Traffic Load Balancing Among Brokers at the IoT Application Layer

---

Citation:
X. Sun and N. Ansari, "Traffic Load Balancing Among Brokers at the IoT Application Layer," in *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 489-502, March 2018, DOI: 10.1109/TNSM.2017.2787859

URL:
https://ieeexplore.ieee.org/document/8241428/

Traffic Load Balancing among Brokers at the IoT Application Layer

Xiang Sun, *Student Member, IEEE,* and Nirwan Ansari, *Fellow, IEEE*

***Abstract*—At the Internet of Things (IoT) application layer, a physical phenomenon, which is sensed by a server (i.e., an IoT device), is defined as an IoT resource. In this paper, we propose to cache popular IoT resources in brokers, which are considered as the application layer middleware nodes. Caching popular resources in the brokers is to move the traffic loads (for delivering the up-to-date contents of the resources) from the servers (which host these popular resources) to the brokers, thus reducing the energy consumption of the servers. However, many brokers may be geographically distributed in the network and caching popular resources in nearby brokers may result in unbalanced traffic loads among the brokers, and may thus dramatically increase the average delay of the brokers in delivering the contents of their cached popular resources to clients. To reduce the average delay among the brokers, we propose to re-cache/re-allocate the popular resources from heavily loaded brokers into lightly loaded brokers in order to balance the traffic loads among brokers. We formulate the popular resource re-caching problem as an optimization problem, which is proven to be NP-hard. We design the Latency awarE populAr Resource re-cachiNg (LEARN) algorithm to efficiently solve the problem, and demonstrate the performance of LEARN via simulations.**

***Index Terms*—Internet of things, resource caching, popularity, CoAP, CoAP Pub/Sub, broker**

I. Introduction

Internet of Things (IoT) is an intriguing paradigm to interconnect smart devices by applying various networking technologies. In order to achieve interconnections, a flexible three-layer IoT architecture, which comprises the perception layer, the network layer, and the application layer [1], [2], has been proposed. In the perception layer, smart devices generate different types of data streams representing the states of the physical world. These data streams are transmitted at the network layer by applying different kinds of communications technologies. The application layer provides high-level functionalities, such as IoT content retrieval services, data management, and access control. In this paper, we focus on analyzing the communications protocols and functionalities at the IoT application layer.

IoT devices are mostly battery-constrained [3], and it is thus not energy efficient to enable an IoT device (e.g., a temperature sensor) to send its IoT content (e.g., the current temperature value) to a large number of clients, who try to retrieve the content of the IoT device. Caching IoT contents in the network layer is proposed to solve the energy inefficiency problem and speed up the content delivery process. Specifically, network nodes (e.g., routers and gateways) would cache the received IoT contents based on their caching strategies [4]–[8]. Then, if a network node receives a content retrieval request (from a client) and it has the requested content, the network node would reply to this request without forwarding it to the original IoT device. However, this would incur the cache inconsistency problem [9]–[11], i.e., the cached content may not accurately reflect the current state of the corresponding IoT device. For example, the content generated by a parking spot sensor indicates the state of the parking spot (i.e., empty or occupied). Suppose that the parking spot is initially empty and this content is cached by a network node. Afterwards, the parking spot becomes occupied and a client, whose content retrieval request is responded by the network node (which previously cached the content of the parking spot being empty), may not obtain the correct state of the parking spot. The reason for the cache inconsistency problem is the transparent nature of IoT contents (i.e., the value of an IoT content is quickly diminished, and thus this IoT content needs to be replaced by a fresh one) and the local caching decision made by network nodes (i.e., an IoT device is unaware of its content having been cached by the network nodes. The IoT device is thus unable to update the caches in those network nodes).

In order to solve the cache inconsistency as well as the energy inefficiency problem during the IoT content delivery process, we propose to cache IoT resources at the application layer. Specifically, an IoT resource (which is different from an IoT content) is defined as a specific physical phenomenon captured by a specific IoT device. Yet, an IoT content represents the state of a physical phenomenon at a specific moment. For instance, a temperature sensor is to sense the temperature value of Bob's smart home and the current temperature value is 30^oC. Then, "the temperature value of Bob's smart home" is an IoT resource (which is hosted by the temperature sensor) and "30^oC" is an IoT content. Caching an IoT resource at the application layer indicates that the IoT resource is cached in a broker, which is an IoT middleware with computing, communications, and storage capabilities. Thus, the IoT device will send the up-to-date content to the broker and clients can retrieve the content of the IoT resource from the broker rather than the IoT device. Based on the proposed IoT resource caching method, we will solve the following two problems:

- Which IoT resources are suitable to be cached in brokers in order to minimize the energy consumption of IoT devices?
- If an IoT resource is suitable to be cached in brokers, which broker should cache the IoT resource such that the traffic loads among brokers are balanced?

X. Sun and N. Ansari are with Advanced Networking Lab., Department of Electrical & Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA. E-mail:{xs47, nirwan.ansari}@njit.edu.

The main contributions of the paper are:

1) In order to solve the cache inconsistency and the energy inefficiency problem during the IoT content delivery process, we propose to cache IoT resources in brokers. We illustrate how to achieve IoT resource caching based on the current IoT application layer communications protocols.
2) We demonstrate that caching IoT resources in brokers may not always benefit IoT devices in reducing their energy consumptions. We propose to cache popular resources in brokers to minimize the energy consumptions of IoT devices. We provide a method of measuring the popularity of an IoT resource.
3) We point out that the traffic loads among brokers may be unbalanced, thus increasing the average delay of brokers in transmitting the contents of the cached popular resources to the clients. Therefore, we propose to re-cache/re-allocate the popular resources from heavily loaded brokers into lightly loaded brokers in order to balance the traffic loads among brokers.
4) We formulate the popular resource re-caching problem as an optimization problem and prove it to be NP-hard. We design the Latency awarE populAr Resource re-cachiNg (LEARN) algorithm to solve the problem and demonstrate the performance of LEARN via simulations.

The rest of the paper is organized as follows. In Section II, we briefly review the related works. In Section III, we illustrate how to implement IoT resource caching based on the current IoT application layer communications protocols. In Section IV, we provide the definition of a popular IoT resource and illustrate how to measure the popularity of IoT resources. In Section V, we propose to balance the traffic load among brokers by re-caching popular IoT resources. We formulate the popular IoT resource re-caching problem and demonstrate the problem to be NP-hard. In Section VI, we propose the LEARN algorithm to efficiently solve the problem. In Section VI, we demonstrate the performance of LEARN via simulations. The conclusion is presented in Section VII.

II. Related Works

In order to avoid unnecessary End-to-End (E2E) communications, in-network caching has been proved to be an effective way to speed up the content delivery process [12]–[14], i.e., some popular contents would be cached in network nodes such that clients can retrieve these contents without explicitly contacting content providers. The feasibility of using in-network caching technologies for IoT has been discussed in the Information-Centric Networking Research Group (ICNRG) under the Internet Research Task Force (IRTF) [9], [15]. However, the traditional in-network caching strategies applied in Content Delivery Networks (CDNs) may not suitable for the IoT system owing to the unique features of IoT [4], [8]. First, most of the IoT devices are resource constrained, and so the main objective of content caching placement in IoT is to minimize the energy consumption rather than to minimize the delay for delivering contents to users in CDNs. Second, the contents generated by IoT servers exhibit transient feature [4], [16]; this feature is quite different from the contents cached in CDNs, whose popularity remains stable over long timescale. Typical examples include popular news with short videos, which are updated every 2-3 hours; new movies, which change their popularity every week; new music videos, which change their popularity about every month [17]. Third, the sizes of IoT contents may be smaller than the sizes of contents in CDNs, but the number of IoT contents may be larger than the number of contents in CDNs.

Owing to these unique features of the IoT system, many in-network content caching strategies in the context of IoT has been proposed. Vural *et al.* [4], [5] proposed an in-networking caching method to facilitate IoT content caching. Specifically, IoT contents are cached in the edge routers and they argued that clients' obtaining the contents from the edge routers may lose freshness (i.e., the obtained data may not be up-to-date) but reduce the network traffic as compared to the clients' obtaining the contents from the original servers. Thus, they dynamically modified the edge routers' content caching probabilities in order to optimize the tradeoff between content freshness and network traffic. Similarly, Hail *et al.* [6] proposed a network layer IoT content caching strategy in the multi-hop wireless network scenario, in which IoT devices are equipped to cache the forwarding contents. They designed a novel distributed probabilistic caching strategy, which is based on the freshness of the content as well as the energy level and the storage capability of the device, in order to improve the energy efficiency of the IoT devices and reduce the content delivery delay. Niyato *et al.* [18] considered the case that the contents generated by the IoT devices should be cached in the local wireless access point and clients should always retrieve the corresponding contents from the wireless access point. They designed an optimal caching update period for each IoT device (in updating its content in the wireless access point) in order to maximize the hit rate in terms of the probability that the clients can successfully obtain the corresponding contents. In order to solve the cache inconsistency problem, Ha and Kim [19] proposed to enable each IoT device to select and maintain a set of network nodes in caching its content. Since the IoT device is aware of which network nodes have cached its content, the IoT device is able to update the cached content in these network nodes to guarantee the consistency. This approach, however, incurs heavy burdens on the IoT device, i.e., the IoT device needs to periodically obtain the information from all the network nodes to determine which network nodes are suitable to cache its content; meanwhile, the IoT device needs to send the up-to-date content to all the network nodes (which have cached the content) by itself.

Different from the above works, we propose to cache popular IoT resources at brokers to reduce the energy consumption of IoT devices and accelerate content delivery. Essentially, caching popular IoT resources at brokers can solve the cache inconsistency problem without introducing heavy burdens to IoT devices.

III. Implementation of caching IoT resources in brokers

In this section, we illustrate the implementation of caching IoT resources in brokers by applying the current application

layer communications protocols.

A. Constraint Application Protocol

CoAP [20] is originally designed for communications among resource constrained devices. CoAP assumes two logical roles, i.e., server and client. A server is a resource host, which generates contents of the resource. A client is a resource requester, which tries to retrieve contents of the resource. A resource is an object reflecting a specific physical phenomenon; normally, a resource is identified by a Uniform Resource Identifier (URI) [21]. For instance, if Bob's mobile phone tries to obtain the current temperature value provided by the temperature sensor, which is equipped in Bob's smart home, the temperature sensor is a server, Bob's mobile phone is a client, and "the temperature value in Bob's smart home" is a resource hosted by the server; meanwhile, this resource can be identified by a URI (i.e., a unique address that can be searched on the Internet), such as "coap://bob_home/temp". The interactions between a server and a client follow a request/response model. For instance, as shown in Fig.1, the client (e.g., Bob's mobile phone) sends a resource retrieval request, which includes the URI that points to a specific resource in the server (e.g.,"the temperature value in Bob's smart home" resource hosted by the temperature sensor). Consequently, the server responds to the resource retrieval request by sending the content of the resource to the client.

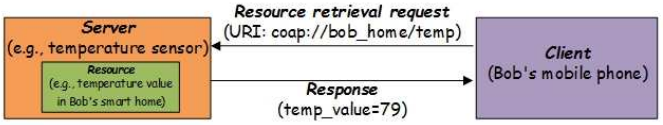


Fig. 1: CoAP request/response interaction model.

A client can continuously observe the resource by sending a resource observe request to the server, which hosts the resource. The resource observe request should contain the URI of the resource as well as the observe conditions, which indicate the criteria for the server to transmit the contents of the resource to the client. For instance, as shown in Fig.2, Client-1 (e.g., bob's mobile phone) sends a resource observe request to the server (e.g., temperature sensor), which indicates that Client-1 tries to observe the resource (which is identified by the resource URI) and the server should send the up-to-date content of the resource every $5\ mins$.

We can divide the clients into two classes: *read clients* and *observe clients*. The read clients send resource retrieval requests to obtain the contents of resources in the corresponding servers. Normally, the servers would not store any information (e.g., URIs) of these clients. The observe clients send resource observe requests to continuously monitor the status of resources. The servers should maintain the URIs of the servers and their corresponding observe conditions by storing them in the local binding table. As shown in Fig.2, two clients (e.g., bob's mobile phone and the air conditioner) send the resource observe requests to the server (e.g., temperature sensor). The server should create a binding table [22], which maintains the URIs of the two clients and their corresponding observe conditions[1]. Consequently, the server can send the contents of the resource to the corresponding clients once their observe conditions are satisfied.

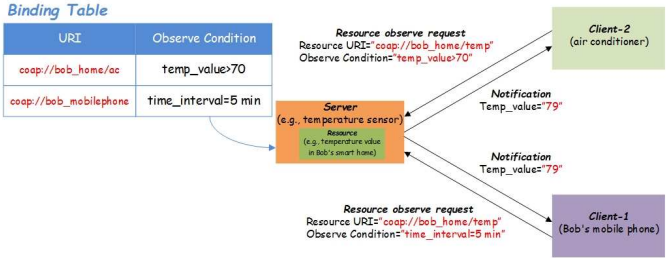


Fig. 2: Clients observe the resource.

B. Resource Directory

Clients need to know the URIs of the resources before they can send the resource retrieval/observe requests to obtain the contents of the resources. In order to enable clients to discover the URIs of the interested resources, another entity, i.e., Resource Directory (RD), has been proposed [23]. An RD hosts the descriptions of resources and provides the resource lookup functionality to clients. Note that the descriptions of a resource include the URI and the context information (e.g., the resource type and the resource location) of the resource. Fig. 3 illustrates the interactions among a server, a client, and an RD. First, a server registers its hosting resource to an RD by sending a resource registration request, which includes the descriptions of the resource, to the RD[2]. Second, the RD stores the descriptions of the resource into its database and returns the database entry ID of the resource[3] (e.g., /rd/1001) to the server. Third, a client may send a resource discovery request to discover the URI of a specific resource. Note that the resource discovery request may include a set of query criteria (e.g., resource type='temperature sensor' and location='bob's home') describing the resource that the client wants to discover. As a result, the RD would return the URI(s) of the resource(s), which matches the query criteria.

C. Caching IoT resources in brokers

The interactions among servers, clients, and RDs enable the clients to find their interested resources' URIs and obtain the contents of these resources. However, it is not efficient when many clients try to obtain the same resource during a time period, i.e., the server may transmit a huge amount of data to these clients. This situation happens when some popular events take place. For example, when a football game is held in a stadium, tens of thousands of smart cars would look for empty parking spots near the stadium. Thus, each street parking meter may need to transmit a huge amount of data to these smart cars

[1]The paper uses different terminologies, from those applied in the corresponding IETF RFCs and drafts.

[2]Note that the URI of the RD is well-known or the server can discover the RD by broadcasting/multicasting an RD discover message [23].

[3]The database entry ID of the resource identifies the location of the resource in the RD's database. The server should know this information such that it can update or delete the descriptions of the resource in the RD later on.

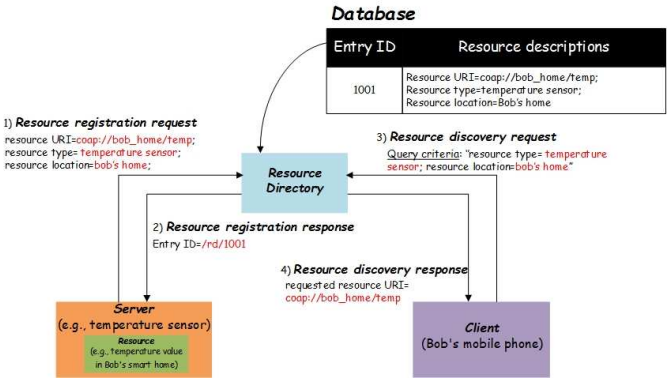


Fig. 3: The interactions among server, client, and RD.

in order to report its parking status (i.e., if the parking spot is empty and when it will be empty if it is currently occupied). Obviously, it is not efficient to enable the street parking meters to transmit the huge amount of data, which may exhaust the network resources and the energy supplies of the street parking meters (which may be powered by batteries). Note that exhausting the network resources of the street parking meters results in increased delay for the street parking meters in transmitting the contents to the clients, and exhausting the energy supplies of the street parking meters disables the street parking meters from sensing and transmitting data.

In order to efficiently deliver the contents of resources, a new entity, i.e., broker, is introduced by the CoAP Publish/Subscribe protocol [24]. A broker is an application layer middleware node equipped with powerful hardwares and sufficient energy supplies. One of the functionalities of the broker is to cache IoT resources. Specifically, a broker can cache a resource hosted by a server, and thus the server would periodically transmit the up-to-date content of the resource to the broker, which consequently helps the server forward the content of the resource to the clients upon requests. Therefore, enabling the broker to deliver the content of the resource has the potential to reduce the traffic loads of the server, which may finally reduce the energy consumption of the server. The communications for the server in enabling the broker to deliver the content of the resource is illustrated in Fig.4. 1) The server would first send a resource creation request to the broker to cache a resource. The resource creation request should contain the resource URI (indicating which resource is requested to be cached) as well as the binding table information associated with this resource. 2) The broker would send the URI of corresponding resource cached in the broker back to the server if the broker determines to cache the resource. Note that there are currently two URIs related to this resource: the URI of the resource hosted by the server and the one cached in the broker. 3) After receiving the response from the broker, the server would periodically send the up-to-date content of the resource to the broker. 4) Meanwhile, the server should update the URI of the resource in the RD's database such that the clients can find the URI of the resource cached in the broker. 5) The read clients, who are interested in the resource, can find the URI of the resource via the RD. 6) Consequently, these clients can obtain the up-to-date content of the resource by sending resource retrieval/observe requests to the broker. 7) The broker sends the content of the resource to the observe clients based on the information in the corresponding binding table.

IV. Popular resource caching at the IoT application layer

In this section, we first illustrate that caching IoT resources may not always reduce the energy consumption of their servers. In order to minimize the energy consumption of servers, we propose to cache IoT resources only if they are popular. We provide a method to measure the popularity of an IoT resource. Table I summarizes the main notations applied in the rest of the paper.

TABLE I: List of Important Notations

Notation	Definition
$\mathcal{I}$	Set of all the IoT resources in the network.
$\mathcal{J}$	Set of all the popular IoT resources in the network.
$\mathcal{K}$	Set of all the brokers in the network.
l_i	Average content size of resource i.
λ_i	Average resource retrieval request arrival rate of resource i.
ψ_i	Content update rate of resource i.
η	Predefined threshold to measure the popularity of IoT resources.
x_{jk}	Binary cache indicator between popular resource j and broker k.
λ_k	Average resource retrieval request arrival rate of broker k.
μ_k	Average service rate of broker k.
t_k	Average delay of broker k.
ρ_k	Average network resource utilization of broker k.

A. Definition of popular resources

Caching a resource in a broker cannot always benefit the server (which hosts the resource) because once the resource is cached in the broker, the server needs to periodically send the up-to-date content of the resource to the broker (in order to keep the content of the resource in the broker fresh) even if no client is interested in the resource. On the other hand, if the resource is not cached in the broker (i.e., the server would respond to the resource retrial requests by itself), the server does not need to transmit any data when no client is interested in the resource. Therefore, caching the resource in the broker may not always reduce the traffic load of the server. Note that a heavier traffic load of the server incurs a higher energy consumption of the server.

In order to minimize the energy consumption of servers, only popular resources should be cached in brokers. A resource is considered popular if caching this resource by a broker will result in traffic load reduction of its hosting server by η, where $\eta \geq 0$ is the predefined traffic load threshold. Specifically, denote $\mathcal{I}$ as the set of resources hosted by the servers in the network and i is used to index these resources. Denote the average content size of resource i as l_i and the average resource retrieval request arrival rate (i.e., the average

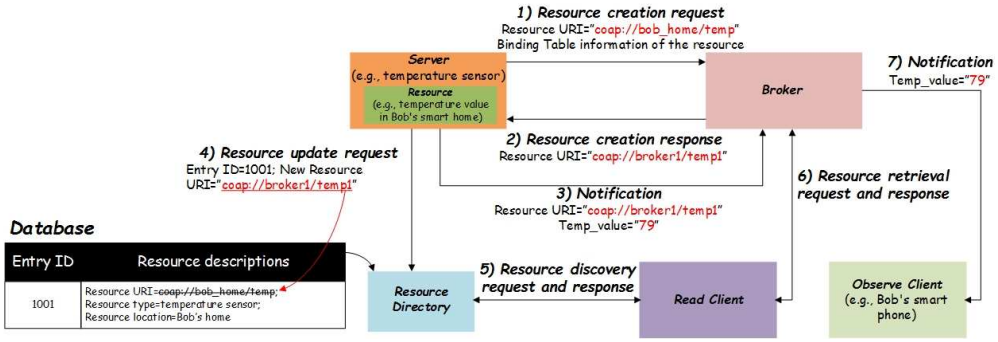
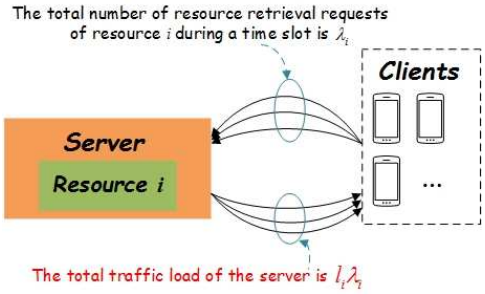
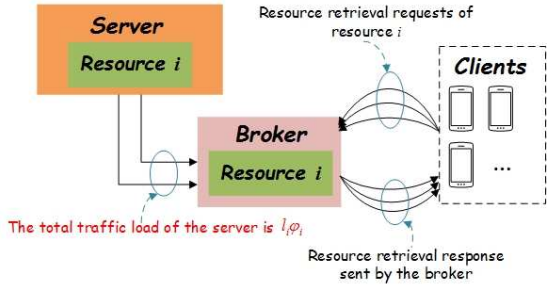


Fig. 4: The interactions among server, client, RD, and broker.



(a) The scenario of resource i not being cached in a broker.



(b) The scenario of resource i being cached in a broker.

Fig. 5: The traffic load of the server in different scenarios.

number of resource retrieval requests[4] per second during a time slot) for resource i as λ_i. As shown in Fig. 5a, if resource i is not cached in a broker, i.e., the server (which hosts resource i) needs to respond to the resource retrieval requests, the traffic load of the server is $l_i\lambda_i$. On the contrary, as shown in Fig. 5b, if resource i is cached in the broker, the server only needs to periodically send the up-to-date content of resource i to a broker, which will eventually help the server respond to the resource retrieval requests. Denote ψ_i as the content update rate of resource i (i.e., the average number of times that the server sends the up-to-date content of resource i to the broker per second during a time slot). Then, if resource i is cached in the broker, the traffic load of the server becomes $l_i\psi_i$. Based on the above, we define the resources that satisfy the following equation to be the popular resources:

$$l_i\left(\lambda_i - \psi_i\right) > \eta,\ \ i \in \boldsymbol{\mathcal{I}}. \tag{1}$$

Essentially, Eq. 1 indicates that popular resource i being cached in a broker can reduce at least η amount of traffic load of the server as compared to popular resource i not being cached in a broker. Note that reducing the traffic loads of the server implies decreasing the energy consumption of the server. Therefore, if a resource becomes popular, it is suitable to be cached in a broker.

B. Monitoring the popularity of a resource

If a resource is not currently cached in a broker, the server should monitor the number of resource retrieval requests of the resource in each time slot. If the resource becomes popular (i.e., Eq. 1 is satisfied), the server would request to cache the resource in a broker by sending a resource creation request to the broker. On the other hand, if the resource is currently cached in the broker, the broker would take the responsibility to monitor the number of resource retrieval requests of the resource in each time slot. If the resource is not popular (i.e., Eq. 1 is not satisfied), the broker would send a resource deletion notification to the corresponding server (which hosts this resource) to imply that the resource is no longer suitable to be cached in the broker. Consequently, the server would respond to the resource retrieval requests by itself.

C. Broker deployment

A broker can be a logical entity (i.e., a function embedded in a router/switch/gateway) or a physical entity (such as a cloudlet [25]–[27], a fog node [28], etc.). A good broker deployment enables each server to discover and communicate with at least one specific broker such that this server's hosting resources can be cached in the corresponding broker. Note that applying different broker deployments does not affect the problem (which will be mentioned in Section V) and the related algorithm. In order to better express our ideas, we provide one possible broker deployment strategy, which

[4]There are many resource retrieval requests related to each resource. These resource retrieval requests comprise two parts, i.e., the resource retrieval requests from read clients and the resource retrieval requests from observe clients (note that read clients and observe clients are defined in Section III.A). The resource retrieval requests from read clients are generated and sent by the read clients. Yet, the resource retrieval requests from observe clients are automatically generated by the server/broker once some observe conditions in the binding table are satisfied. For instance, an observe client tries to obtain the content of a resource every 5 mins; thus, the server/broker would automatically generate a resource retrieval request for the observe client every 5 mins and send the content to the observe client accordingly.

leverages the edgeIoT architecture [28], as an example. Specifically, as shown in Fig. 6, each Base Station (BS), which has already been deployed in the mobile network and provides high radio coverage, is equipped with multiple wireless interfaces (such as Zigbee and low power area network) such that different servers can communicate with the corresponding BSs [29]. Thus, each BS is considered as a smart gateway to provide various communications interfaces to its local servers. Meanwhile, each BS is connected to a broker, which may comprise a number of interconnected physical machines to provide the IoT resource caching functionality. Thus, each server can actually communicate with the broker via the corresponding BS. In addition, each broker can communicate with the Internet as well as other BSs/brokers via the Software Defined Networking (SDN) based cellular core [30]–[33]. In the SDN-based cellular core network, OpenFlow switches are adopted to separate out all control functions from the data forwarding function. All the switches are controlled by the SDN controller via the OpenFlow protocol [34]. The OpenFlow controller manages the forwarding plane of BSs and OpenFlow switches, monitors the traffic at the data plane, and establishes user sessions.

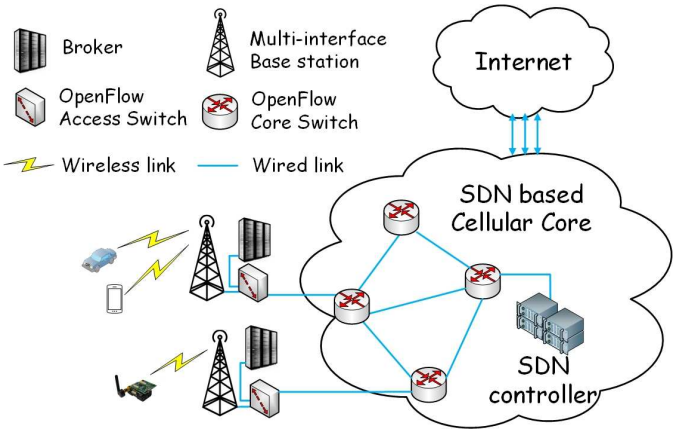


Fig. 6: A possible solution of broker deployment.

The mentioned broker deployment provides each server with the flexibility in caching its hosting popular resource in a broker, i.e., a resource can be cached in the local broker, whose connected BS is the gateway of the server, or in a neighboring broker via the SDN based cellular core. If a popular resource is cached in the broker, the broker is responsible for delivering the contents of the resource to the clients via the SDN based cellular core and the Internet[5].

V. Traffic Load Balancing Among Brokers

Many brokers can be geographically distributed in the network. Normally, if a resource becomes popular, the server would select the local broker, which incurs the smallest Round Trip Time (RTT), to cache its popular resources. This may lead to unbalanced traffic loads among brokers. For example, as shown in Fig. 7, broker-1 caches two popular resources, i.e., resource-1 and resource-2, and broker-2 caches one popular resource, i.e., resource-3. Thus, broker-1 needs to respond to $2n$ and $3n$ resource retrieval requests related to resource-1 and resource-2 during a time slot, respectively. While, broker-2 needs to respond to n resource retrieval requests related to resource-3 during a time slot. If the sizes of these resources' contents are the same, broker-1 would transmit more data to the clients than broker-2 would, i.e., traffic loads are unbalanced between the two brokers. The unbalanced traffic loads may significantly increase the average delay of delivering the contents of popular resources to clients.

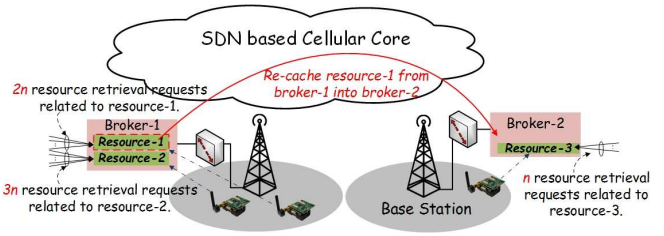


Fig. 7: The illustration of unbalanced traffic loads among brokers and the resource re-caching process.

In order to reduce the average delay, traffic loads can be offloaded from heavily loaded brokers to lightly loaded brokers. For example, as shown in Fig. 7, broker-1 can offload its traffic loads to broker-2 by enabling broker-2 to cache resource-2 (such that broker-2 should take the responsibility to respond to the resource retrieval requests related to resource-2). Here, we define the process of a popular resource, which is originally cached by one broker, to be cached by another broker as resource re-caching/re-allocation. Essentially, balancing the traffic loads among brokers is implemented by resource re-caching/re-allocation. Note that a popular resource can only be cached among brokers in the same broadcast domain[6]. This can actually prevent a popular resource being cached by a broker far away from the server (which hosts the resource).

A. Average delay model

The average delay of a broker in delivering the traffic load comprises two parts, i.e., the average queueing delay of the traffic waiting in the network queue of the broker and the average transmission delay of the broker in transmitting the traffic. Note that the average network delay of delivering the traffic to the clients over the SDN-based mobile core network and the Internet is not considered in the paper.

Suppose the resource retrieval request arrival process for resource i during a time slot follows a Poisson process with the average arrival rate λ_i $(requests/sec)$. Denote $\mathcal{J}$ ($\mathcal{J} \subseteq \mathcal{I}$) as the set of popular resources in the network, i.e., $\mathcal{J} = \{i | l_i\,(\lambda_i - \gamma_i) > \eta\}$, and j is used to index these popular resources. Meanwhile, denote $\mathcal{K}$ as the set of brokers in the same broadcast domain and k is used to index these brokers. Denote x_{jk} as a binary variable indicating whether popular

[5]In this paper, we consider the case that all the clients are Internet clients, i.e., clients would retrieve the contents of IoT resources via the Internet.

[6]A number of proximity brokers would be assigned the same vLAN tag by the SDN controller. The brokers having the same vLAN tag are considered to be in the same broadcast domain, where the brokers can share their information by broadcasting them in the domain.

resource j should be cached in broker k (i.e., $x_{jk}=1$) or not (i.e., $x_{jk}=0$). Thus, the total number of resource retrieval request arrivals in broker k, which is the sum of the resource retrieval request arrivals of popular resources that are cached by broker k, also follows a Poisson process with the average arrival rate

$$\varsigma_k = \sum_{j\in\boldsymbol{\mathcal{J}}} \lambda_j x_{jk}. \tag{2}$$

Assume that the content sizes of resources are the same, denoted as l, and the data transmission rate of broker k, denoted as r_k, is deterministic, which depends on the network interface capacity of broker k. Thus, the service rate of broker k (i.e., the number of the resource retrieval requests can be responded by broker k per second), denoted as μ_k, is also deterministic, where $\mu_k = \frac{r_k}{l}$ $(requests/sec)$. Thus, the process of broker k in delivering the contents of popular resources in response to the received resource retrieval requests can be considered as an M/D/1 queueing model and the average delay (i.e., the queueing delay plus the transmission delay) for broker k in response to a resource retrieval request can be derived as

$$t_k = \underbrace{\frac{1}{\mu_k}}_{transmission\ delay} + \underbrace{\frac{\rho_k}{2\mu_k\left(1-\rho_k\right)}}_{queueing\ delay}, \tag{3}$$

where ρ_k indicates the average network resource utilization of broker k, i.e., $\rho_k = \frac{\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{jk}}{\mu_k}$.

Note that a smaller value of t_k indicates broker k can deliver its traffic load (i.e., the contents of its cached resources in response to the received resource retrieval requests) to the clients faster. From E.q. (3), we can derive that as the average network resource utilization of a broker increases, the average queueing delay of the broker would exponentially increase, thus dramatically increasing the average delay of the broker. If the average network resource utilization of a broker equals to 1, its average delay goes to infinity.

B. Problem formulation

We formulate the popular resource re-caching problem as follows:

$$\boldsymbol{P0}:\ \underset{\boldsymbol{\rho},\boldsymbol{\mathcal{X}}}{argmin} \sum_{k\in\boldsymbol{\mathcal{K}}}\left(\frac{1}{\mu_k}+\frac{\rho_k}{2\mu_k\left(1-\rho_k\right)}\right), \tag{4}$$

$$s.t. \quad \forall k\in\boldsymbol{\mathcal{K}}, \rho_k\mu_k = \sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{jk}, \tag{5}$$

$$\forall k\in\boldsymbol{\mathcal{K}}, 0<\rho_k<1, \tag{6}$$

$$\forall j\in\boldsymbol{\mathcal{J}}, \sum_{k\in\boldsymbol{\mathcal{K}}} x_{jk}=1, \tag{7}$$

$$\forall j\in\boldsymbol{\mathcal{J}}, \forall k\in\boldsymbol{\mathcal{K}},\ x_{jk}\in\{0,1\}. \tag{8}$$

Here, $\boldsymbol{\rho}=\{\rho_k|k\in\boldsymbol{\mathcal{K}}\}$ denotes the average network resource utilization of all the brokers and $\boldsymbol{\mathcal{X}}=\{x_{jk}|j\in\boldsymbol{\mathcal{J}},k\in\boldsymbol{\mathcal{K}}\}$ denotes popular resources to be cached by brokers. The objective of $\boldsymbol{P0}$ is to minimize the total average delay of the brokers. Constraint (5) implies the network resource utilization of each broker. Constraint (6) imposes that the network resource utilization of a broker should be less than 1 in order to keep the system stable and reliable. Constraint (7) means each popular resource should be cached by a specific broker.

Theorem 1. $\boldsymbol{P0}$ is NP-hard.

Proof: By substituting Constraint (5) (i.e., $\rho_k = \frac{\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{jk}}{\mu_k}$) into $\boldsymbol{P0}$, $\boldsymbol{P0}$ can be transformed into

$$\begin{aligned}\boldsymbol{P1}:\quad & \underset{\boldsymbol{\mathcal{X}}}{argmin} \sum_{k\in\boldsymbol{\mathcal{K}}} \frac{2\mu_k-\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{jk}}{2\mu_k\left(\mu_k-\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{jk}\right)},\\ s.t. \quad & \forall k\in\boldsymbol{\mathcal{K}}, \sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{jk}<\mu_k,\\ & Constraints\ (7)\ and\ (8).\end{aligned} \tag{9}$$

Thus, proving $\boldsymbol{P0}$ to be NP-hard is transformed to proving $\boldsymbol{P1}$ to be NP-hard, which can be demonstrated by proving the decision problem of $\boldsymbol{P1}$ to be NP-complete [35]. The decision problem of $\boldsymbol{P1}$ can be described as follows: given a positive value b, is it possible to find a feasible solution $\boldsymbol{\mathcal{X}}$ for $\boldsymbol{P1}$ such that the total average delay of the brokers is less than b, i.e.,

$$\sum_{k\in\boldsymbol{\mathcal{K}}} \frac{2\mu_k-\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{jk}}{2\mu_k\left(\mu_k-\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{jk}\right)} \le b, \tag{10}$$

and all the constraints of $\boldsymbol{P1}$ (i.e., Constraints (7), (8), and (9)) are satisfied?

Assume that $b\to+\infty$, i.e., Eq. (10) always holds for any solution. Then, the decision problem of $\boldsymbol{P1}$ is transformed into whether a feasible solution $\boldsymbol{\mathcal{X}}$, which satisfies Constraints (7), (8), and (9), exists or not. Now, we consider the scenario that there are only two brokers in the network (i.e., $k\in\{1,2\}$) and the service rate of the two brokers are the same, i.e., $\mu_1=\mu_2=\frac{1}{2}\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j+\varepsilon$, where ε is a very small positive value. Then, the decision problem of $\boldsymbol{P1}$ can be transformed into: whether there exists a feasible solution $\boldsymbol{\mathcal{X}}$, which meets the following constraints,

$$\begin{cases}\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{j1}=\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{j2}=\frac{1}{2}\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j;\\ \forall j\in\boldsymbol{\mathcal{J}},\ x_{j1}+x_{j2}=1;\\ \forall j\in\boldsymbol{\mathcal{J}},\ x_{j1}\in\{0,1\},\ x_{j2}\in\{0,1\}.\end{cases}$$

The above problem is essentially a partition problem, i.e., whether the set of popular resources $\boldsymbol{\mathcal{J}}$ can be allocated into two brokers such that the traffic load of the two brokers[7] are the same, i.e., $\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{j1}=\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{j2}=\frac{1}{2}\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j$. Therefore, the partition problem is reducible to the decision problem of $\boldsymbol{P1}$. The partition problem is a well-known NP-complete problem, and thus the decision problem of $\boldsymbol{P1}$ is also an

[7]We assume that the content size of each resource is the same, and thus the traffic load of broker k refers to the sum of the average number of resource retrieval request arrival rates of the popular resources, which are cached by broker k, in the rest of the paper, i.e., $\sum_{j\in\boldsymbol{\mathcal{J}}}\lambda_j x_{jk}$.

NP-complete problem. Consequently, $\boldsymbol{P1}$ is NP-hard, which further indicates $\boldsymbol{P0}$ is NP-hard. ■

VI. LATENCY AWARE POPULAR RESOURCE RE-CACHING(LEARN)

In order to efficiently solve $\boldsymbol{P0}$, we design the novel LEARN algorithm. The basic idea of LEARN is to decompose $\boldsymbol{P0}$ into two sub-problems, i.e., *optimal network utilization assignment among brokers* and *popular resource re-allocation*, and sequentially solve them to obtain the optimal resource caching vector $\boldsymbol{\mathcal{X}} = \{x_{jk}|j \in \boldsymbol{\mathcal{J}}, k \in \boldsymbol{\mathcal{K}}\}$. Note that the *optimal network utilization assignment* problem is to obtain the optimal network utilization of each broker (i.e., $\boldsymbol{\rho}$) to minimize the total average delay of all the brokers. The *popular resource re-allocation* problem is to allocate the popular resources (i.e., $\boldsymbol{\mathcal{X}}$) among brokers such that the sum of the difference between the actual network utilization and the optimal network utilization of each broker is minimized.

A. Optimal network utilization assignment among brokers

Constraint (5) is the coupling constraint that defines the relationship between $\boldsymbol{\rho}$ and $\boldsymbol{\mathcal{X}}$. Removing the coupling constraint allows $\boldsymbol{P0}$ to be solved via two separate simpler problems. In order to remove the coupling constraint, we relax Constraints (5) by summing ρ_k for all $k \in \boldsymbol{\mathcal{K}}$, i.e.,

$$\sum_{k\in\boldsymbol{\mathcal{K}}} \mu_k\rho_k = \sum_{k\in\boldsymbol{\mathcal{K}}}\sum_{j\in\boldsymbol{\mathcal{J}}} \lambda_j x_{jk} = \sum_{j\in\boldsymbol{\mathcal{J}}}\left(\lambda_j \sum_{k\in\boldsymbol{\mathcal{K}}} x_{jk}\right). \quad (11)$$

Since $\sum_{k\in\boldsymbol{\mathcal{K}}} x_{jk} = 1$ ($\forall j \in \boldsymbol{\mathcal{J}}$), we have

$$\sum_{k\in\boldsymbol{\mathcal{K}}} \mu_k\rho_k = \sum_{j\in\boldsymbol{\mathcal{J}}} \lambda_j, \quad (12)$$

and thus $\boldsymbol{P0}$ can be transformed into

$$\begin{aligned} \boldsymbol{P2}: \quad \boldsymbol{\mathcal{Z}}(\boldsymbol{\rho}) = & \underset{\boldsymbol{\rho}}{argmin} \sum_{k\in\boldsymbol{\mathcal{K}}}\left(\frac{1}{\mu_k} + \frac{\rho_k}{2\mu_k(1-\rho_k)}\right), \\ & s.t. \quad \sum_{k\in\boldsymbol{\mathcal{K}}} \mu_k\rho_k = \sum_{j\in\boldsymbol{\mathcal{J}}} \lambda_j, \\ & \forall k \in \boldsymbol{\mathcal{K}}, \quad 0 < \rho_k < 1. \end{aligned}$$

The physical meaning of $\boldsymbol{P2}$ is to calculate the optimal average network utilization of each broker such that the total average delay of all the brokers is minimized. Note that $\boldsymbol{P2}$ is not related to $\boldsymbol{\mathcal{X}}$ and can be easily solved based on the following two lemmas.

Lemma 1. $\boldsymbol{P2}$ is a convex problem.

Proof: The constraints of $\boldsymbol{P2}$ are linear, and thus we only need to prove the objective function of $\boldsymbol{P2}$ is convex, i.e., the Hessian matrix of the objective function is positive definite [36]. For every $k \in \boldsymbol{\mathcal{K}}$, we have

$$\frac{\partial^2 \boldsymbol{\mathcal{Z}}}{\partial\rho_k\partial\rho_k} = \frac{4\mu_k}{(1-\rho_k)^3} > 0, \quad \frac{\partial^2 \boldsymbol{\mathcal{Z}}}{\partial\rho_k\partial\rho_{\iota\in\boldsymbol{\mathcal{K}}\setminus k}} = 0.$$

Thus, the Hessian matrix of $\boldsymbol{\mathcal{Z}}$ can be expressed as

$$\boldsymbol{\mathcal{H}} = \begin{vmatrix} \frac{\partial^2 \boldsymbol{\mathcal{Z}}}{\partial\rho_1\partial\rho_1} & \cdots & \frac{\partial^2 \boldsymbol{\mathcal{Z}}}{\partial\rho_1\partial\rho_{|\boldsymbol{\mathcal{K}}|}} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 \boldsymbol{\mathcal{Z}}}{\partial\rho_{|\boldsymbol{\mathcal{K}}|}\partial\rho_1} & \cdots & \frac{\partial^2 \boldsymbol{\mathcal{Z}}}{\partial\rho_{|\boldsymbol{\mathcal{K}}|}\partial\rho_{|\boldsymbol{\mathcal{K}}|}} \end{vmatrix} = \begin{vmatrix} \frac{4\mu_1}{(1-\rho_1)^3} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{4\mu_{|\boldsymbol{\mathcal{K}}|}}{\left(1-\rho_{|\boldsymbol{\mathcal{K}}|}\right)^3} \end{vmatrix},$$

where $|\boldsymbol{\mathcal{K}}|$ indicates the total number of brokers. Obviously, $\boldsymbol{\mathcal{H}}$ is a symmetric matrix with all the diagonal entries positive, i.e., $\boldsymbol{\mathcal{H}}$ is positive definite. Therefore, $\boldsymbol{\mathcal{Z}}$ is a convex function and $\boldsymbol{P2}$ is a convex problem. ■

Lemma 2. $\boldsymbol{P2}$ has a close form solution $\boldsymbol{\rho}^* = \{\rho_k^*|k \in \boldsymbol{\mathcal{K}}\}$, where

$$\rho_k^* = 1 - \frac{\sum\limits_{k\in\boldsymbol{\mathcal{K}}} \mu_k - \sum\limits_{j\in\boldsymbol{\mathcal{J}}} \lambda_j}{\mu_k\,|\boldsymbol{\mathcal{K}}|}. \quad (13)$$

Proof: Since $\boldsymbol{P2}$ has been proved to be a convex problem, the corresponding Karush Kuhn Tucker (KKT) conditions can be obtained as follows:

$$\forall k \in \boldsymbol{\mathcal{K}}, \quad \begin{cases} \frac{1}{2\mu_k(1-\rho_k)^2} + \alpha_k - \beta_k - \mu_k\gamma = 0, \\ \alpha_k(\rho_k - 1) = 0, \\ \beta_k\rho_k = 0, \end{cases} \quad (14)$$

$$\sum_{k\in\boldsymbol{\mathcal{K}}} \mu_k\rho_k = \sum_{j\in\boldsymbol{\mathcal{J}}} \lambda_j, \quad (15)$$

where α_k, β_k, and γ are the Lagrangian multiplexers. In order to satisfy Eq. 14, we can derive $\alpha_k = 0$, $\beta_k = 0$, and

$$\rho_k = 1 - \sqrt{\frac{1}{2\mu_k^2\gamma}}. \quad (16)$$

Substitute Eq. 16 into Eq. 15, and we have

$$\gamma = \frac{1}{2}\left(\frac{|\boldsymbol{\mathcal{K}}|}{\sum\limits_{k\in\boldsymbol{\mathcal{K}}} \mu_k - \sum\limits_{j\in\boldsymbol{\mathcal{J}}} \lambda_j}\right)^2. \quad (17)$$

We can derive the optimal solution ρ_k^* by substituting Eq. 17 into Eq. 16, i.e., $\rho_k^* = 1 - \frac{\sum_{k\in\boldsymbol{\mathcal{K}}} \mu_k - \sum_{j\in\boldsymbol{\mathcal{J}}} \lambda_j}{\mu_k|\boldsymbol{\mathcal{K}}|}$. ■

B. Popular resource re-allocation

Note that $\boldsymbol{\rho}^*$ indicates the average optimal network utilization of all the brokers in order to minimize the average delay of all the brokers. The next step is to re-allocate the popular resources among brokers such that the average network utilization of each broker (i.e., $\frac{\sum_{j\in\boldsymbol{\mathcal{J}}} \lambda_j x_{jk}}{\mu_k}$) can be as close as to its optimal value (i.e., ρ_k^*). Thus, we formulate the popular resources re-allocation problem as follows:

$$\begin{aligned} \boldsymbol{P3}: \quad & \underset{\boldsymbol{\mathcal{X}}}{argmin} \sum_{k\in\boldsymbol{\mathcal{K}}}\left|\rho_k^* - \sum_{j\in\boldsymbol{\mathcal{J}}} \frac{\lambda_j x_{jk}}{\mu_k}\right|, \\ s.t. \quad & \forall k \in \boldsymbol{\mathcal{K}}, \frac{\sum\limits_{j\in\boldsymbol{\mathcal{J}}} \lambda_j x_{jk}}{\mu_k} < 1, \\ & \forall j \in \boldsymbol{\mathcal{J}}, \sum_{k\in\boldsymbol{\mathcal{K}}} x_{jk} = 1, \\ & \forall j \in \boldsymbol{\mathcal{J}}, \forall k \in \boldsymbol{\mathcal{K}}, \; x_j \in \{0,1\}, \end{aligned}$$

where the objective is to minimize the sum of the difference between the actual network utilization and the optimal network utilization for each broker in the network. The constraints of $\boldsymbol{P3}$ are equal to Constraint (6)–(8) in $\boldsymbol{P0}$. Note that $\boldsymbol{P3}$ is also an NP-hard problem (which can be proved by applying the same method in Theorem 1). Thus, we design a heuristic algorithm, i.e., Optimal Popular ResourcE re-cAching (OPERA), to solve $\boldsymbol{P3}$. Define g_k as the utility function of broker k, where

$$g_k = \rho_k^* - \sum_{j \in \boldsymbol{\mathcal{J}}} \frac{\lambda_j x_{jk}}{\mu_k}. \tag{18}$$

Note that a broker incurring a larger g_k indicates that the broker is less loaded, and thus can cache more popular resources. The basic idea of OPERA is to iteratively allocate a popular resource to a suitable broker[8], which is defined as the broker that achieves the largest value of the utility function among all the available brokers. A broker is said to be available if the average network resource utilization of the broker is still less than 1 in caching the popular resource, i.e., $\frac{\sum_{j \in \boldsymbol{\mathcal{J}}} \lambda_j x_{jk}}{\mu_k} < 1$ holds after the popular resource is allocated to the broker.

Specifically, as shown in Algorithm 1, we first initialize $\boldsymbol{\mathcal{X}} = \mathbf{0}$ (where $\boldsymbol{\mathcal{X}} = \{x_{jk} | j \in \boldsymbol{\mathcal{J}}, k \in \boldsymbol{\mathcal{K}}\}$) and sort the popular resources in descending order based on their traffic loads[9]. Then, we select the first popular resource in the sorted popular resource set and allocate it into a suitable broker. Denote j^* as the selected popular resource and k^* as the suitable broker to cache popular resource j^*, where

$$k^* = \underset{k}{argmax} \left\{ g_k \left| \sum_{j \in \boldsymbol{\mathcal{J}}} \lambda_j x_{jk} + \lambda_{j^*} < \mu_k, k \in \boldsymbol{\mathcal{K}} \right. \right\}. \tag{19}$$

Here, $\left\{ k \left| \sum_{j \in \boldsymbol{\mathcal{J}}} \lambda_j x_{jk} + \lambda_{j^*} < \mu_k, k \in \boldsymbol{\mathcal{K}} \right. \right\}$ refers to the set of available brokers with respect to popular resource j^*. We allocate popular resource j^* into broker k^*, i.e., $x_{j^*k^*} = 1$. After the allocation, we update the utility function of broker k^* based on Eq. 18. We continue to allocate the next popular resource in the set of sorted popular resource into its suitable broker until all the popular resources are all allocated. Note that the complexity of OPERA is $|\boldsymbol{\mathcal{J}}| \log |\boldsymbol{\mathcal{K}}|$.

C. Summary of LEARN

As mentioned before, the basic idea of LEARN is to sequentially solve the two sub-problems, *optimal network utilization assignment among brokers* and *popular resource re-allocation*, to obtain $\boldsymbol{\mathcal{X}}$. As shown in Algorithm 2, first, the popular resources are identified from all the resources in the network, and the information (i.e., the average resource retrieval request arrival rate and the content size) of each popular resource as well as the service rate of each broker are collected (i.e., Step 1 and Setp 2). Then, the optimal average network resource utilization for each broker is calculated based on Lemma 2 (i.e., Step 3). Afterwards, the optimal resource caching vector $\boldsymbol{\mathcal{X}}$ is obtained by executing the OPERA algorithm (i.e., Step 4). Finally, all the popular resources will be re-allocated based on $\boldsymbol{\mathcal{X}}$ (i.e., Step 5). Note that the complexity of LEARN is determined by the complexity of OPERA.

[8]Allocating a popular resource to a broker means that the broker is enabled to cache the popular resource.

[9]The traffic load of a popular resource (i.e., the value of λ_j ($j \in \boldsymbol{\mathcal{J}}$)) refers to the average resource retrieval request arrival rate related to the popular resource during a time slot.

Algorithm 1 $\boldsymbol{\mathcal{X}} = OPERA(\boldsymbol{\mathcal{L}}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\rho}^*)$.

Input: 1) The content size vector of all the popular resources, i.e., $\boldsymbol{\mathcal{L}} = \{l_j | j \in \boldsymbol{\mathcal{J}}\}$. 2) The average resource retrieval request arrival rate vector for all the popular resources, i.e., $\boldsymbol{\lambda} = \{\lambda_j | j \in \boldsymbol{\mathcal{J}}\}$. 3) The service rate vector for all the brokers, i.e., $\boldsymbol{\mu} = \{\mu_k | k \in \boldsymbol{\mathcal{K}}\}$. 4) The optimal solution of $\boldsymbol{P2}$, i.e., $\boldsymbol{\rho}^* = \{\rho_k^* | k \in \boldsymbol{\mathcal{K}}\}$.

Output: The optimal popular resource caching vector, i.e., $\boldsymbol{\mathcal{X}} = \{x_{jk} | j \in \boldsymbol{\mathcal{J}}, k \in \boldsymbol{\mathcal{K}}\}$.

1: Initialize $\boldsymbol{\mathcal{X}} = \mathbf{0}$.
2: Sort the popular resources in descending order based on their traffic loads.
3: Calculate the utility functions of all the brokers based on Eq. 18.
4: $n = 1$;
5: **while** Not all the popular resources are allocated **do**
6: Select the n^{th} popular resource, denoted as j^*, in the set of sorted popular resources;
7: Find the suitable broker k^* based on Eq. 19;
8: Let $x_{j^*k^*} = 1$;
9: Update the utility function of broker k^*;
10: $n = n + 1$;
11: **end while**
12: **return** $\boldsymbol{\mathcal{X}}$.

Algorithm 2 The LEARN algorithm

1: Find the popular resources based on Eq. 1 to form the set of popular resources $\boldsymbol{\mathcal{J}}$.
2: Obtain the values of $\boldsymbol{\mathcal{L}}$, $\boldsymbol{\lambda}$, and $\boldsymbol{\mu}$.
3: Calculate the value of $\boldsymbol{\rho}^*$ based on Lemma 2.
4: $\boldsymbol{\mathcal{X}} = OPERA(\boldsymbol{\mathcal{L}}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\rho}^*)$.
5: Re-allocate the popular resources among brokers based on the value of $\boldsymbol{\mathcal{X}}$.

VII. Simulation

The network comprises 3×3 Base Stations (BSs), each of which is connected to a broker. The coverage of each BS is 1 km^2. Meanwhile, there are N=3,000 servers in the network and each server hosts one resource. The distribution of the servers may exhibit spatial dynamics, and the location of a server follows a Normal distribution, i.e., $\{z_i^x, z_i^y\} \sim \boldsymbol{N}(1500, 500)$, where z_i^x and z_i^y indicate the location of the server (which hosts resource i ($1 \leq i \leq N$)). As shown in Fig. 8, the server density in BS-5's coverage area is higher than the one in other BSs' coverage areas. Servers within the a BS's coverage area can directly communicate with the BS via various wireless access technologies. The network capacity of each

broker is the same, i.e., $\mu_1 = \mu_2 = \cdots = \mu_{|K|} = 350\ Mbps$. Meanwhile, the content size of each resource is 500 Kb. The average arrival rate for each resource during a time slot is randomly selected between $[0.1\ request/s, 2\ request/s]$, i.e., $\forall i \in \boldsymbol{\mathcal{I}}, \lambda_i = \boldsymbol{U}\ (0.1, 2)$. Moreover, we set the threshold $\eta = 0$ and the content update rate for all the resource to be the same, i.e., $\psi = \psi_1 = \psi_2 = \cdots = \psi_{|\boldsymbol{\mathcal{I}}|} = 0.2$.

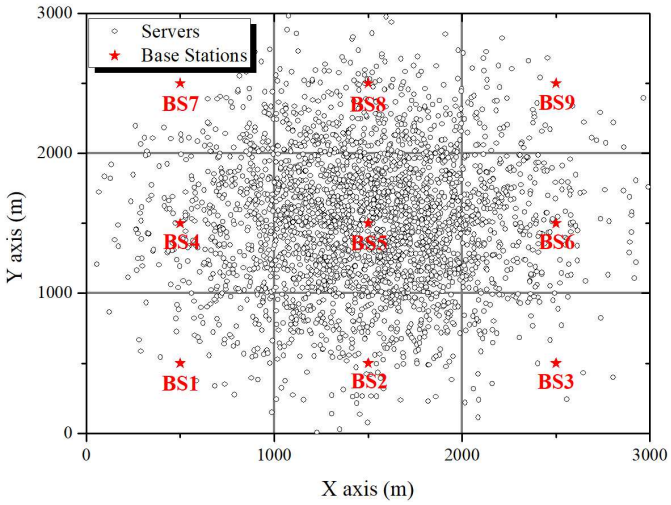


Fig. 8: The network topology.

A. Energy consumption of servers

In this section, we investigate how much energy will be saved in servers for caching popular resources in brokers (by executing LEARN). As comparisons, we consider two other methods, i.e., Non-cache and Always-cache. In the Non-cache method, all the IoT resources are not cached by brokers even if they are popular, and thus servers would respond to the resource retrieval requests by themselves. In the Always-cache method, all the IoT resources are cached by brokers no matter whether they are popular or not. We assume that transmitting one unit of data consumes one unit of energy; Fig. 9 shows the normalized energy consumption of the servers by applying LEARN, Non-cache, and Always-cache[10]. Obviously, LEARN always incurs the lowest energy consumption as compared to the others. Note that as the content update rate of resources (i.e., the value of ψ) increases, fewer resources are considered as popular resources and are suitable to cache in brokers, and thus the energy consumption of the servers incurred by Always-cache increases accordingly. When $\psi = 0.2$, all the resources are non-popular, and thus the energy consumption of the servers incurred by LEARN and Non-cache are the same.

B. Average delay analysis

In this section, we analyze the performance in terms of the average delay of all the brokers in response to their received

[10]The normalized energy consumption of the servers incurred by LEARN/Non-cache/Always-cache equals to the energy consumption of the servers incurred by LEARN/Non-cache/Always-cache divided by the energy consumption of the servers incurred by Always-cache, respectively.

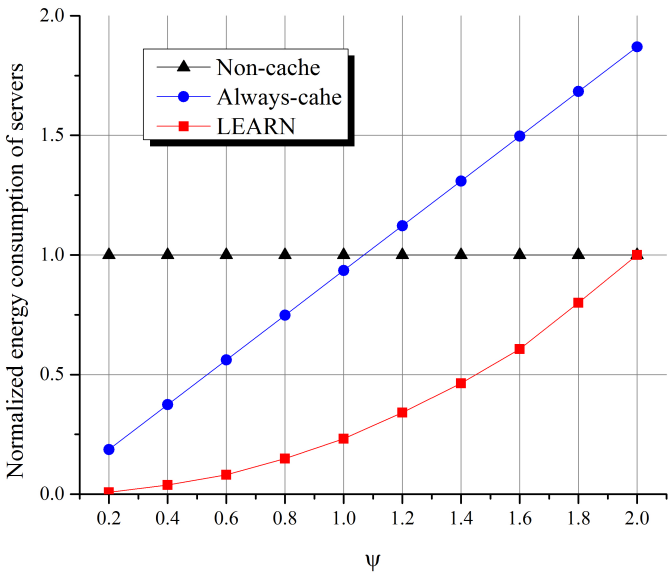


Fig. 9: The normalized energy consumption.

resource retrieval requests by applying LEARN. As comparisons, we also analyze the two other popular resource caching methods, i.e., Nearest Popular Resource Caching (NPRC) and OPTimal load balancing (OPT).

In NPRC, each popular resource is iteratively allocated to the broker, which incurs the shortest RTT among all the available brokers. Note that an available broker is defined as the broker, whose average network resource utilization is still less than 1 if the broker determines to cache the popular resource. In OPT, we relax the binary constraints $x_{jk} \in \{0, 1\}$ in $\boldsymbol{P0}$ to be $x \in [0, 1]$. Thus, solving OPT is equivalent to solving $\boldsymbol{P2}$, which has been proved to have the optimal value, i.e., $\boldsymbol{\rho}^*$. OPT generates the lower bound of $\boldsymbol{P0}$ but does not obtain the feasible solution of $\boldsymbol{P0}$. The purpose of providing the results of OPT is to demonstrate the optimality of LEARN.

We monitor the average delay among all the brokers for 100 time slots. Fig. 10 shows the average delay for a broker (in delivering an IoT content) during the monitoring period. LEARN achieves the similar performance as compared to OPT, which demonstrates that LEARN is a good heuristic algorithm. NPRC incurs much longer average delay as compared to LEARN and OPT because popular resources are cached by their nearest available brokers, thus resulting in unbalanced traffic loads among brokers. In other words, some brokers incur higher average network resource utilization and thus suffer from longer average delay, while other brokers incur lower average network resource utilization and thus incur shorter average delay. In order to demonstrate this conjecture, we further analyze the average network resource utilization and the average delay for each broker during the monitoring period. As shown in Fig. 11, each broker incurs almost the same network resource utilization by applying LEARN and OPT. However, Broker-5 (which is connected to BS-5) incurs higher network resource utilization than other brokers by applying NPRC because the popular resource density in the BS-5's coverage area is higher than other BSs, and thus

Broker-5 caches more popular resources than other brokers by applying NPRC, i.e., Broker-5 may incur higher traffic load than others. Consequently, Broker-5 incurs higher network resource utilization. Note that higher network resource utilization leads to longer average delay based on Eq. 3. Thus, as shown in Fig. 12, Broker-5 incurs significant longer average delay than others by applying NPRC. This is the reason why NPRC incurs much longer average delay among the brokers as compared to LEARN and OPT as shown in Fig. 10.

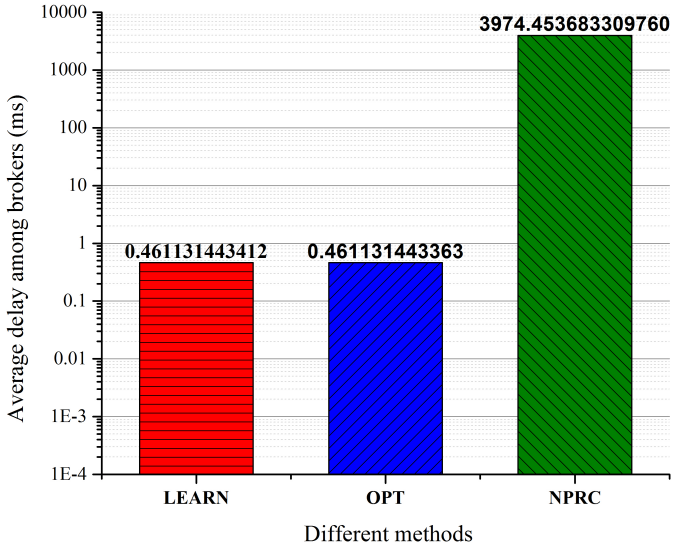


Fig. 10: The average delay among all the brokers during the monitoring period.

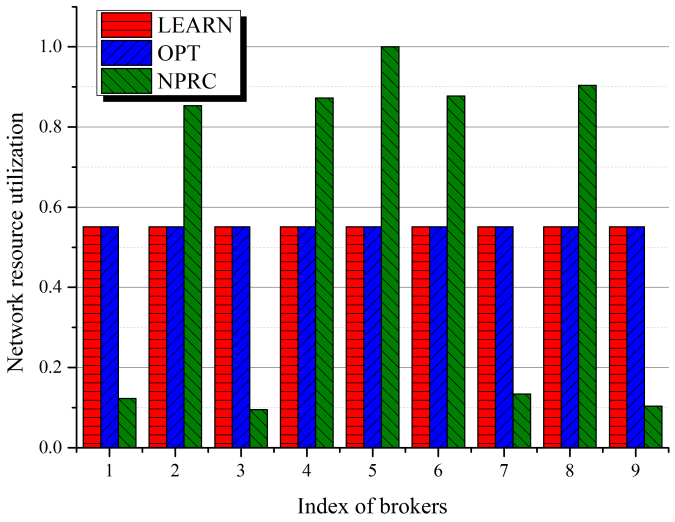


Fig. 11: The average network resource utilization for each broker.

C. The performance impact on the traffic load of the network

We further analyze the performance of LEARN, OPT, and NPRC by varying the number of servers[11] in the network.

[11]Increasing the number of servers is actually increasing the number of the popular resources that are needed to be cached in the brokers.

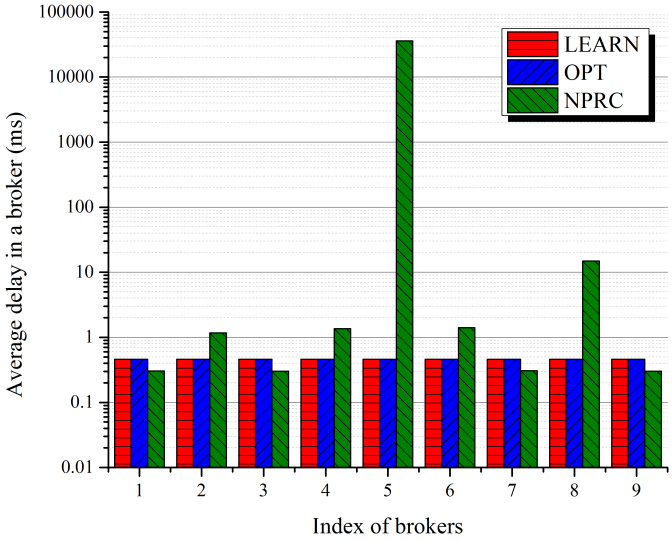


Fig. 12: The average delay for each broker during the monitoring period.

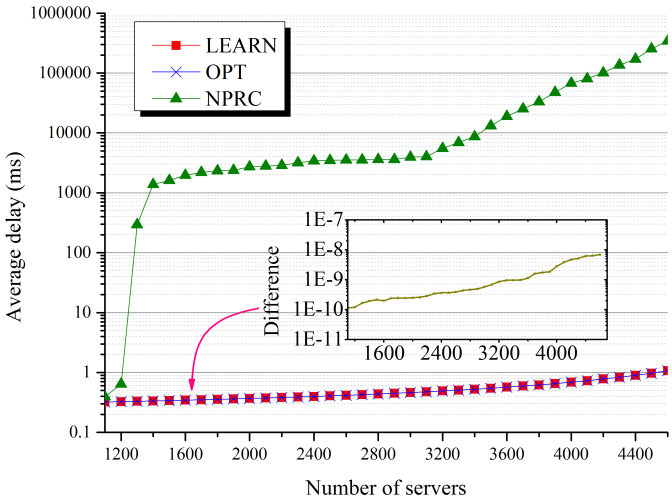


Fig. 13: The average delay among all the brokers by varying the number of servers.

Note that varying the number of servers in the network changes the total traffic load of the network. Fig. 13 shows the average delay among the brokers in delivering their traffic loads during the monitoring period based on different number of servers. LEARN and OPT exhibit the similar average delay, i.e., the difference between LEARN and OPT varies between 1×10^{-10} and 1×10^{-8} as the traffic load of the network changes. The subtly small average delay difference demonstrates that LEARN closely adheres to the lower bound as the traffic load of the network increases. On the other hand, the average delay incurred by NPRC significantly increases as the number of servers increases. Note that the average delay curve of NPRC can be divided into three segments. 1) When $1100 \leq N < 1400$, all the brokers, except Broker-5 (which is over-loaded but still has the residual capacity to cache the popular resources), are lightly loaded, and thus the average delay among the brokers is determined by the

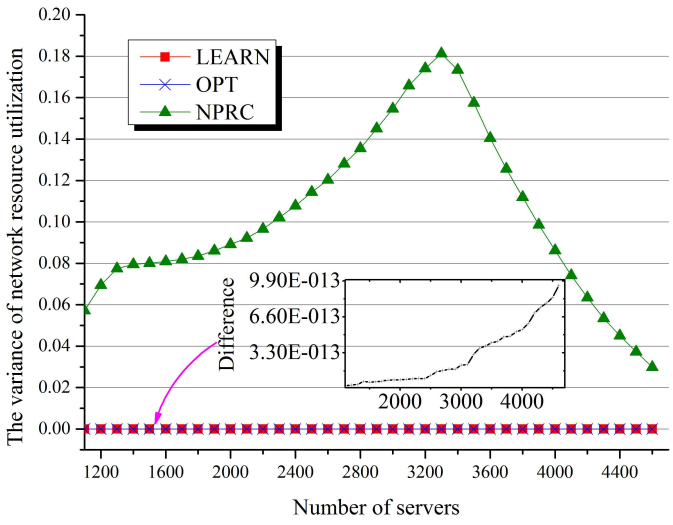


Fig. 14: The variance of the average network resource utilization among brokers.

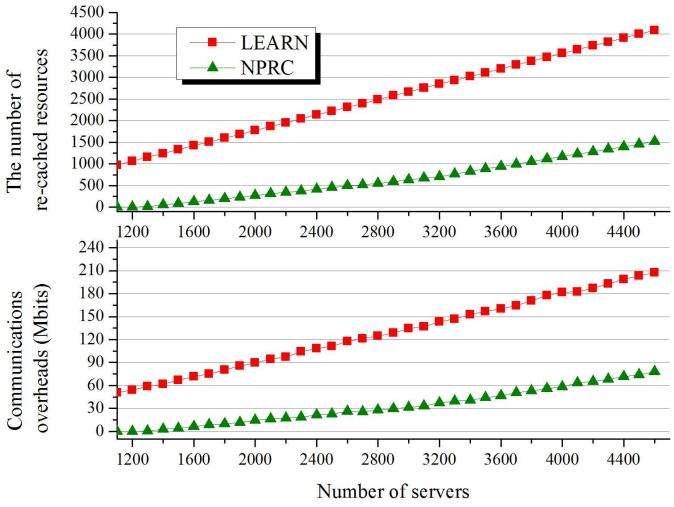


Fig. 15: The communications overheads ($a=130B, b=110B/client, \zeta=200B$).

average delay of Broker-5, whose average delay exponentially increases with the number of servers increases; 2) When $1400 \leq N < 3200$, Broker-5 is congested (i.e., Broker-5 cannot cache any more popular resources), and it would re-cache its local popular resources to its neighboring brokers (i.e., Broker-2, Broker-4, Broker-6, and Broker-8), which are lightly loaded. Consequently, the average delay among all the brokers increases slightly; 3) When $3200 \leq N \leq 4600$, most of the neighboring brokers become over-loaded, and thus their average delay would increase dramatically as their traffic loads increase. Consequently, the average delay among all the brokers also increases dramatically.

Fig. 14 shows the variance of the average network resource utilization among brokers by changing the number of servers in the network. LEARN and OPT (which always incurs zero variance) can always yield the similar variance as the number of servers increases. This indicates that the resource utilization of each broker (by applying LEARN and OPT) keeps nearly the same as the total traffic load in the network increases, thus implying that LEARN and OPT can always evenly distribute the total traffic loads among the brokers based on their network resource capacities. On the other hand, the variance incurred by NPRC is always larger than those incurred by LEARN and OPT, i.e., traffic loads among the broker are unbalanced.

D. Communications overheads consideration

We further analyze the communications overheads incurred by LEARN and NPRC. Note that the communications overheads are generated once a popular resource is re-cached, i.e., the popular resource previously cached by a broker (e.g., Broker-1) is currently re-cached by another broker (e.g., Broker-2). Specifically, as shown in Fig.16,

- Step-1: once the popular resource is determined to be cached by Broker-2, Broker-1 should send a resource creation request to Broker-2. The resource creation request includes the URI of the popular resource hosted by the server as well as the binding table of the popular resource. Note that transmitting the binding table to Broker-2 is to inform Broker-2 about the observe clients of the popular resource such that Broker-2 can continue to transmit the up-to-date contents of the popular resource to those observe clients.
- Step-2: once Broker-2 receives the resource creation request, it would store the binding table, create a new URI, which identifies the popular resource cached in Broker-2, and return this new URI to Broker-1.
- Step-3: Broker-1 needs to inform the server about the new URI (which points to the popular resource in Broker-2) by sending a notification message to the server such that the server can send the up-to-date contents to Broker-2.
- Step-4: after receiving the notification message, the server needs to respond with a confirmation message to Broker-1.
- Step-5: after receiving the new URI, the server should update the URI of the popular resource in the RD by sending a resource update request to the RD such that the read clients can find the new URI, which points to the popular resource in Broker-2.
- Step-6: after receiving the resource update request, the RD needs to send to the server with a resource update response in order to confirm that the URI has been updated.

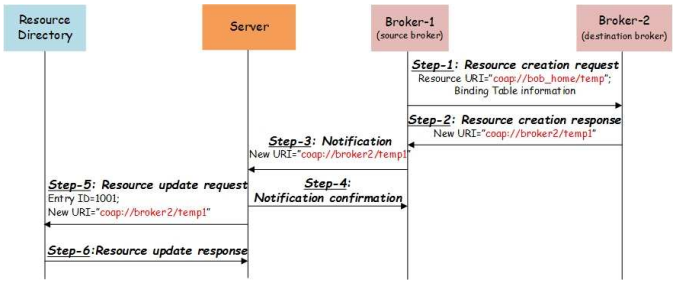


Fig. 16: The illustration of communications overheads incurred by popular resource re-caching.

1) Communications overheads at brokers: Extra messages are generated by the two brokers in Step-1, Step-2, and Step-3. Denote ζ as the amount of communications overheads incurred from Step-2 to Step-3. Note that the value of ζ is the same among different popular resources, i.e., no matter which popular resource is re-cached, the total amount of communications overheads incurred from Step-2 to Step-3 is the same. Denote ω_j as the amount of data incurred by Step-1. The value of ω_j depends on the size of the binding table of resource j, which is further determined by the number of observe clients of resource j, i.e., more observe clients of resource j result in a larger size of the binding table of resource j, thus increasing the value of ω_j. Hence, we have $\omega_j = a + bn_j$, where n_j is the number of the observe clients of resource j, b is the coefficient that maps the number of the observe clients into communications overheads, and a is the offset of communications overheads. Thus, we can derive the total amount of communications overheads by re-caching popular resource j as follows:

$$o_j = a + bn_j + \zeta. \tag{20}$$

Basically, the total amount of communications overheads is determined by the number of re-cached popular resources and the amount of communications overheads incurred in each popular resource re-caching. Assume that the number of observe clients of each popular resource is randomly selected between 0 and 100. Fig. 15 shows the average number of re-cached popular resources and the average amount of communications overheads incurred by LEARN and NPRC. Obviously, LEARN incurs more popular resources that are re-cached and more communications overheads than NPRC during a time slot. In addition, as the number of servers increases, the difference between the communications overheads incurred by LEARN and those incurred by NPRC increases accordingly.

2) Communications overheads at servers: Extra messages are generated by the server (which originally hosts the resource that is re-cached by a different broker) in Step-4 and Step-5. Thus, the amount of communications overheads incurred by the server is determined by the size of the notification confirmation message (in Step-4) and resource update request (in Step-5). Note that the server consumes extra energy in transmitting the two messages when its popular resource is re-cached by a different broker. Thus, there is a tradeoff between reducing the average delay of the brokers (by re-caching their popular resources) and reducing the energy consumption of servers. We will investigate on how to optimize the tradeoff in the future.

VIII. Conclusion

In this paper, we have proposed to cache popular resources in brokers and designed a metric to measure the popularity of a resource. In addition, we have formulated the popular resource re-caching problem to minimize the total average delay among the brokers. We have proved the problem to be NP-hard and designed the LEARN algorithm to solve the problem. We have demonstrated the performance of LEARN via extensive simulations.

References

[1] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, "Security of the Internet of Things: perspectives and challenges," *Wireless Netw.*, vol. 20, no. 8, pp. 2481–2501, 2014.

[2] A. Al-Fuqaha, *et al.*, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," in *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, Fourthquarter 2015.

[3] X. Guo, L. Chu and X. Sun, "Accurate Localization of Multiple Sources Using Semidefinite Programming Based on Incomplete Range Matrix," in *IEEE Sensors J.*, vol. 16, no. 13, pp. 5319-5324, July1, 2016.

[4] S. Vural, N. Wang, P. Navaratnam, and R. Tafazolli, "Caching Transient Data in Internet Content Routers," in *IEEE/ACM Trans. Netw.*, doi: 10.1109/TNET.2016.2616359.

[5] S. Vural, *et al.*, "In-network caching of internet-of-things data," in *2014 IEEE Int. Conf. Commun. (ICC)*, Sydney, NSW, 2014, pp. 3185–3190.

[6] M. A. Hail, M. Amadeo, A. Molinaro, and S. Fischer, "Caching in Named Data Networking for the wireless Internet of Things," in *2015 Int. Conf. Recent Advances in Internet of Things (RIoT)*, Singapore, 2015, pp. 1–6.

[7] E. Baccelli, *et al.*, "Information centric networking in the IoT: Experiments with NDN in the wild," in *Proc. 1st Int. Conf. Information-centric Networking*, Paris, France, Sep. 24–26, 2014, pp. 77–86.

[8] M. Amadeo *et al.*, "Information-centric networking for the internet of things: challenges and opportunities," *IEEE Netw.*, vol. 30, no. 2, pp. 92-100, March-April 2016.

[9] Requirements and Challenges for IoT over ICN. [Online]. Available: https://tools.ietf.org/pdf/draft-zhang-icnrg-icniot-requirements-01.pdf.

[10] A. Lindgren, *et al.*, "Design choices for the IoT in Information-Centric Networks," in *2016 13th IEEE Annual Consumer Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, 2016, pp. 882-888.

[11] A. Rao, O. Schelen, and A. Lindgren, "Performance implications for IoT over information centric networks," in *Proc. 11th ACM Wksh. Challenged Netw.*, New York City, NY, Oct. 03–07, 2016, pp. 57–62.

[12] J. Ni and D. H. K. Tsang, "Large-scale cooperative caching and application-level multicast in multimedia content delivery networks," *IEEE Commun. Mag.*, vol. 43, no. 5, pp. 98-105, May 2005.

[13] M. Diallo, *et al.*, "Leveraging caching for Internet-scale content-based publish/subscribe networks," in *Proc. Int. Conf. Commun. (ICC)*, Jun. 2011, pp. 1-5.

[14] L. Dong, *et al.*, "On the cacheand-forward network architecture," in *Proc. Int. Conf. Commun. (ICC)*, Jun. 2009, pp. 15.

[15] Proposed Design Choices for IoT over Information Centric Networking. [Online]. Available: https://tools.ietf.org/pdf/draft-lindgren-icnrg-designchoices-00.pdf.

[16] X. Sun and N. Ansari, "Dynamic Resource Caching in the IoT Application Layer for Smart Cities," in *IEEE Internet of Things J.*, doi: 10.1109/JIOT.2017.2764418, early access.

[17] N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire, "Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution," in *IEEE Commun. Mag.*, vol. 51, no. 4, pp. 142–149, April 2013.

[18] D. Niyato, D. I. Kim, P. Wang, and L. Song, "A novel caching mechanism for Internet of Things (IoT) sensing service with energy harvesting," in *2016 IEEE Int. Conf. Commun. (ICC)*, Kuala Lumpur, 2016, pp. 1–6.

[19] M. Ha and D. Kim, "On-demand cache placement protocol for content delivery sensor networks," in *2017 Int. Conf. Comput., Netw. Commun. (ICNC)*, Santa Clara, CA, 2017, pp. 207–216.

[20] The Constrained Application Protocol (CoAP). [Online]. Available: https://tools.ietf.org/html/rfc7252.

[21] RESTful Design for Internet of Things Systems. [Online]. Available: https://tools.ietf.org/html/draft-keranen-t2trg-rest-iot-03.

[22] Dynamic Resource Linking for Constrained RESTful Environments. [Online]. Available: https://tools.ietf.org/html/draft-ietf-core-dynlink-01.

[23] CoRE Resource Directory. [Online]. Available: https://tools.ietf.org/html/draft-ietf-core-resource-directory-09.

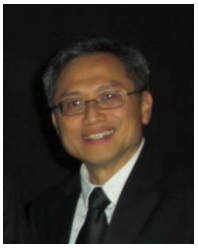
[24] Publish-Subscribe Broker for the Constrained Application Protocol (CoAP). [Online]. Available: https://tools.ietf.org/html/draft-koster-core-coap-pubsub-05.

[25] X. Sun and N. Ansari, "PRIMAL: PRofIt Maximization Avatar pLacement for mobile edge computing," in *2016 IEEE Int. Conf. Commun. (ICC)*, Kuala Lumpur, 2016, pp. 1–6.

[26] X. Sun and N. Ansari, "Latency Aware Workload Offloading in the Cloudlet Network," in *IEEE Commun. Lett.*, vol. 21, no. 7, pp. 1481–1484, July 2017.

[27] X. Sun and N. Ansari, "Avaptive Avatar Handoff in the Cloudlet Network," in *IEEE Trans Cloud Comput.*, doi: 10.1109/TCC.2017.2701794, early access.
[28] X. Sun and N. Ansari, "EdgeIoT: Mobile Edge Computing for the Internet of Things," in *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 22–29, December 2016.
[29] N. Ansari and X. Sun, "Mobile edge computing empowers Internet of Things," *IEICE Trans. Commun.*, doi: 10.1587/transcom.2017NRI0001, early access.
[30] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "Softcell: Scalable and flexible cellular core network architecture," in *Proc. 9th ACM conf. emerging netw. experiments technol.*, Santa Barbara, CA, Dec. 09–12, 2013, pp. 163–174.
[31] X. Sun and N. Ansari, "Green Cloudlet Network: A Distributed Green Mobile Cloud Network," in *IEEE Netw.*, vol. 31, no. 1, pp. 64–70, January/February 2017.
[32] X. Sun, N. Ansari, and Q. Fan, "Green Energy Aware Avatar Migration Strategy in Green Cloudlet Networks," in *2015 IEEE 7th Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Vancouver, BC, 2015, pp. 139–146.
[33] X. Sun and N. Ansari, "Green Cloudlet Network: A Sustainable Platform for Mobile Cloud Computing," in *IEEE Trans. Cloud Comput.*, doi: 10.1109/TCC.2017.2764463, early access.
[34] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A Survey," *IEEE Commun. Surv. Tut.*, vol.16, no.1, 2014, pp. 483-512.
[35] A.A. Puntambekar, "Design and analysis of algorithms," *Technical Publications*, 2009.
[36] S. Boyd and L. Vandenberghe, "Convex optimization," *Cambridge university press*, 2004.

Xiang Sun [S'13] received a B.E. degree in electronic and information engineering and an M.E. degree in technology of computer applications from Hebei University of Engineering, Hebei, China. He is currently working towards the Ph.D. degree in electrical engineering at NJIT, Newark, New Jersey. His research interests include mobile edge computing, big data networking, green computing and communications, content/resource caching in Internet of Things, and drone-aided mobile access networks.



Nirwan Ansari [S'78, M'83 ,SM'94, F'09] is Distinguished Professor of Electrical and Computer Engineering at NJIT. He has also been a visiting (chair) professor at several universities.

He recently authored Green Mobile Networks: A Networking Perspective (Wiley-IEEE, 2017) with T. Han, and co-authored two other books. He has also (co-)authored more than 500 technical publications, over 200 in widely cited journals/magazines. He has guest-edited a number of special issues covering various emerging topics in communications and networking. He has served on the editorial/advisory board of over ten journals. His current research focuses on green communications and networking, cloud computing, and various aspects of broadband networks.

He was elected to serve in the IEEE ComSoc Board of Governors as a member-at-large, has chaired ComSoc technical committees, and has been actively organizing numerous IEEE International Conferences/Symposia/Workshops. He has frequently delivered keynote addresses, distinguished lectures, tutorials, and invited talks. Some of his recognitions include several Excellence in Teaching Awards, some best paper awards, the NCE Excellence in Research Award, the IEEE TCGCC Distinguished Technical Achievement Recognition Award, the ComSoc AHSN TC Technical Recognition Award, the NJ Inventors Hall of Fame Inventor of the Year Award, the Thomas Alva Edison Patent Award, Purdue University Outstanding Electrical and Computer Engineer Award, and designation as a COMSOC Distinguished Lecturer. He has also been granted 35 U.S. patents.

He received a Ph.D. from Purdue University in 1988, an MSEE from the University of Michigan in 1983, and a BSEE (summa cum laude with a perfect GPA) from NJIT in 1982.