MR-Edge: a MapReduce-based Protocol for IoT Edge Computing with Resource Constraints

Qian Wang, Brian Lee, Niall Murray, Yuansong Qiao
Software Research Institute
Athlone Institute of Technology
Athlone, Co. Westmeath, Ireland
{qwang|ysqiao|nmurray}@research.ait.ie, blee@ait.ie

***Abstract*—Edge computing is proposed to remedy the Cloud-only processing architecture for Internet of Things (IoT) because of the massive amounts of IoT data. The challenge is how to deploy and execute data processing tasks on heterogeneous IoT edge network. As MapReduce is a well-known model in Cloud computing for distributed processing of big data, this paper aims to devise a MapReduce-based protocol to achieve IoT edge computing. Our design is built upon the novel Information Centric Networking (ICN), which supports function naming and forwarding so as to facilitate task distribution among edge devices. To guarantee the correctness of task execution, a tree topology is formed in our approach to establish the logical connection between different types of edge devices, namely processing-capable nodes and forward-only ones. Moreover, the proposed protocol includes a task maintenance scheme that enables the coexistence of multiple IoT computation jobs. A testbed is implemented on ndnSIM to verify the feasibility of our design. The results show our approach could significantly decrease the network traffic compared with centralized data processing.**

Keywords—Internet of Things (IoT), Information Centric Networking (ICN), Edge computing, MapReduce programming

I. INTRODUCTION

As the Internet of Things (IoT) grows exponentially, so do the volumes of data it produces [1]. Cloud computing [2] becomes a popular choice to process the enormous amounts of IoT data because of its rich power and resources. However, this approach increases the network congestions when delivering all IoT data to the Cloud and then returning the analysed result back to the IoT network. Furthermore, IoT systems usually span widely with the deployment of a large number of devices to provide data, for example Smart Cities and intelligent traffic systems [3]. At least part of the raw data could be filtered and/or aggregated in a distributed manner instead of totally transferred and centralized manipulated by the Cloud. With more and more computing devices connected to IoT, it is feasible and rational to develop IoT edge computing [4] to solve the above mentioned problems, which requires lots of endeavours.

This paper mainly discusses two aspects. The first is in deployment of tasks at IoT edge network and then generation of execution plans to successfully complete the tasks. Due to the heterogeneity of IoT devices, some of them are able to execute tasks while some are not. It is necessary to assign tasks to appropriate devices. In addition, logical connections between IoT edge nodes should be established for every task, which may vary with different tasks requirements. This paper borrows the idea of MapReduce [5] as a solution, which is one of the powerful approach in distributed computing for big data. MapReduce enables second-order functions to take users' functions as input. IoT users could make use of this feature to define their desired operations. The outputs of MapReduce are key-value pairs, which is easier for different IoT applications to share or compare information. At least it pre-processes raw data for further usage.

The second challenge is to seek a better network support. This paper mainly focuses on the type of IoT edge computing that requires cooperation of heterogenous edge devices for each IoT application. The result is end-to-end IP communication model has inborn limitations to satisfy the requirement. It's not practical for IoT users to know the capability of each IoT edge device and then assign different tasks to the corresponding one. Moreover, emerging IoT applications are data-oriented as they desire to obtain knowledge analysed from lots of raw data rather than building connections with several IoT devices. Security and privacy concerns certainly requires the verification of identity, which are out of range of this paper. Hence, novel Information Centric Networking (ICN) [6] could be a potential solution for IoT with its focus on forwarding named content and functions.

Although many fruitful research projects are working on ICN in-network processing, fewer of them specifically designed for IoT scenarios. To name a few, [7] suggests the network to execute functions in sequence based on specific naming scheme, starting from data source to the functions one by one. This approach is limited to the cases that all requested data can be obtained at one source, which is rarely common in IoT. NFaaS [8] gives a solution to select the optimal edge node to execute a task. It assumes the function is undertaken on one node for every task and does not consider the relationship between different functions. MR-CCN [9] applies MapReduce over ICN for datacentre scenarios, whose routing path depends on the existing CamCube topology.

To this end, we propose MR-Edge, a MapReduce-based protocol to achieve IoT edge computing. It is inspired by the powerful MapReduce processing model to process massive amounts of IoT data in a distributed way. As IoT connects various kinds of devices, some of them have the computational resources to execute tasks (act as a Mapper or a Reducer in our design) but some do not (defined as Forwarders). Therefore, our design allocates tasks only to the capable IoT edge nodes to ensure tasks running successfully. For the edge devices cannot execute tasks, they aggregate multiple received packets into one and then forward to their neighbours. To guarantee the correctness of the final computation result, MR-Edge organizes the IoT edge nodes into a tree-based topology for each job. Additionally, a task maintenance scheme is devised to support multiple IoT applications at the same time.

The contributions of this paper are summarized as below:

- A protocol is designed to coordinate multiple heterogenous IoT nodes joining the edge computing, where some of them are processing-capable while some are not.
- A task maintenance scheme is proposed to coordinate IoT edge nodes working for multi-jobs simultaneously.
- An ICN naming scheme is defined to support task deployment, distribution and execution within IoT edge network.
- A testbed is developed on ndnSIM to verify the feasibility of our design.

The rest of this paper is organized as following: Section II present the related work. Section III shows the design of MR-Edge in detail. The experimental setup and evaluation results are presented in Section IV. Section V concludes the paper and discusses future work.

II. Related Work

IoT in-network processing has been studied by many researchers. For example T-Res [10] configures IoT sensor to direct communicate with actuators for monitoring tasks based on CoAP. Although there is no computing tasks involved, T-Res verifies the capability of IoT network to do simple data processing to be a complementary tool of Cloud processing. The authors [11] use the SDN controller to help distributing MapReduce function in WSN, which lacks of considering the coordination between reducers to process the data with same key. Moreover, it requires all nodes to maintain a flow table as a guideline to transmit and process data, which is not flexible to support complex computation logic.

Emerging IoT applications are data-oriented so that a few projects have applied ICN into IoT scenarios for better performance. Data retrieval is more efficient by ICN name-based forwarding [12] and data transmission is improved under poor network connectivity [13] with the support of ICN in-network caching and multicast. NFN [14] firstly extends the concept of named data to named functions. It enables in-network processing when both data and functions are found and available. However, it did not mention where and how the network deploys and executes data processing. NDN-Q [15] implements in-network query for vehicle networks by asking every network node to maintain their data on database model, which misses to specify how to form the execution graph when multiple nodes are selected to run data processing scripts. MR-CCN [9] is more related to our work, which implements MapReduce over CCN with the special focus on datacentre scenarios. However, their design lacks of developing a protocol to form the computational graph in generic topology, which is based on the routing path of the Camcube topology.

Our earlier work, MR-IoT [16] has verified to use ICN-based MapReduce model to process IoT data in distributed manner. It includes building a MapReduce tree on IoT network and then executes tasks along the tree edges. MR-IoT assumes that all IoT nodes are capable of processing data. Due to heterogeneous devices linked to IoT, their processing capability cannot be the same. For this reason, this paper proposes an improvement by considering the resource constraints on IoT nodes. The resource constraints may be caused by the heterogeneity of devices and/or the dynamic allocation of resources to jobs. As a result, the software-defined roles of edge devices vary for IoT task execution.

III. MR-Edge Design

To improve the performance of IoT data processing, this paper achieves IoT edge computing by exploiting the potential of IoT edge devices. Our design is built on Named Data Networking (NDN) [17], which facilitates IoT applications to issue requests including named functions and data. We consider the computational resource constraints on edge nodes and assign the computing tasks to the capable ones. An ICN-style MapReduce protocol is developed for the purpose of tasks' accuracy and multi-tasks coexisting.

A. Network Model and Assumptions

To implement edge computing for IoT tasks, the key challenge is how to deploy and execute it. This paper concentrates on one task type that requires the cooperation of multiple edge nodes. More specifically, it is called an MR-Edge job for each user's request and a job involves map and reduce tasks for MR-Edge mappers and reducers respectively. We assume each job requires real-time raw data so that different jobs do not share the same data or function. The processing requirement of all MR-Edge jobs is defined as the same data can only be processed once. Hence, we choose a tree topology to guarantee the accuracy of final results.

Every computing job has specific requirements not only on data and processing logic but also on the capability (e.g. CPU) of computing devices. It is another research topic of how to describe computing resources and select proper devices to satisfy different jobs, which is not the main concern of this paper. For simplicity, MR-Edge divides heterogeneous IoT edge nodes into two types: processing-capable (act as an MR-Edge mapper or reducer) and forward-only (a MR-Edge forwarder). Processing-capable nodes have enough computing resources to meet a job's requirement. Forward-only nodes cannot undertake tasks for current job as no computing resources available. All MR-Edge nodes join the procedure to build a tree for each job but only processing-capable nodes parse and run assigned (map or reduce) functions on required data.

Multiple IoT jobs can coexist in our design but we assume the network builds a unique execution tree for each job in sequence. Our design uses NDN routing protocol to build a shortest path tree for every job. The authors are aware that more algorithms (e.g. minimum spanning tree or Steiner tree [18]) should be applied to build a tree in order to meet specific job requirements or optimize IoT edge resources, which will be a part of future work. This paper aims to run multiple jobs on IoT edge network as first attempt.

B. Concept Overview

Fig. 1 is an example to illustrate how MR-Edge assigns different tasks to IoT nodes and then organizes them working together to accomplish different jobs. We currently assume the procedure of matching computational resource needs with available computing devices has been done. The result is every IoT edge node is either processing-capable or forward-only for MR-Edge jobs according to their computational resource status.

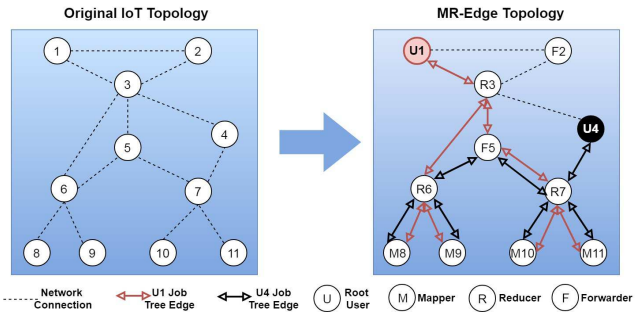

Fig.1 Network Topology: Original IoT VS. MR-Edge

- Processing-capable Nodes

The processing-capable nodes can undertake partial tasks for current MR-Edge jobs. Both MR-Edge mappers and reducers belong to this type of node. We regard the stub nodes of IoT edge network as Mappers (abbreviated as M in Fig.1), which connect with multiple sensors. They take raw sensing data as input, run user-defined map functions on the data and then output a series of key-value pairs. MR-Edge denominates other processing-capable nodes as reducers (abbreviated as R in Fig.1) which receive data from mappers, neighbour reducers or forward-only nodes. Reducers run user-defined reduce function on the data and return processed results to their upstream neighbours.

- Forward-only Nodes

Some IoT edge nodes cannot execute tasks because of resource constraints. They become MR-Edge forwarders, abbreviated as F in Fig.1. We set the forwarders at least forwarding packets from/to their neighbours if they are not capable of processing data. In detail, a forwarder receives Interests from their upstream neighbours and continues forwarding to their downstream neighbours on the job tree. They do not need to parse or run functions within the Interests. When a forwarder receives multiple Data packets for the same job tree, it integrates all received data into a single Data packet and then returns. Data aggregation helps to minimize the number of packet transmissions, which is a common and essential approach in WSN [19].

- Root User Nodes

Any node within IoT network could issue MR-Edge jobs, we define it as a root user node, abbreviated as U in Fig.1. A unique job tree is constructed for each root user before executing their jobs. The root user is the root of their tree with other MR-Edge reducers/forwarders as branches and mappers as leaves. A root user may also be a branch node in other job trees.

- Job Tree Edges

The dotted lines in Fig.1 are original IoT network connections. Each of them is possible to become an edge (solid lines with arrows and colours) on MR-Edge job trees. Some of them are shared by different job trees. However, the ring-connections have to be decomposed to guarantee the accuracy of MR-Edge jobs. Only the job tree edges will be used to exchange Interest and Data packets.

C. *Construction of Job Trees*

As IoT edge network contains both processing-capable and forward-only devices, the problem is how to coordinate them to achieve IoT edge computing. MR-Edge enables all nodes to communicate with each other in order to establish a tree-based logical connection for computing jobs. The job tree is formed based on NDN routing table. Every node has its own table so that it knows how to reach a specific node from itself. However, a node has no idea of others' routing information. All nodes need to exchange their information to form a tree.

- Interest naming for trees construction

Tree construction is launched when a root user wants to issue jobs. We define a BuildJobTree Interest for this procedure. It is written as below with a slight difference:

/NeighborName/BuildJobTree/JobID (a)

/NeighborName/BuildJobTree/JobID/UpstreamNodeName (b)

BuildJobTree Interest (a) is created and sent out by root users. MR-Edge reducers and forwarders rewrite Interest (a) as (b) and then forward to their neighbours. There are at most four parts within a BuildJobTree Interest: (1) */NeighborName* is the name of each neighbour with original IoT network connections. For example in Fig. 1 when R3 receives a BuildJobTree Interest from U1, it will ask F2, U4, F5 and R6 for U1's JobID. (2) */BuildJobTree* is the identifier for MR-Edge nodes to trigger the procedure of building job trees. (3) */JobID* is the combination of a root user's name plus a random number. MR-Edge requires all root users to generate their own job ID before sending a BuildJobTree Interest. The combination ensures each job ID is unique. (4) */UpstreamNodeName* is a required part if MR-Edge reducers or forwarders need to discover its downstream neighbours for current job. For example in Fig. 1 both R3 and R7 receive a BuildJobTree Interest (a) from U4. They rewrite Interest (a) as (b) by adding their name at the end of the Interest. In this way, F5 could know which BuildJobTree Interest is sent by whom when it receives two requests to build a tree with same JobID.

- Procedure of trees construction

MR-Edge introduces two tables in the application layer so that it can work for different root users at the same time. One is Job Tree Table (JT-Table) which stores information in "JobID – TaskNeighbours" pairs and is also used for executing multiple jobs (explained in sub-section D). The other is BuildJobTree Table (BJT-Table) which temporarily saves the replies from neighbour nodes about whether joining current job tree or not. Every row in this table is "NeighborName – PendingReply" and all records will be erased after the procedure of building current tree is done.

All MR-Edge nodes participate in job trees construction. Reducers and forwarders deal with two events in this procedure: receiving BuildJobTree Interests and sending out BuildJobTree Interests. Mappers only receive and reply BuildJobTree Interests.

Fig.2 presents the main steps to build job trees. When a reducer or forwarder receives a BuildJobTree Interest (a), it firstly retrieves the JobID within the Interest and check if the JobID exists in the JT-Table. If it exists and the corresponding TaskNeighbours are not null, the discovery of current job tree has been done and the root user can issue tasks. If not, the second step is the reducer/forwarder parses the JobID to get the root user's name and checks if this Interest is sent by the neighbour on its NDN routing table to reach the root user (named as SelectedUpstream). If the received BuildJobTree Interest is not from the

SelectedUpstream, the reducer/forwarder replies "no" immediately. It means the reducer/forwarder will not use this path to return data for current job tree. Otherwise, it jumps to the third step which continues exploring downstream neighbours for this tree before replying to its SelectedUpstream. However, if a node has no child node to provide data, it will also reply "no" immediately to its SelectedUpstream and ends the tree construction. For example, F2 in Fig. 1 receives the BuildJobTree Interest from U1 but it cannot join U1's job tree as no child node existing.

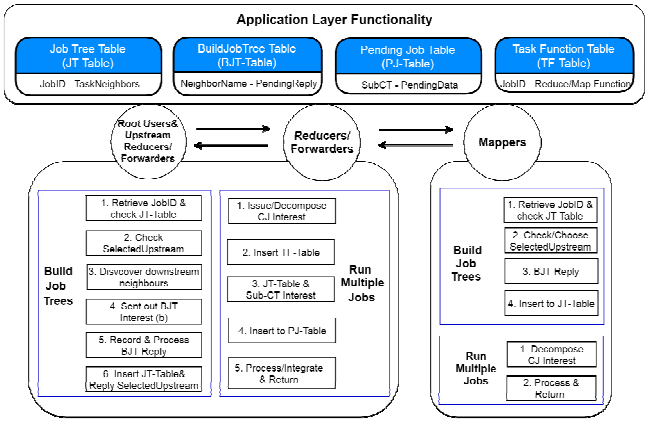

Fig.2 MR-Edge Protocol Design

For instance R3 in Fig. 1 sends out four BuildJobTree Interests for the task tree U1 (e.g. JobID: U1&1) and then waits replies from F2, U4, F5 and R6. In detail, F2 replies "no" as it has no child node. U4 also replies "no" as its downstream neighbour R7 does not choose it as the SelectedUpstream. Both F5 and R6 answer "yes". For the JT-Table of R3, one row is inserted and looks like "U1&1 - /F5/R6". Similarly, when the task tree of U4 (e.g. JobID: U4&1) is ready, R3 does not join this tree so that its JT-Table is not updated. However, the JT-Table of F5 changes and the table has two records: "U1&1 – /R6/R7" and "U4&1 – R6".

MR-Edge mappers maintain their JT-table as "JobID – SelectedUpstream" pairs. One mapper may receive multiple BuildJobTree Interests from different reducers/forwarders. When a mapper receives a new BuildJobTree Interest, it retrieves the JobID and checks its JT-Table as the first step, shown in Fig.2. If there has been a record for the same tree and this Interest is not from the stored SelectedUpstream, the mapper replies "no" to this neighbour. If the received JobID is a new one, each mapper only selects one reducer or forwarder as the upstream for each job tree. The last step of mappers is to insert the SelectedUpstream to its JT-Table. Mappers currently do not discover downstream neighbours during job trees construction because we assume mappers process the data from all connected sensors.

D. *Multiple Jobs Execution*

Fig.2 illustrates the workflow of MR-Edge nodes to execute jobs. Root users trigger this procedure by issuing jobs which is called ComputingJob (CJ) Interest and written as (c). In detail, */JobNeighbours* are the neighbours' name stored in the JT-Table. */JobID* is used to retrieve information from JT-Table. The rest of CJ Interest (*/MapFunction/ReduceFunction/contentFilter)* is real job content and defined by each root user.

/JobNeighbours/JobID/MapFunction/ReduceFunction/contentFilter (c)

Two tables are designed to guarantee the accuracy of running multiple jobs. Pending Job Table (PJ-Table) stores "JobID – PendingData" pairs and Task Function Table (TF-Table) is for "JobID – Reduce/Map Functions". Every job is sent by root users, traverses intermediate reducers and forwarders and finally reaches mappers. The procedure of job processing is in the reverse direction of job dissemination.

MR-Edge reducers have both PJ-Table and TF-Table. When a reducer receives a CJ Interest, it firstly decomposes the Interest into two parts: JobID and job content (defined functions and required data). Secondly, the reducer parses the job content to get the specific reduce function and stores in its TF-Table. Thirdly, the reducer searches the JobID in its JT-Table in order to get the TaskNeighbours for current tree. Then it rewrites the received CJ Interest by adding its own TaskNeighbours at the beginning (we call it subCJ Interest for clarity) and continues sending out. Fourthly, a new row is inserted into the reducer's PJ-Table for every subCJ Interest in the format of "subCJ - 0". The content "0" will be updated when corresponding Data packet is received. The PJ-Table is created for every task so that different Data packets can be appropriately processed. Finally, when all TaskNeighbours reply for the subCJ Interests of the same job, the reducer retrieves corresponding reduce function in its TF-Table, runs the function on all received data and then returns the processed result to its SelectedUpstream or the root user.

The steps for MR-Edge forwarders to support multiple jobs are similar with the described workflow of reducers. There are two differences should be mentioned. One is forwards have no TF-Table as they do not need to run user-defined functions on received data. The other is forwarders aggregate multiple subCJ for the same job into one and then return to its SelectedUpstream.

When a job is received by MR-Edge mappers, they firstly decompose CJ Interest to get the user-defined map function and store it in their TF-Table. Every mapper gathers the data from connected sensors and run map function to process. The outputs of mappers are key-value pairs and returned to their SelectedUpstream. All mapper data is further processed by the reducers at each level of the job tree. The root user gets the final result(s) returned from its TaskNeighbours, which is organized as (d).

k1,v1/k2,v2/.../k,v** (d)

IV. Evaluation and Analysis

We run a series of tests to verify the feasibility of MR-Edge and also compares the IoT network traffic produced with and without our approach. All simulations are performed on ndnSIM [20] and a network generator BRITE [21] is used to create the original topology of IoT. The total number of reducers and forwarders are fixed to twenty (Node Id 0-19) during all MR-Edge tests. Any of them may act as a user node to issues tasks. To measure the changes of network traffic, we adjust the proportion of reducers/forwarders within the network as well as the number of mappers connected to each reducer and forwarder.

Two types of data transmission speed (bandwidth + delay) are set for the simulation: 250 Kbits per second + 10 milliseconds between a MR-Edge mapper and a MR-Edge reducer/forwarder, based on the Zigbee protocol. 54 Mbits per second + 1 millisecond between MR-Edge reducers and forwarders using the IEEE 802.11 parameter.

To check jobs execution on MR-Edge, we configure three root user nodes (Node 3, 6 and 12), five forwarders (Node 15, 16, 17, 18 and 19) and the rest twelve nodes as reducers. We also set every root user starts to issue tasks at different time, in detail Node 12 starts at 0ths, Node 6 at 3rds and then Node 3 at 6ths. For clarity, we show the figures that every reducer/forwarder connects with one mapper for running multiple jobs. As we can see in Fig.3, original IoT network (with no job tree existing) generated by BRITE has ring-connections. In order to execute a single job, we observe whether MR-Edge could decompose overlapped IoT connections into a tree-based graph. For multiple jobs running, we check if MR-Edge separates different jobs, deals with corresponding data and then returns all results correctly.

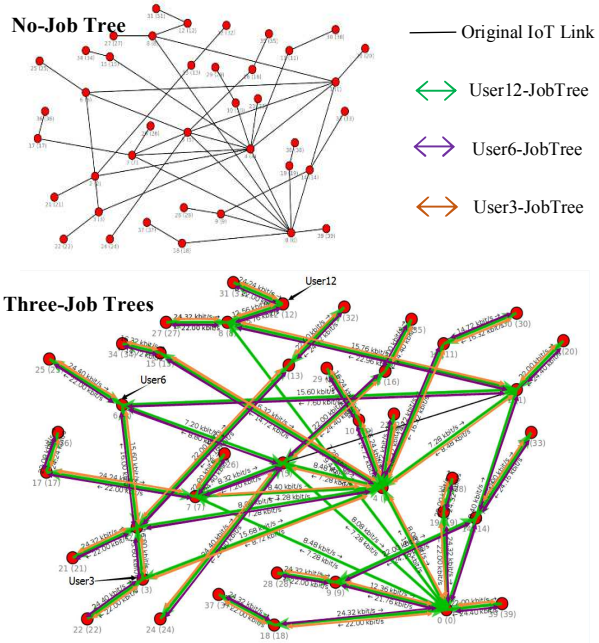

Fig.3 Network Topology

Fig.3 shows the changes of IoT link load when executing no job up to three jobs. All IoT nodes run MR-Edge protocol to build a unique job tree for each root user. Consequently, only the edges on corresponding job tree are used by MR-Edge nodes to forward Interest/Data to their job neighbours. The green lines with arrows are the job tree edges for user-12, the purple ones are for user-6 and the oranges ones are for user-3. There is still a black line existing with three trees running. It means this network link is not used by current three jobs. After all three job trees are built, we continue running the simulation for 20s with the frequency of 10 Interests per second per user. The results shows MR-Edge could complete all jobs by returning the correct result to each root user.

To evaluate the performance of MR-Edge, a comparison study (Central-User) is designed to make the root user requesting all sensor data directly and centralized processing by itself. Node 3, 6 and 12 are still the root user nodes for Central-User tests and they send Interest (e) to request data. We also examine the effect of three proportions of MR-Edge reducers/forwarders on the network traffic. In specific, the combinations are: 10Reducer-10Forwarder (abbreviated as MR-Edge-10R10F), 15Reducer-5Forwarder (MR-Edge-15R5F) and 20Reducer-0Forwarder (MR-Edge-20R0F). The computational job Interest used for MR-Edge tests is expressed as (f). As mappers are assumed to gather data from IoT sensors, three data sizes produced by the mappers are simulated: 25, 50 and 75 bytes respectively. Moreover, we set N as the size of one received packet for a reducer to process. We change the size of a reducer's output data from N/2, 2N/3 to N and record corresponding network traffic. Each test lasts for 20s with the Interest sending frequency of 10 per second.

/SensorName (e)

/JobNeighbours/JobID/Map(k,v)->(k+1,v*2)/Reduce(v1,v2)>(v1+v2)/allSensor (f)

The results of network traffic are summarized in Fig.4. It is obvious that Central-User tests produce more data traffic within the network compared with any test of our design. As the user needs to send one Interest for each sensor data, afterwards the same number of Data is traversed through the whole network back to the user node. This procedure rises the network traffic sharply. Along with the sensor data size grows, the data transmission by the Central-User approach shifts from almost 1798 to 2745 kilobytes (red colour filled columns).

The performance of MR-Edge is related to the number of reducers within the network as well as the data compression rate of reducers. Both of the two factors have a positive effect on decreasing network traffic. When the sensor data size is fixed and the data reduce rate is the same, the more reducers exist in MR-Edge, the less IoT data traversed within the network. For example, when the sensor data is 25 bytes and the data reduce rate is N/2, the network traffic could save 20% when all IoT edge nodes are MR-Edge reducers (produces approximately 1508 kilobytes) compared with Central-User processing (about 1798 kilobytes). The network traffic is also less when the same number of reducers use more efficient data processing functions to process fixed quantity of sensor data.

With more sensor data produced, the network traffic generated by our design rises in a range of 268 to 381 kilobytes with different number of reducers and data reduce rates. However, the comparison study has a more significant growth (946 kilobytes) for centralized processing. As a result, MR-Edge not only enables IoT edge computing but also greatly lowers the volume of transmitted data within IoT network.

V. Conclusion and Future Work

The rapid expansion of IoT connects more and more computing devices to the network. The potential of these devices could be explored in order to improve the performance of IoT. However, the heterogeneity nature of IoT network determines that the computing resources of IoT devices cannot be the same. More specifically, the capability of IoT nodes decides whether they can execute tasks. Thus, this paper proposes MR-Edge that organises multiple IoT edge devices with different computing resources to cooperate with each other for completing IoT tasks.

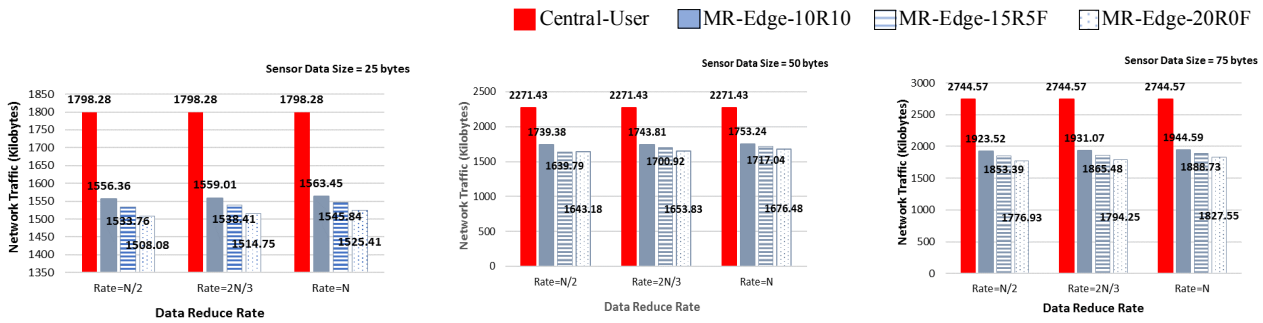

Fig. 4 Network Traffic Comparison

Our design defines three roles according to the status of computational resources on IoT nodes: mappers (run user-defined map functions on sensor data), forwarders (data aggregation only, with no processing capability) and reducers (run user-defined reduce functions on data received from mappers, forwarders or neighbour reducers). A shortest path tree is built for each MR-Edge job to satisfy the computation-once requirement for the same data. To execute multiple jobs simultaneously, we develop the application layer functionality to manage job trees and requested functions and data. The simulation tests are performed on ndnSIM, which verifies the feasibility of our design. In addition, MR-Edge could significantly lower IoT network traffic compared with centralized processing.

The future work will mainly make efforts on two aspects. One is to develop a computation-aware algorithm to build the job tree or other graph for different IoT applications. The other is the optimization on computational resources for multiple jobs. Both requires to formulate specific objective functions with consideration of network conditions (e.g. bandwidth, link delay), job types (e.g. the same data computed exactly/at least/at most once), node energy consumption and so on. As our design is based on ICN, naming scheme and/or name resolution may also require improvements for our future work.

ACKNOWLEDGMENT

This publication has emanated from research supported by research grants from Athlone Institute of Technology under President's Seed Fund 2016 and Science Foundation Ireland (SFI) under Grant Numbers 13/SIRG/2178 and 16/RC/3918, co-funded by the European Regional Development Fund.

REFERENCES

[1] "Number of Connected IoT Devices Will Surge to 125 Billion by 2030, IHS Markit Says - IHS Technology." [Online]. Available: https://technology.ihs.com/596542/number-of-connected-iot-devices-will-surge-to-125-billion-by-2030-ihs-markit-says. [Accessed: 16-Apr-2018].

[2] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," Future Gener. Comput. Syst., vol. 56, pp. 684–700, 2016.

[3] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges," in 2012 10th International Conference on Frontiers of Information Technology, 2012, pp. 257–260.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, Oct. 2016.

[5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008.

[6] G. Wang et al., "Design Considerations for Applying ICN to IoT." [Online]. Available: https://tools.ietf.org/html/draft-irtf-icnrg-icniot-01. [Accessed: 30-Apr-2018].

[7] L. Liu et al., "Demonstration of a Functional Chaining System Enabled by Named-Data Networking," in Proceedings of the 3rd ACM Conference on Information-Centric Networking, New York, NY, USA, 2016, pp. 227–228.

[8] M. Kro; and I. Psaras, "NFaaS: Named Function as a Service," presented at the 4th ACM Conference on Information-Centric Networking (ICN 2017), Berlin Germany, 2017.

[9] F. Pianese, "Information Centric Networks for Parallel Processing in the Datacenter," in 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops, 2013, pp. 208–213.

[10] D. Alessandrelli, M. Petraccay, and P. Pagano, "T-res: Enabling reconfigurable in-network processing in iot-based wsns," in Distributed Computing in Sensor Systems (DCOSS), 2013 IEEE International Conference on, 2013, pp. 337–344.

[11] A. C. G. Anadiotis, G. Morabito, and S. Palazzo, "An SDN-Assisted Framework for Optimal Deployment of MapReduce Functions in WSNs," IEEE Trans. Mob. Comput., vol. 15, no. 9, pp. 2165–2178, Sep. 2016.

[12] M. Amadeo, C. Campolo, and A. Molinaro, "Multi-source data retrieval in IoT via named data networking," in Proceedings of the 1st international conference on Information-centric networking, 2014, pp. 67–76.

[13] M. Amadeo, C. Campolo, and A. Molinaro, "Information-centric networking for connected vehicles: a survey and future perspectives," IEEE Commun. Mag., vol. 54, no. 2, pp. 98–104, 2016.

[14] C. Tschudin and M. Sifalakis, "Named functions and cached computations," in Named functions and cached computations, Las Vegas, NV, USA, 2014.

[15] W. Drira and F. Filali, "NDN-Q: An NDN query mechanism for efficient V2X data collection," in 2014 Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking Workshops (SECON Workshops), 2014, pp. 13–18.

[16] Q. Wang, B. Lee, N. Murray, and Y. Qiao, "MR-IoT: An information centric MapReduce framework for IoT," in 2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC), 2018, pp. 1–6.

[17] L. Zhang et al., "Named Data Networking," SIGCOMM Comput Commun Rev, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[18] B. Wang and J. C. Hou, "Multicast routing and its QoS extension: problems, algorithms, and protocols," IEEE Netw., vol. 14, no. 1, pp. 22–36, Jan. 2000.

[19] V. Potdar, A. Sharif, and E. Chang, "Wireless sensor networks: A survey," in Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on, 2009, pp. 636–641.

[20] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2: An updated NDN simulator for NS-3," Technical Report NDN-0028, Revision 2, NDN, 2016.

[21] "BRITE: Boston university Representative Internet Topology gEnerator," Jul-2017. [Online]. Available: https://www.cs.bu.edu/brite/. [Accessed: 28-Sep-2017].