenges for IoT over ICN," Internet Engineering Task Force, Internet-Draft draft-zhang-icnrg-icniot-requirements-01, Apr. 2016, work in Progress. [Online]. Available: <https://datatracker.ietf.org/doc/html/draft-zhang-icnrg-icniot-requirements-01>
- [22] W. Shang, A. Bannis, T. Liang, Z. Wang, Y. Yu, A. Afanasyev, J. Thompson, J. Burke, B. Zhang, and L. Zhang, "Named data networking of things (invited paper)," in *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, April 2016, pp. 117–128.
- [23] M. Amadeo, C. Campolo, A. Iera, and A. Molinaro, "Named data networking for IoT: An architectural perspective," in *2014 European Conference on Networks and Communications (EuCNC)*, June 2014, pp. 1–5.
- [24] W. Shang, Y. Yu, T. Liang, B. Zhang, and L. Zhang, "Ndn-ace: Access control for constrained environments over named data networking," NDN Project, Tech. Rep. NDN-0036, Revision 1, dec 2015.

- [25] A. Bannis, “Named data network internet of things toolkit (NDN-IoTT),” Available at <https://github.com/remap/ndn-pi>.
- [26] N. P. Team, “Ndncomm hackathon: Demonstrate ndn over bluetooth le on the arduino,” Available at <https://github.com/ndncomm/ndn-btle/tree/arduino>.
- [27] Z. Wang and J. Meng, “Ndn ebams node running in mini-ndn,” Available at <https://github.com/zhehaowang/bms-node>.
- [28] J. Burke, P. Gasti, N. Nathan, and G. Tsudik, “Securing instrumented environments over content-centric networking: the case of lighting control and ndn,” in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPs)*, April 2013, pp. 394–398.
- [29] —, “Secure sensing over named data networking,” in *2014 IEEE 13th International Symposium on Network Computing and Applications*, Aug 2014, pp. 175–180.
- [30] M. Amadeo, C. Campolo, A. Iera, and A. Molinaro, “Information centric networking in IoT scenarios: The case of a smart home,” in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 648–653.
- [31] M. Amadeo, C. Campolo, and A. Molinaro, “Internet of things via named data networking: The support of push traffic,” in *2014 International Conference and Workshop on the Network of the Future (NOF)*, Dec 2014, pp. 1–5.
- [32] J. Chen, M. Arumaiturai, L. Jiao, X. Fu, and K. K. Ramakrishnan, “Copss: An efficient content oriented publish/subscribe system,” in *2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*, Oct 2011, pp. 99–110.
- [33] Y. Ye, Y. Qiao, B. Lee, and N. Murray, “PIoT: Programmable IoT using information centric networking,” in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, April 2016, pp. 825–829.