Named data aggregation in wireless sensor networks

Younes Abid, Bilel Saadallah, Abdelkader Lahmadi, Olivier Festor

Named Data Aggregation in Wireless Sensor Networks

Younes Abid†, Bilel Saadallah†, Abdelkader Lahmadi* and Olivier Festor†
*Université de Lorraine, Loria, †Inria
Vandoeuvre-lès-Nancy, F-54506, France
Email: {abdelkader.lahmadi, olivier.festor}@loria.fr

***Abstract*—In this paper, we present a novel named data aggregation method dedicated to wireless sensor networks. The method relies on an adaptation of the CCNx protocol implementation that we have extended with in-network processing functions to aggregate named data efficiently. We have implemented and tested our solution with the Contiki operating system which is an operating system for resources-constrained embedded systems and wireless sensor networks. Our simulation and measurement results using the Cooja simulator and physical nodes show that our solution has a small overhead in terms of exchanged messages and provides acceptable data retrieval delays.**

I. Introduction

The Internet of Things (IoT) [1], a part of the Future Internet, conceives the world as a global network of interconnected objects where they are seen as proactive participants in information, business and social processes. These objects have the ability to generate and exchange data among themselves and to interact with their surrounding environment. Wireless sensors networks (WSN) are considered as a promising element of the IoT. Their capacities of monitoring the environments where they are deployed and collecting information is completely compliant with the IoT expectations. WSNs have been used in a variety of applications from different domains such as healthcare, environmental monitoring, production control and smart building monitoring. Their capacities to efficiently cover different environments and provide monitoring information make them an essential element in the IoT vision of a global network of interconnected objects. By interconnecting wide-range devices, the generation and exchange of information will become easier, rising in consequence the data availability and control for users and applications regardless of their location. In a such data-driven network, it may seem spontaneous to turn to content centric networking communication architecture for WSNs.

Content Centric or Named Data Networking [2] is an emerging communication architecture that is based on named data. Information is exchanged only in response to a request specifying the name of the data to retrieve. CCN forms a data-polling communication model which seems adapted for the functioning of a WSN considering the fact that its main use is to gather data. Using a data-driven network, a query instructs each node to sense its environment at a certain rate, for a period of time, and deliver matching data back to the sink.

The potentialities and the benefit of using Named Data Networking (NDN) for data collection in wireless sensor networks have been explored recently in [3]. They confirmed through simulation that a NDN approach fits well with WSN and it is able to achieve good performance in terms of overhead and data retrieval delays for periodical monitoring applications. Our work is close to theirs where we also believe that content centric and named data networking is a promising paradigm for WSN. In an early stage work [4], we have also explored this direction, where we showed the potentiality of using the CCNx protocol [5] for wireless sensor networks.

In this work, we present a novel aggregation service using a content centric communication layer adapted to wireless sensor networks where their nodes are usually with limited resources and their primary task is to collect data. We take the initiative to apply the Content-Centric Networking architecture in embedded systems, and sensor networks particularly since the CCNx protocol [5] was designed to be deployed in different environments, mainly where providing data content is of primary focus. In order to achieve this, we adapted and integrated the CCNx protocol into Contiki, an operating system for memory-constrained embedded systems and wireless sensor networks [4]. We have evaluated the communication layer using simulation and TelosB physical nodes. On top of this novel named networking layer dedicated to WSNs, we designed an aggregation service where data is queried and processed into the network using an extended naming schema for the available interest and content messages of the CCNx protocol.

The paper is organized as follows. In Section II we present background details about the CCN approach and the CCNx protocol operations and messages. In Section III, we detail the design and the implementation of the CCN communication layer and its associated aggregation service for WSNs. Performance evaluation results are presented in Section IV. Finally, in Section V we provide the conclusion and future work.

II. CCN approach

Content Centric Networking (CCN) is a communication architecture built on named data[2] where the identification and the transport of contents rely on their names and not on their location. The CCN approach addresses the issues limiting the current use of networks by increasing the availability of data. It provides caching to reduce congestion and improve delivery speed. In term of security, CCN suggests that trust in content is easily misplaced when relying on data locations. Instead, it builds security into the network at the level of data. In addition, because the communication relies only on data names, no mapping between contents and locations is done. Thus, the configuration of network devices is much simpler.

A. The CCNx protocol

The CCNx protocol [5] is a transport protocol for the Content-Centric Networking communication architecture. According to the CCN specifications, it is built on named data where the content name replaces the location address. The CCNx protocol provides location-independent delivery services for named data packets. The services include multi-hop forwarding for end-to-end delivery, flow control, transparent and automatic multicast delivery using buffer storage available in the network, loop-free multi-path forwarding, verification of content's integrity regardless of delivery path, and carriage of arbitrary application data[6].

Applications use the CCNx protocol on top of a lower-layer communication service that can handle packet transmitting. No restrictions are imposed on the nature of the lower-layer service. It may be a physical transport, another network or a traditional transport protocol. Although the CCNx protocol is designed to deliver contents based on their names, applications can run it on top of UDP or TCP to take advantage of existing IP connectivity. Since content is named independently of location in the CCNx protocol, it may also be preserved indefinitely in the network. Every packet of data may be cached at any CCNx router. Providing support for multicast or broadcast delivery, the network's use is more efficient when many people are interested in the same content.

The CCNx protocol supports a wide range of network applications by leaving the choice of naming conventions to the application. It may be natural to think of stored content applications such as distribution of video or document files, but the CCNx model also supports real-time communication and discovery protocols and it is general enough to carry conversations between hosts such as TCP connections.

In the CCNx specification, each node requires the following data structures to provide buffering/caching of data, manage content requests and forward messages to other nodes in the network.

- **Face**. It is a generalization of the concept of interface. In the CCN specifications, a face may be a connection to a network or directly to an application party. It may be configured to send and receive broadcast or multicast packets on a particular network interface, or to send and receive packets using point-to-point addressing in the underlying transport, or using a tunnel.
- **Content Store (CS)**. It is a cache where data is stored. It holds Content Objects created locally using the data collected by sensor devices and Content Objects received from other nodes. Contents are indexed to facilitate their retrieval and suppression.
- **Forwarding Information Base (FIB)**. It is a table of destinations for Interests, organized for retrieval by longest prefix match lookup on names. An entry in FIB can be a prefix that points to a set of destinations rather than a specific one.
- **Pending Interest Table (PIT)**. It is a table of sources for unsatisfied Interests. It is organized for retrieval by complete prefix match lookup on names (a match occurs when the interest prefix to compare and the prefix in the PIT entry match completely). Each entry in the PIT may point to a list of identifiers of the faces which are sources of the unsatisfied Interest.

B. CCNx operations and messages

CCN communication is driven by the consumers of data. There are two CCN message types: Interest message and Content Object message. The Interest message is a request of named data. It contain the full name that identifies a piece of content. This content will be specifically retrieved if its available in a node of the network. It can also contain simply a prefix of the content name. Then any content whose name matches the Interest name prefix can be a potential response to this Interest. The Content Object message is used to supply data. A Content Object message contains a data payload preceded by the identifying name. An Interest message contains only the prefix of the content to be retrieved, while a Content Object message is composed of two elements. As depicted in Figure 1 The first element contains the content name and the second element is reserved for data .

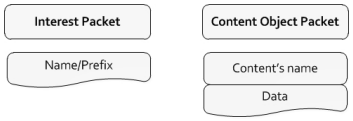

Fig. 1. CCNx messages format.

When an interest message is received by a node, it is processed according to the steps depicted in Figure 2. An ordered prefix match lookup is done on names, where a Content Store match will be preferred over a PIT match which will be preferred over a FIB match.

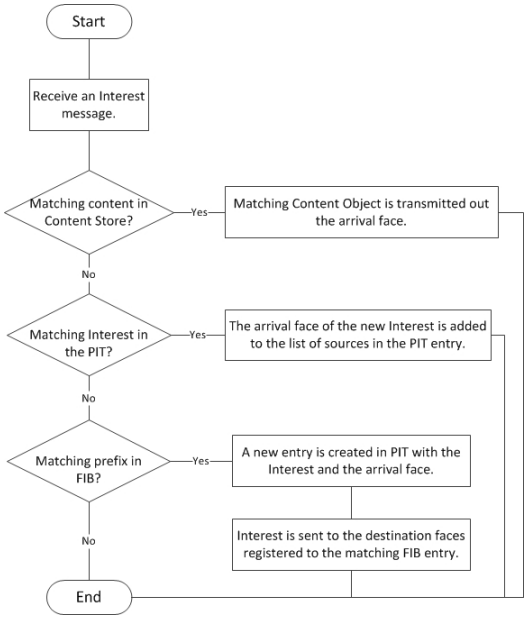

Fig. 2. Incoming Interest processing.

A received Content Object is processed according to the steps depicted in Figure 3. Lookups are performed on CS and PIT. If a matching Content Object is found in the CS, it means

that the newly arrived Content Object is a duplicate which can safely be discarded, because all previous Interest messages have already been satisfied and new ones will be satisfied out of the CS. A PIT match means the data was solicited by one ore more Interests sent by this node. If there is a match in the PIT, the Content Object is transmitted on all of the source faces for the Interests represented in the PIT, except the face with the identifier 0. If a matching PIT entry has the face identifier 0 among its list of source faces, it means that this node is the original requester of the received Content Object. In this case, it delete the reference of the face with the identifier 0 from the list of faces in the matching PIT entry before sending the Content Object on the rest of source faces (if existing). Next, the matching PIT entry is deleted from the table. If no match is found in the previous steps, then the content is unsolicited. A node must not forward unsolicited data and may discard unsolicited data. Here we store unsolicited data in the Content Store in case it is subsequently requested.

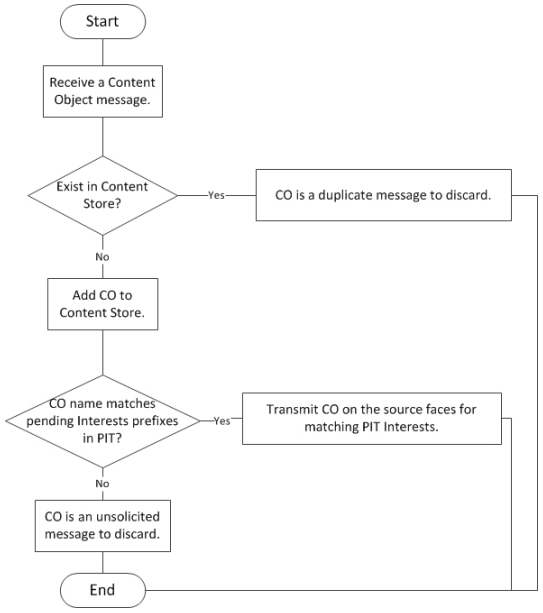

Fig. 3. Incoming Content Object processing.

III. Approach Design and Implementation

In this section, we provide the details of our named data aggregation communication layer where we adapted and extended the CCNx protocol. We adapted an existing CCNx implementation (version 0.3.0) to provide a communication layer for WSNs based entirely on named data . Some parts of CCNx source code are reused, others are simplified or adjusted to meet the constraints of sensor networks. Secondly, we present the design and the implementation of the collection and aggregation service using this communication layer by extending the naming schema of Interest and Content messages.

A. CCNx adaptation and extension

We integrated the adapted CCNx communication layer in Contiki operating system [7] as depicted in Figure 4. To meet the Contiki's implementation style, the CCN communication layer is composed of a Stack and a Driver. The Stack implements the CCN processing, forwarding and caching functions and manage event posting to processes. The Driver handles messages exchange with the lower layer.

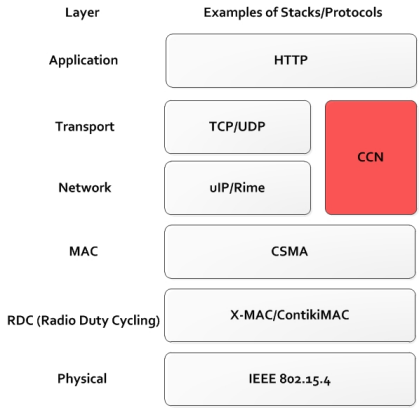

Fig. 4. The integration of the CCN layer into Contiki system architecture.

The CCN communication layer handles packets transmission and does not rely on other transport protocols to deliver messages. It uses directly the MAC layer implementations available in Contiki to transmit its messages. It is built according to a full CCN-compliant communication model that relies entirely and uniquely on named data. Each sensor node is connected to the network through its faces. We designed two communication faces. At first, we made the choice to create at each node a single face reserved for broadcast to communicate with other nodes. This would be convenient when a consumer, which wants to retrieve a piece of content, asks for it by broadcasting its Interest message to all nodes within its communication range. Then, we added a second face to which we attributed permanently the identifier 0 in order to differentiate the Interest messages initiated by a node (the node is the first source of the request) from the other Interests messages relayed by this node.

The adapted CCNx protocol relies on the IEEE 802.15.4 standard specifications to implement the physical and MAC layers. The size of a 802.15.4 frame is limited to 127 bytes, with 72-116 bytes of payload available after link-layer framing, depending on a number of addressing and security options. In this work, we fix the maximal size of a CCN packet or message to 102 bytes (127 bytes - 25 bytes of a MAC header).

In order to provide a simple CCN communication model without a fragmentation mechanism, the message formats are simplified to leave more payload bytes for names and data carrying. Based on the messages scheme defined by the CCNx protocol [5], all the optional fields are discarding from the CCN messages provided by our implementation. In the Interest message, we only kept the prefix of the content to be retrieved, while in a Content Object message, we only kept, the content name and retrieved data.

In the Content Centric Networking approach, the *Name* element plays a pivotal role in the communication. CCN relies on the use of names to deliver messages and exchange contents between the network nodes. A CCN name can identify a specific chunk of data. It can also identifies a collection of data in the case where this name is a prefix of the name of every piece of content in the collection. This would explain why a CCN name may be refereed to as a name prefix or simply a

prefix. To represent names, CCN refers to URI scheme. In the example given in 5, the prefix */Temperature/Inria* identifies the set of contents indicating the temperatures in the offices 132, 214 and 125 in the Inria building.

Prefix :
/Temperature/Inria

Names of Contents :
/Temperature/Inria/Floor1/Office132 : 21° C
/Temperature/Inria/Floor2/Office214 : 19° C
/Temperature/Inria/Floor1/Office125 : 18° C

Fig. 5. An example of CCN naming.

In order to minimize the cost of lookup operations when processing and forwarding CCN messages, FIB and PIT tables are coupled so that PIT entries will be attached to the FIB entry whose prefix offers the longest match. This would result in some modifications to the processing steps presented in Figure 2, as any lookup performed on PIT will result in a lookup on FIB prefixes. So to optimize the lookup operations, we reversed step two and three as shown in Figure 6. Step one remains unchangeable so the first lookup will be performed on CS. If no matching Content Object is found, a lookup is performed on FIB. If there is no matching entry, we can deduce automatically with the coupling of the two tables that there are no matching PIT entries. The Interest message will be then discarded. If a FIB match is found, we check the PIT entries attached to the matching FIB entry. If a PIT entry match is found, we simply add the arrival face to the list of face identifiers of the matching pending Interest in the PIT and processing is over. Otherwise, if no PIT entry matches the received Interest message, a new entry is added to PIT and attached to the matching FIB entry. The Interest will be then forwarded out the faces registered to the FIB matching entry.

More details about our implementation of the CCNx protocol for Contiki is available in [4].

B. Named Data aggregation

Our adaptation of the CCNx implementation is only able to collect data from available nodes through the sending of the interest message by a sink and the reception of data from one or several nodes according to the requested naming prefix. We have extended this implementation with an aggregation service where the requested data and the processing functions to be applied are including in the interest naming schema.

1) Tree building: To collect and aggregate data from available nodes in the network, we need firstly to build a tree where each node has a short path to the sink node. Each node will select a single parent node to the sink. The tree is built using a mechanism similar to the hop distance metric proposed in the RPL protocol [8]. However, we are only using a single message which is a specific interest message containing the current rank of the node and the MAC address of the selected parent node to the sink node. The rank information denotes the number of hops between the sink and a node in the network. The sink node initiates the process by broadcasting an interest message containing only its own rank value. Each node in its communication range will receive this message and computes its rank according to the algorithm 1. When broadcasting its interest, each node includes in the interest, the MAC address of its node parent which will be used later in the data collection and aggregation phase to avoid duplicated messages or a wrong computation result of an aggregation function.

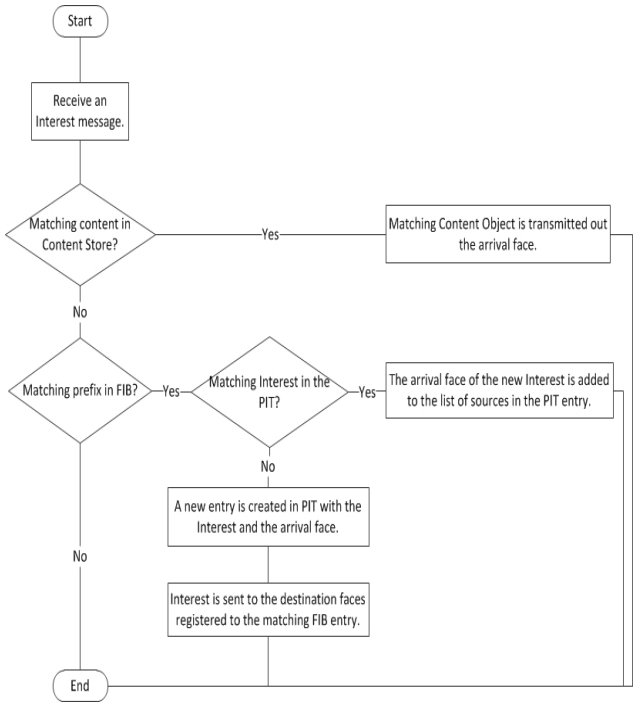

Fig. 6. Modified algorithm for incoming Interest processing.

Algorithm 1 Collection Tree Building

Function CTB ()
1: **if** Interest message is received **then**
2: Extract $received\_rank$ from Interest message
3: Extract $parent\_mac$ from the interest message
4: **if** (my_rank = -1) or ($received\_rank < my\_rank$) **then**
5: /* This is a message from a parent node */
6: $my\_rank = received\_rank + 1$
7: Broadcast Interest (my_rank, $parent\_mac$)
8: **else if** $received\_rank > my\_rank$ **then**
9: /* this is a message from a child node */
10: **if** $parent\_mac$ = my MAC address **then**
11: Add source node of the interest as a child
12: **end if**
13: **end if**
14: **end if**

2) Data collection: After building the collection tree, the sink node broadcasts a collect interest using the prefix "/COLLECT" to the set of its neighbours. After receiving the message, each node compares its current rank with the rank in the naming prefix of the interest message. If the message comes from a low ranked node, then it checks the FIB table to broadcast it. If the interest message requests only sensed data without processing, then the nodes checks the content store (CS) for matching content to be sent using a content object message. However, if the interest includes also an aggregation operator, then the node calculates the local

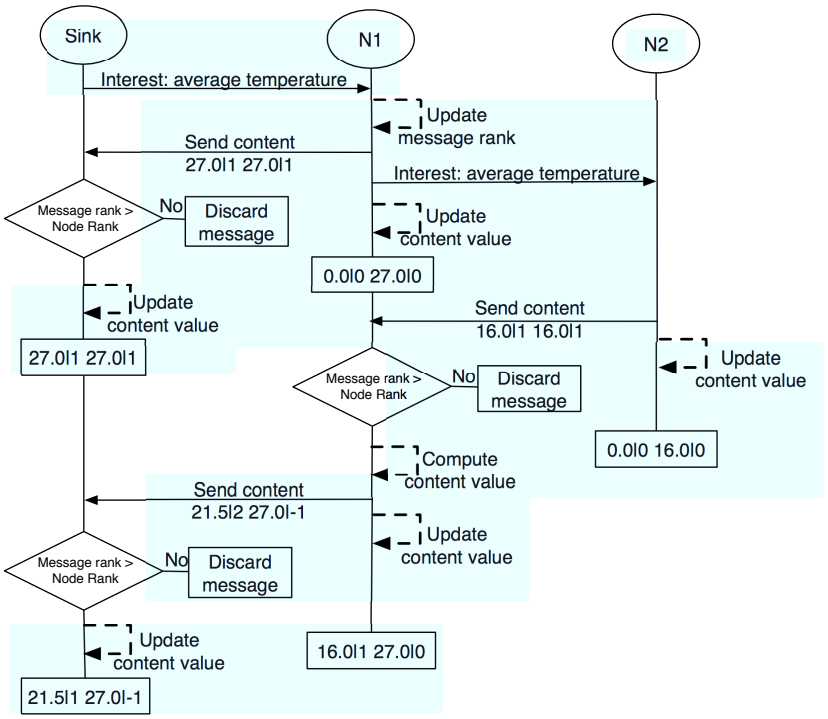

Fig. 7. Sequence of actions and exchanged messages during the aggregation of a weighted average temperature between the sink and two nodes.

result using available data in CS and sends it using a content message.

When a node receives a content message, it checks the included rank and the MAC address values. If the received rank value is lower than its own rank, then message is discarded since it is sent by a parent node. Otherwise, it starts checking the MAC address. If the received MAC address matches the node own address, then it starts processing the message. If the received content is a result of an aggregation function, then it checks its content store for a matching content. If a content is found, it calculates its own result. If the received result and the computed results are different, then it will update its content store. Finally, the node checks its PIT table in order to send the data to the appropriate face. If an entry is found, the content message with the new result is sent. We have to note, that after sending the message, the PIT entry is not deleted immediately, but its delete timer is reset to allow other content messages to be forwarded using the same path to the sink node.

To considerably reduce the number of message generated by the aggregation function, we can use timers to differ the broadcasting of its content message. Thus, each node will send one message representing the result of a function of several values from different nodes instead of sending many messages representing the result of its own and received values. For example, a node may send the average of six temperature values instead of three messages each one representing the average of two values. In fact, the node waits until its children send back their sensed values to calculate the average value in only one operation.

C. Data aggregation example

Let's consider an example where the sink is interested in the average temperature value from the sensed values by the different nodes in the network. In this example, we suppose that each node sends immediately the result without waiting for incoming content messages from several low ranked nodes.

Figure 7 depicts the sequence of exchanged messages and the actions of each node to compute and provide a weighted average temperature to the sink. To calculate the weighted average temperature, the sink sends to its neighboring node (N1) a COLLECT Interest. After receiving that interest, N1 retransmits it and invokes a local operation based on available information in the CS. The result of that operation is sent back to the sink and the corresponding CS entry is updated. When N2 receives COLLECT Interest, it will accomplish the same task previously described. When N1 receives CONTENT message from N2, it will calculate the new average, transmit it to the sink and update CS. The same tasks are repeated each time a new message is received. The content message contains two data. The first data denotes the last computed value. The second data denotes the last sent value by the node. For each data, we associate a weight that denotes the number of used values to compute the average.

IV. Performance evaluation

First, we evaluated our implementation of the CCN communication layer in Contiki using the Cooja simulator [9] and TelosB physical nodes. Next we evaluated the aggregation service through simulation.

A. CCN layer evaluation

To validate and evaluate the performances of the implemented CCN communication layer, we developed an application that runs on top of the Contiki OS. The application provides a data collection scenario where a sink asks periodically the sensed data by a certain node. Each node creates a content with the gathered information and stores it locally in the Content Store. The content contains the length of the collection frame, the timestamp, the time synchronization status, the cpu time, the low power mode time, the transmit time, the listen time and the sensors values (battery voltage, battery indicator, light1, light2, temperature, humidity, radio intensity, etx1, etx2, etx3 and etx4). Nodes can update their contents by locally refreshing sensing data. The content stored at a Node N_i is identified by the name "$/COLLECT/Ni$". The sink node periodically sends an Interest message to each node requesting its available data. The Interest message holds the name "/COLLECT/Ni". If there are more than one collecting node in the network, each one has to have a unique identifier number i. Because the transmission is limited to broadcast, this is the only available method that guarantees data collection from a specific node and subsequently from all nodes. If two collecting nodes share the same identifier number, then their two contents will be referenced with the same name, which implies that only one content will be delivered to the sink in response to its request. In this application, the sink or the nodes receiving a propagating Content Object process the message without keeping a copy in their local Content Store. Giving the very limited memory resources on a sensor device, it would not support keeping more than few Content Objects in its CS. In addition, both sink and nodes have the prefix "/COLLECT" in their FIB tables so that they can forward the Interests generated by the sink and provide a hop-by-hop communication.

1) Simulation results: We have simulated, using Cooja, 4 scenarios of data collection with a varying number of nodes between 10 to 40 with a step of 10. In each scenario, there is only one sink node. A collecting node can exchange messages with the sink node and the other nodes using the broadcast mode. Sensor nodes are placed randomly in a 100m x 100m area. The sink transmits an Interest message every 20 seconds to a node N_i. Thus, a data collection round requires *N x 20 seconds* to request N deployed nodes.

Figure 8 depicts box plots of the obtained collection delay of a single interest and its data response sent by the sink under varying network sizes 10, 20, 30 and 40 nodes running on the Cooja simulator. We observe that the mean delay is close to the 25th percentile and remains stable around 219 ms while increasing the number of nodes. However the maximum delay reaches 512 ms and remains stable for 20, 30 and 40 nodes. Thus, we observe that on average a data requires 219 ms to be retrieved by the sink. We observe also that the collection delay is at minimum 211 ms for the different network sizes.

B. TelosB nodes experiments

We carried two test scenarios based on the same data collection application using a set of 10 TelosB sensor nodes. All nodes were placed randomly in the same room. The sink sends an Interest message each 20 seconds to a node N_i. A data collection round takes then 200 seconds to be done

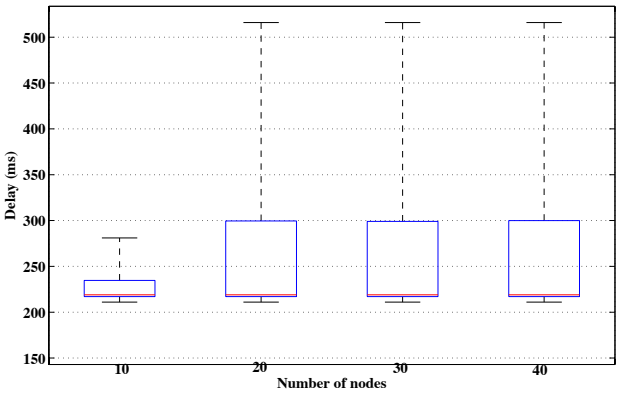

Fig. 8. Box plot of data retrieval delays measured at the sink node under varying network sizes.

over 10 nodes. A collecting node updates its locally collected information each 70 seconds by sensing its environment. In the first test scenario, data-collecting nodes do not store the Content Objects they may receive in their Content Stores. They simply forward them when they are allowed to. On the contrary, in the second test scenario, content storage in Content Store is enabled for a data-collecting node. Nodes update the stored Content Objects when receiving new contents from other nodes with the same names. We are interested in analyzing how content caching in the network's nodes may affect the time delays and the network load.

The results of the experiments using the TelosB sensor nodes lead to a set of observations about the storage of forwarded contents in CS. Table I shows the results of the data-collection application at the sink level for the two scenarios with a caching function on the CS respectively disabled and enabled. With an execution time of approximately 1h32 and an Interest generation rate of 1 message each 20 seconds, both applications in the two scenarios generated 276 Interest messages for data collection from nodes. This test validates the Interest generation and sending mechanism in the CCN stack. Theoretically, generating one Interest every 20 seconds in a period of 92 minutes leads to a total of 276 Interests (92 * 60 / 20 = 276). However, the CS-disabled application sent an additional 374 Interest messages, rising the total number of Interests sent over the CCN driver to 650 messages. This is due to the CCN forwarding mechanism which implies to forward a received Interest message if no matching content is found locally in the Content Store cache. The Interest forwarding led to more Content Object forwarding as well through the network. When a node receives an Interest message to which he has no matching data, it keeps a copy of it in the PIT table and forwards the message. If it receives a matching Content Object, it deletes the corresponding pending entry in the PIT and forwards the Content Object message. But nothing prevents the node from receiving the same Interest message again if the message is still transiting in the network. In this case, the node does not observe a corresponding pending entry in its PIT as it was cleared up and will then re-forward the Interest and eventually the Content Object if it will receive later a matching piece of content. When the application does not allow the CCN stack to store the forwarded contents locally, more data forwarding will be done. On the contrary,

in the case where forwarded contents are stored locally, if a node receives the same Interest message for a second time, it will directly reply with the matching content kept in the CS cache and the processing of the incoming Interest is finished. There will be in consequence no extra Interest forwarding and eventually less content forwarding which will make the network load less important. From the results in Table I, we observe that the network load is more important in the CS-disabled scenario with 151568 bytes received and 29118 bytes sent in comparison to the CS-enabled scenario results where only 60222 bytes are received and 7571 bytes are sent.

-	CS-disabled scenario	CS-enabled scenario
Execution time (seconds)	5539	5520
Generated Interests	276	276
Satisfied Interests	272	266
Total sent Interests	650	276
Bytes received	151568 (Interest: 16794 / Content Object: 134774)	29118 (Interest: 683 / Content Object: 28435)
Bytes sent	60222 (Interest: 13053 / Content Object: 47169)	7571 (Interest: 5567 / Content Object: 2004)
Average delay (ms)	315	272

TABLE I. THE RESULTS OF TESTS ON THE TELOSB NODES.

Thus, storing forwarded Content Objects locally in the Content Store cache has an effect on delay. As the content will spread over the network's nodes, the availability of the content will increase and the sink may receive the data it requests with better delays. The CS-enabled application has a better average delay of 272 ms per packet while the CS-disabled application has an average delay of 315 ms per packet. Table II shows the measured delay in the first 7 data collection rounds for the two scenarios. After few rounds, the round average delay of the CS-enabled application became lower than the delay of the CS-disabled application as illustrated in Figure 9.

Data Collection Round	1st Round	2nd Round	3rd Round	4th Round	5th Round	6th Round	7th Round
Average Delay with CS enabled (ms)	440,11	310,9	317,8	258,67	262,2	260,8	260
Average Delay with CS disabled (ms)	296	327,9	355,3	371,8	320,9	313,9	300,5

TABLE II. SUMMARY OF AVERAGE DELAYS WITH ENABLED AND DISABLED CACHING STRATEGY USING 10 TELOSB NODES.

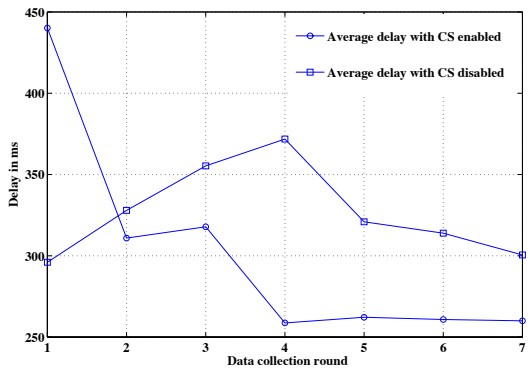

Fig. 9. Data retrieval average delays at the sink over several collection rounds with enabled and disabled caching strategies and using 10 TelosB nodes.

Delays of the CS-enabled scenario are regrouped in Table III, we can see that the 1st round average delay is the highest among the delays of the different rounds with a value of 440.11 ms. The average delay tends to decrease in the next data collection rounds as shown in Figure 10. Before the first collection round, the Content Store of each node holds only its proper content. When the data collection starts, for each Interest message, only one node has a matching content that it propagates in response to the received request. As long as a Content Object is spreading over the network to reach the sink, it is replicated at the intermediate nodes and kept in CS. From the second data collection round, sink can get the requested content from many nodes (the content's origin or the replicating nodes). By keeping the first Content Object received and discarding the duplicates, the sink may ensure to satisfy its Interest with better delays.

Data Collection Round	1st Round	2nd Round	3rd Round	4th Round	5th Round	6th Round	7th Round
Average Delay (ms)	440,11	310,9	317,8	258,67	262,2	260,8	260
Max Delay (ms)	547	461	453	265	265	273	265
Min delay (ms)	336	250	257	250	258	257	257

TABLE III. SUMMARY OF COLLECTION DELAYS ON DIFFERENT ROUNDS USING 10 TELOSB NODES AND AN ENABLED CACHING STRATEGY.

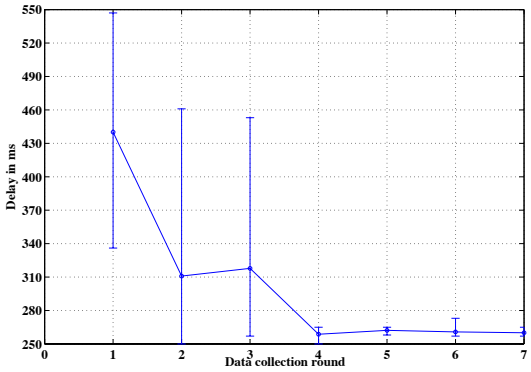

Fig. 10. Minimum, average and maximum data retrieval delays measured at the sink node using 10 TelosB nodes and an enabled caching strategy.

C. Aggregation service evaluation

Using the Cooja simulator, the tree building phase presented in Algorithm 1 is evaluated in terms of the number of exchanged messages between nodes. The number of sensing nodes in the network is varied between 1 and 10. As depicted in Figure 11, we observe that the number of exchanged messages increases linearly when increasing the number of nodes in the network. We can see that in order to assign ranks to N nodes there are $N + 1$ exchanged messages in the network. It is obvious, since to assign a rank, only a single Interest message is sent by each node of the network.

To evaluate the aggregation service in terms of the number of exchanged messages and the required time to obtain an aggregated value from a set of node, we instantiated a maximum aggregation function to be applied on the sensed temperature by each node. We placed the nodes using a linear topology where the sink the node is the first node of the line. We considered three scenarios. In the worst scenario, the maximum temperature value is provided by the node with the highest rank (at the end of the line). In the best case, the maximum temperature value is provided by the closest node to the sink.

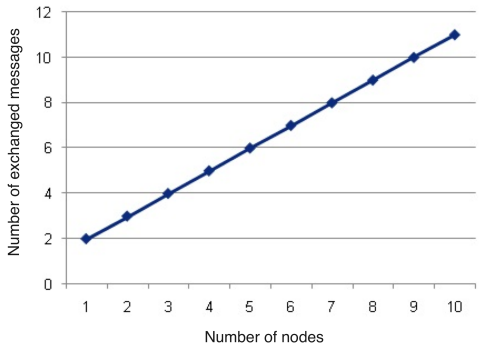

Fig. 11. Number of exchanged messages to build the collection tree.

The third scenario, denotes a case where the maximum value is provided by a node in the middle of the line.

Figure 12 shows the number of exchanged messages to calculate the maximum value for the three different scenarios, while varying the number of nodes in the network. In the best case scenario, we observe that the number of exchanged messages increases linearly. For example, with two nodes, the number of exchanged messages is 4, since we have 2 generated interest messages and only 2 content messages. The first content messages is sent by the closest node containing its maximum value and the second message is sent by the node at the end of the line. However, in the worst case scenario, the number of exchanged messages becomes 5. We have 2 interest messages and 3 content messages since the closest node to the sink, will forward the content message issued by the node providing the maximum temperature value.

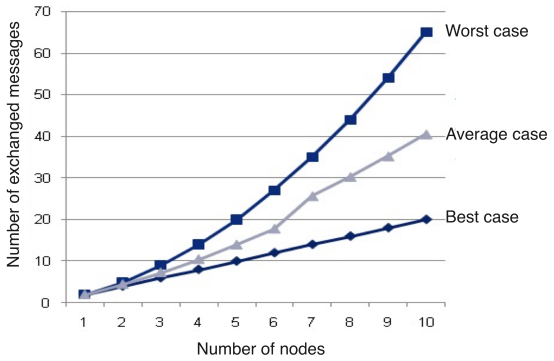

Fig. 12. Number of exchanged message to calculate the maximum sensed temperature in three scenarios.

Figure 13 shows the required time to calculate the maximum temperature value at the sink node. We observe that all the curves have a linear shape. This is due to the fact that the computation time depends mainly on the rank of the node having the maximum temperature value.

V. Conclusion and Future work

In this paper, we presented a novel named data collection protocol dedicated to wireless sensor networks. The proposed approach relies on an adaptation and integration of the CCNx protocol to fit with resources constrained devices and networks. We have evaluated the proposed communication layer using

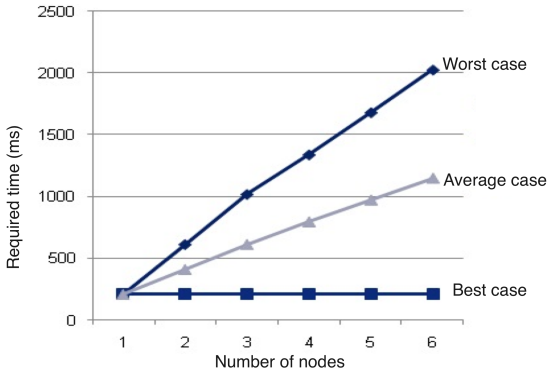

Fig. 13. Required time to obtain the maximum temperature value at the stink for the three scenarios.

real devices and simulation. In a next step, we have extended our approach with an aggregation service where the sink node is able to request data using an interest message and asks also the nodes to apply an aggregation function such as average or maximum. We have evaluated this service using simulation and we showed that it has a small overhead and good performance. In future work, our goal will be to make a real deployment of the named data aggregation service in WSN to collect for example the temperature within a building. We will also compare the performance of our solution with other protocols like SPIN [10] or RPL [8] based collection applications.

References

[1] M. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. Grieco, G. Boggia, and M. Dohler, "Standardized protocol stack for the internet of (important) things," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 3, pp. 1389–1406, 2013.

[2] 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

[3] M. Amadeo, C. Campo, A. Molinaro, and N. Mitton, "Named Data Networking: a Natural Design for Data Collection in Wireless Sensor Networks," in *IEEE Wireless Days (WD)*, Valencia, Espagne, Nov. 2013. [Online]. Available: http://hal.inria.fr/hal-00862236

[4] B. Saadallah, A. Lahmadi, and O. Festor, "CCNx for Contiki: implementation details," INRIA, Rapport Technique RT-0432, Nov. 2012. [Online]. Available: http://hal.inria.fr/hal-00755482

[5] "http://www.ccnx.org/releases/latest/doc/technical/ccnxprotocol.html," the CCNx protocol presentation.

[6] "http://www.ccnx.org," the CCNx project website.

[7] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, ser. LCN '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462. [Online]. Available: http://dx.doi.org/10.1109/LCN.2004.38

[8] J. Hui and J. Vasseur, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," Internet Requests for Comment, RFC Editor, Fremont, CA, USA, Tech. Rep. 6550, Mar. 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6550.txt

[9] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, 2006, pp. 641–648.

[10] J. Kulik, W. Heinzelman, and H. Balakrishnan, "Negotiation-based protocols for disseminating information in wireless sensor networks," *Wireless Networks*, vol. 8, no. 2-3, pp. 169–185, 2002.