Computer Networks 154 (2019) 88–104

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Towards efficient and scalable implementation for coding-based on-demand data broadcast

G. G. Md. Nawaz Ali[a,b], Kai Liu[c,*], Victor C.S. Lee[d], Peter H.J. Chong[e], Yong Liang Guan[b], Jun Chen[f]

[a] Department of Automotive Engineering, Clemson University, SC, USA
[b] School of Electrical & Electronic Engineering, Nanyang Technological University, Singapore
[c] College of Computer Science, Chongqing University, Chongqing, China
[d] Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong
[e] Department of Electrical and Electronic Engineering, Auckland University of Technology, New Zealand
[f] School of Information Management, Wuhan University, China

ARTICLE INFO

Article history:
Received 7 January 2018
Revised 17 August 2018
Accepted 21 February 2019
Available online 27 February 2019

Keywords:
On-demand broadcast
Data scheduling
Network coding
Algorithm design
Performance evaluation

ABSTRACT

Network coding has been demonstrated as a promising solution to further enhancing the bandwidth efficiency for on-demand broadcast. In this work, first, we show the performance improvement of a straightforward implementation of coding based on-demand data broadcast algorithms over the traditional on-demand broadcast approaches. Second, as the straightforward implementation of the optimal approach has overwhelming computational overhead, we propose an efficient generalized implementation scheme, which can be applied to all the existing on-demand scheduling algorithms. The proposed scheme reduces the computational overhead while achieves the same performance as the straightforward implementation. Third, to further enhance system scalability, we propose an approximate implementation method with even lower computational overhead while maintaining near optimal performance. Finally, we conduct an extensive simulation study and the results demonstrate that the proposed efficient implementation schemes can improve the system performance over 40% compared with the traditional broadcast approach, and the computational overhead can be reduced by 75% compared with the straightforward implementation. In addition, we show that the proposed approximate implementation can further reduce the computational overhead significantly and it is able to strike a balance between the service performance and system scalability.

© 2019 Published by Elsevier B.V.

1. Introduction

Data broadcast is an attractive solution for large scale data dissemination in wireless communication environment. In general, there are two broadcast approaches: push-based broadcast [30] and pull-based broadcast [19]. In push-based broadcast, the scheduler periodically disseminates data items based on historical data access patterns of clients. This approach is suitable for applications with small database sizes and stable data access patterns. On the other hand, in pull-based broadcast, which is commonly known as the on-demand broadcast, the scheduler disseminates data items based on explicit requests received from clients. In other words, the scheduling decisions are made on-line based on current pending requests. Obviously, on-demand broadcast is more suitable for dynamic and large-scale data dissemination. Recent research shows that network coding is an effective approach to improving the on-demand broadcast performance [4,10,15,31]. However, it may increase the computational overhead significantly. In this work, we aim at proposing a framework to reduce the computational overhead for implementing the coding version of all the existing on-demand scheduling algorithms while maintaining their performance.

A great number of on-demand scheduling algorithms have been proposed in previous studies [3,17,20,21,37], which target at satisfying different quality of service (QoS) requirements. Based on the QoS requirement of different applications, the existing on-demand scheduling algorithms can be generally classified into three categories: real-time, non-real-time and stretch optimal scheduling. First, in real-time scheduling, the on-demand requests should be served within the stipulated deadlines. In this category, the com-

* Corresponding author.
E-mail addresses: gga@clemson.edu (G. G. Md. Nawaz Ali), liukai0807@gmail.com (K. Liu), csvlee@cityu.edu.hk (V.C.S. Lee), peter.chong@aut.ac.nz (P.H.J. Chong), eylguan@ntu.edu.sg (Y.L. Guan), christina_cj@whu.edu.cn, christina.cj@hotmail.com (J. Chen).

https://doi.org/10.1016/j.comnet.2019.02.012
1389-1286/© 2019 Published by Elsevier B.V.

monly adopted QoS requirement is the deadline miss ratio (DMR), which is the ratio of number of on-demand requests miss their deadlines to the total number of requests submitted to the system. The well-known real-time on-demand scheduling algorithms include EDF [38] and SIN [37], etc. Secondly, in non-real-time scheduling, the commonly adopted QoS requirement is the system responsiveness, which indicates average response time of the system to the on-demand requests. The well-known non-real-time on-demand scheduling algorithms include FCFS [35], MRF [34] and $R \times W$ [3], etc. Thirdly, in stretch-optimal scheduling, the commonly adopted QoS requirement is the metric called stretch, which is the another measurement of the system response time with consideration that different data sizes may require different service time. So, stretch is defined as the ratio of the response time to the service time [1]. The well-known stretch-optimal on-demand scheduling algorithms include LTSF [1] and STOBS [27], etc.

Although existing on-demand scheduling algorithms aim to optimize data scheduling with different QoS requirements, they do not fully utilize the advantage of on-demand broadcast where the server is able to be aware of the requested and cached data items of clients. Considering a broadcast tick as the duration for transmitting one unit-sized data item, all these strategies can only broadcast one data item in a broadcast tick. This restriction hinders the efficiency of wireless bandwidth. In contrast, network coding enables the broadcast of multiple data items via an encoded form in a broadcast tick, and it may further improve bandwidth efficiency by incorporating the cached content in clients into consideration.

Network coding has aroused significant interest in the community of wireless communication, and it has shown great potential on improving bandwidth efficiency in on-demand broadcast systems [6,10,12,15,41]. Different from previous studies, we propose a generalized structure, by which any existing on-demand scheduling algorithm can migrate to its corresponding coding version. Specifically, the framework is designed to best exploit the benefit of applying network coding in on-demand scheduling with low computational overhead, and maintain the original QoS objectives of different on-demand scheduling algorithms. The main contributions of this work are outlined as follows.

- We analyze the performance improvement and the cost for the straightforward implementation of network coding based on-demand scheduling algorithms. Specifically, we present a scheduling framework for the purpose of transforming the representative scheduling algorithms to their respective network coding implementations while keeping their original scheduling criteria.
- We propose a generalized network coding implementation scheme, which is applicable to all the existing on-demand scheduling algorithms. Specifically, we design an efficient data structure and a pruning technique for the purpose of reducing the computational overhead in scheduling while achieving the same performance as the algorithms with straightforward implementation.
- To further enhance the system scalability, we propose an approximate implementation scheme, which can achieve similar performance with the straightforward implemented algorithms, but it significantly reduces the scheduling overhead. Specifically, a tuning parameter is incorporated in approximate implementation, which can be adopted to strike a balance between the system performance and the computational cost.

To support above claims, we build a simulation model and perform a comprehensive evaluation. The results verify the effectiveness of applying network coding into on-demand scheduling, as well as demonstrate the superiority of our proposed generalized framework and approximate implementation scheme.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the scheduling framework. Section 4 describes the straightforward network coding implementation and analyzes its complexity. Section 5 proposes an efficient coding implementation scheme as well as an approximate implementation scheme. Section 6 builds the simulation model and gives the performance evaluation. Finally, Section 7 concludes this paper.

2. Related works

2.1. On-demand scheduling

A great number of works have been focused on the on-demand scheduling algorithms for real-time applications. EDF (Earliest Deadline First) [38] is one of the foremost classical scheduling algorithms in real-time systems. It broadcasts the data item with the shortest remaining lifetime to cater for the urgency of requests. Hu [14] considered request urgency, service productivity and scheduling fairness in algorithm designing. The objective is to achieve low request deadline miss rate and low mean access time simultaneously. Chen et al. [8] gave a theoretical analysis on the minimum required number of channels to complete time-critical services, given a certain time-bound. An algorithm called SUSC (Scheduling under Sufficient Channels) is proposed to generate the broadcast program when there are sufficient channels. Alternatively, an algorithm called PAMAD (the Progressively Approaching Minimum Average Delay) is proposed to handle the situation where the number of channels is insufficient. Chen et al. [9] focused on scheduling for multi-item requests with time constraints in on-demand broadcast environments. A algorithm called DTIU (Dynamic Temperature Inverse Urgency) is proposed, which makes scheduling decisions by considering both request popularity and request timing requirements. Xu et al. [37] proposed a scheduling algorithm for time-critical requests called SIN (Slack time Inverse Number of pending requests). It is motivated by two existing strategies: EDF, which considers the urgency of requests, and MRF, which involves the productivity of data broadcast. SIN outperforms other existing real-time on-demand broadcast algorithms significantly. Lv et al. [23] proposed a profit-based on-demand real-time scheduling in both single- and multi-channel wireless broadcast environments. Liu and Su [17] proposed two heuristic data scheduling approaches for real-time multi-item requests. The objectives are to reducing deadline miss ratio and data access latency. Ali et al. [24] studied the real-time on-demand scheduling in time-constrained applications.

Meanwhile, there have been extensive studies on-demand scheduling algorithms for non-real-time systems. FCFS (First Come First Served) [35] broadcasts data items sequentially according to their arrival order. SRPT (Shortest Remaining Processing Time) [1] chooses the data item with the shortest remaining service time to broadcast. Wong [34] proposed two well-known strategies for scheduling requests in on-demand broadcast systems: MRF (Most Request First) and LWF (Longest Wait First). MRF broadcasts the data item that has the largest number of pending requests to account for broadcast productivity. LWF calculates the total time that all pending requests for a data item have been waiting. The data item with the longest total waiting time is chosen to broadcast. Aksoy and Franklin [3] proposed a low-overhead and scalable scheduling algorithm called RXW (Number of Pending Requests Multiply Waiting Time) that combines the benefits of MRF and FCFS. It calculates the number of pending requests for a data item multiplied by the amount of time that the oldest outstanding request for that data item has been waiting. At each broadcast tick,

the request with the maximum RXW value will be chosen to serve. Liu et al. [19] studied the non-real-time scheduling algorithm for multiple dependent data items. Lu et al. [22] proposed a two-stage scheduling algorithm for reducing access latency for multi-item requests. Liu et al. [21] considered an on-demand broadcast architecture in vehicular networks, where the roadside unit is responsible for broadcasting data items to passing vehicles based on their explicit requests. Further, the peer-to-peer communication paradigm is also exploit in this work to enhance system performance.

Some studies have been focused on stretch optimal scheduling, where the variable sizes of data items is taken into consideration for measuring scheduling performance. A metric called *stretch* is proposed in [1] to qualitatively measure the scheduling performance in such a paradigm. An algorithm called LTSF (Longest Total Stretch First) is proposed in [1] to broadcast the data item with the largest current stretch, where the current stretch of a data item is the ratio of the aggregated waiting time of all the outstanding requests for the data item to the service time of the data item. Sharaf and Chrysanthis [27] proposed a algorithm called STOBS (Summary Tables On-demand Broadcast Scheduler), which broadcasts the data item with the maximum $\frac{R\times W}{S}$ value, where *R, W* and *S* represent the number of pending requests for the data item, the waiting time of the oldest request and the data item size, respectively. Wu and Lee [36] studied the data broadcast performance performance of diverse size data item with deadlines. Ali et al. [5] studied the stretch performance with diverse data items in vehicular networks.

2.2. Network coding assisted broadcast scheduling

Network coding is initially proposed for multicast routing by Ahlswede et al. [2]. Birk and Kol [6] firstly applied network coding into on-demand broadcast systems. Chu et al. [12] proposed an OE (On-demand Encoding) mechanism to reduce the response time in on-demand broadcast. However, it can only encode at most two data items into a packet. Zhan et al. [41] transformed the coding based scheduling problem into the maximum clique problem, and proposed a greedy method to approximately solve the problem. Chen et al. [10] considered the high computational overhead of coding based scheduling algorithms and proposed an improved approach to reducing the scheduling overhead while maintaining system performance. Ji et al. [15] studied the coding assisted on-demand data broadcast with cooperative caching. They formulated the Maximum Channel Efficiency Encoding (MCEE) problem by introducing network coding and cooperative caching technique for enabling cooperation among the non-neighbor mobile clients. De-Nian and Ming-Syan [39] analyzed the coding problem on the push-based broadcast. They transformed the coding problem into an optimization problem and proved that the coding problem is NP-hard.

Liu et al. [18] proposed a coding-assisted broadcast scheduling (CBS) for maximizing the broadcast efficiency in the limited broadcast bandwidth situation. For solving the CBS problem, they employed memetic computing, which is a nature inspired computational model for dealing with complex problems. Wang and Yin [32] proposed a distributed relay selection scheme with network coding in wireless lossy communication channel. To combat the wireless lossy channel property and mobility of clients, the proposed scheme jointly considers geographical locations, channel condition, mobility and packet receiving statuses of clients. Wang et al. [31] proposed a network coding-based scheduling algorithm for reducing the deadline misses of real-time multi-item requests. To minimize the deadline miss ratio, the proposed algorithm exploits the coding opportunities between cached and requested data items and integrates network coding with data scheduling. Yu et al. [40] proposed a limited feedback assisted network coded broadcast in lossy wireless medium. In the proposed approach, the packets are grouped into generations. In each generation, a two-phase coding scheme is proposed. Wang et al. [33] proposed a dynamic transmission rate with network coding for improving the service quality in time critical applications. To strike the balance between transmission rate and transmission delay, they proposed a joint Rate Selection and Network Coding (RSNC) scheme. Sagduyu et al. [25] studied the relay-assisted wireless broadcast with network coding for minimizing the number of transmissions and improving the network throughput. The studied approach jointly considers network coding and packet scheduling. Sui et al. [29] proposed a deadline-aware Cooperative Data Exchange (CDE) scheme with network coding to maximize the total received packets at the clients. They proved that the problem is NP-hard and proposed a heuristic solution to solve the general problem with an auxiliary graph model. Zhan and Xiao [42] investigated a memory-aware network coding based scheduling problem in real-time applications and proposed a novel solution to minimize the number of transmissions. The model assumed that each receiver has enough memory to buffer all encoded packets.

3. Scheduling framework

3.1. System architecture

The system architecture shown in Fig. 1 represents a typical on-demand data broadcast system in wireless communication environments [3,19]. The system consists of one server and a number of clients. When a data item is requested by a client but not found in the local cache, the client submits a request via the uplink channel to the server, and monitors the downlink channel waiting for the requested data item. Typically, the bandwidth of the downlink channel is much greater than that of the uplink channel, which exhibits an asymmetric wireless communication environment. According to a certain scheduling algorithm, the server retrieves the requested data items from the local database and constructs an encoded packet and then broadcasts through the downlink channel. The client gets the requested data item by decoding the received packet, and the decoded data iitem will be cached locally. A client can decode a requested data item from an encoded packet if it caches all the other data items in the encoded packet except the requested one. The XOR operation is commonly adopted for encoding and decoding due to its trivial computation overhead [11,12,28]. For example, as shown in Fig. 1, C3 (i.e., Client 3) has cached d_1. Suppose C3 requests d_2 and the server broadcasts an encoded packet $d_1 \oplus d_2$. C3 is able to decode $d_1 \oplus d_2$ and retrieve d_2 by such a schedule. Due to the limited cache size of clients, the LRU (Least Recently Used) cache replacement policy is adopted for cache management. We assume a closed system model [5,11], where a client submits a new request only while the previous submitted request is satisfied or misses its deadline.

3.2. Graph model

In this section, we discuss the idea of how the on-demand scheduling algorithms migrate to the corresponding network coding versions using the graph model. We construct a CR (Cached-Requested)-graph [7,11] based on the clients' cached data items and requested data items. A CR graph could be used to make encoding decisions for satisfying maximum number of clients by each broadcast unit and eliminating redundant encoding. To construct the graph *G*, the servers needs the full knowledge of clients submitted requests and their corresponding cached data items. Similar to Ali and co-workers [4,11,15], we assume that a client piggybacks its updated cache index information with its on-demand request. Hence the server has the full knowledge of the cached and requested data item relationships without extra cost,

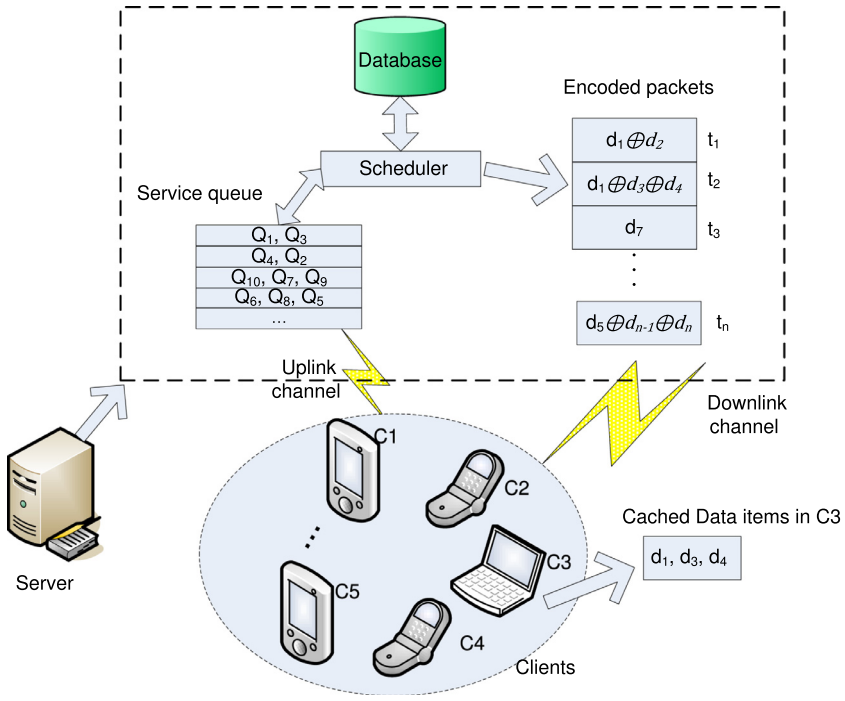


Fig. 1. System architecture.

Table 1
Summary of notations.

Notation	Description
c_i	Client i
d_j	Data item j
$W(c_i)$	Set of requested data items by c_i
$H(c_i)$	Set of cached data items by c_i
v_{ij}	Client c_i requests data item d_j
$t_{v_{ij}}$	Waiting time of v_{ij}
$l_{v_{ij}}$	Data item size of corresponding vertex v_{ij}
$T_{v_{ij}}$	Slack time of v_{ij}
δ	A clique
C_δ	Set of clients covered in δ
D_δ	Set of requested data items in δ
$\delta_{\max}$	Maximum clique
$\delta_{\max}^{v_{ij}}$	Maximal clique which covers vertex v_{ij}
γ	An encoded packet
$Dg_{v_{ij}}$	Degree of vertex v_{ij}
$\delta_{Dg_{v_{ij}}}$	Set of vertices connected to v_{ij}
$t_{Dg_{v_{ij}}}^{avg}$	Average waiting time of a vertex in $\delta_{Dg_{v_{ij}}}$
M	Set of QoS metrics
$M-r$	Set of QoS metrics excluding the data productivity
P	Priority of a vertex
P_{M-r}	Priority of a vertex without considering data productivity

from which it can make the encoding decision. A brief overview of graph construction is given below.

The set of clients in the system is represented by $C = \{c_1, c_2, \cdots, c_n\}$, where n is total number of clients. $W(c_i)$ represents the set of data items requested by client c_i, and $H(c_i)$ represents the set of data items cached at client c_i $(1 \le i \le n)$. The server maintains a database D with m data items, where d_j is the jth data item $(1 \le j \le m)$. The primary used notations are summarized in Table 1.

Definition 1. Given $C = \{c_1, c_2, \cdots, c_n\}$, $D = \{d_1, d_2, \cdots, d_m\}$, $W(c_i) \subseteq D$, $H(c_i) \subseteq D$, $W(c_i) \cap H(c_i) = \emptyset$, the graph $G(V, E)$ is constructed by the following rules:

$$V = \{v_{ij} |\ client\ c_i\ requests\ data\ item\ d_j,\ 1 \le i \le n,\ 1 \le j \le m\}$$
$$E = \{(v_{i_1 j_1}, v_{i_2 j_2}) |\ j_1 = j_2;\ or\ j_1 \ne j_2,\ d_{j_2} \in H(c_{i_1}),\ d_{j_1} \in H(c_{i_2})\}$$

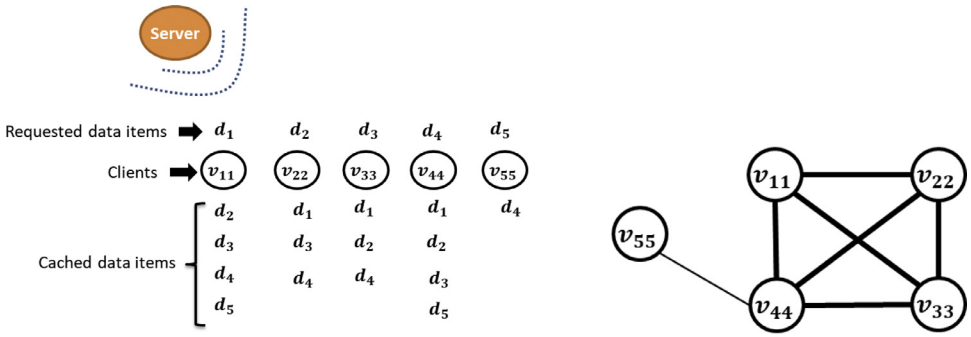
In the CR-graph $G(V, E)$, each vertex represents a data item requested by a client. $v_{ij} \in V(G)$ represents that a client c_i requests data item d_j, where $1 \le i \le n$ and $1 \le j \le m$. For any two distinct vertices $v_{i_1 j_1} \in V(G)$ and $v_{i_2 j_2} \in V(G)$ there will be an undirected edge $(v_{i_1 j_1}, v_{i_2 j_2}) \in E(G)$ according to the following rules:

- $(v_{i_1 j_1}, v_{i_2 j_2}) \in E(G)$ with $j_1 = j_2$ means that if client c_{i_1} and client c_{i_2} request the same data item, there is an edge between the two vertices $v_{i_1 j_1}$ and $v_{i_2 j_2}$.
- $(v_{i_1 j_1}, v_{i_2 j_2}) \in E(G)$ with $j_1 \ne j_2$, $d_{j_2} \in H(c_{i_1})$, and $d_{j_1} \in H(c_{i_2})$ means that if client c_{i_1}'s cache contains the data item being requested by client c_{i_2} and vice versa, there is an edge between vertices $v_{i_1 j_1}$ and $v_{i_2 j_2}$.

According to the construction rule of CR-graph, for an edge $(v_{i_1 j_1}, v_{i_2 j_2}) \in E(G)$, if $j_1 = j_2$, the server broadcasts only d_{j_1}. If $j_1 \ne j_2$, the server broadcasts $d_{j_1} \oplus d_{j_2}$, and clients decode their corresponding required data items.

A clique is a subset of the vertices in an undirected graph, such that every two vertices in the subset are connected by an edge. We assume that $\delta = \{v_{i_1 j_1}, v_{i_2 j_2}, \cdots, v_{i_k j_k}\}$ be an arbitrary clique in G, where $\delta \in V(G)$. Let $C_\delta = \{c_i | v_{ij} \in \delta\}$ be the set of clients covered in δ, where $C_\delta \subseteq C$. Let $D_\delta = \{d_{\delta(j)} | v_{ij} \in \delta\}$ be the set of requested data items in δ, which can be represented as $D_\delta = \{d_{\delta(1)}, d_{\delta(2)}, \cdots, d_{\delta(|D_\delta|)}\}$. Based on the above CR-graph construction rules, from the clients' cached and requested data item relationships as shown in Fig. 2(a), an example CR-graph is constructed in Fig. 2(b). The vertices connected with the black thick edges constitute the maximum clique, namely, $\delta_{\max} = \{v_{11}, v_{22}, v_{33}, v_{44}\}$, whereas, $D_{\delta_{\max}} = \{d_1, d_2, d_3, d_4\}$.

Let $\gamma = d_{\delta(1)} \oplus d_{\delta(2)} \oplus \cdots \oplus d_{\delta(|D_\delta|)}$ be an encoded packet. Recalling that the proposed system is a closed system, hence, each vertex represents a unique client. Again, from the properties of clique, for any client $c_i \in C_\delta$, it can receive the requested data item d_j from the encoded packet γ if its corresponding vertex $v_{ij} \in \delta$. Hence, by broadcasting one encoded packet all clients in a clique



(a) Cached and requested data items. (b) CR-graph.

Fig. 2. An example of CR-graph.

Table 2
Priority calculation of straightforward implementation.

Algorithm	Priority, $P_{v_{ij}}$	Remarks
MRF_N	$\lvert\delta_{\max}^{v_{ij}}\rvert$	Find the highest $P_{v_{ij}}$ among $\forall v_{ij} \in V(G)$; $1 \le i \le n$; $1 \le j \le m$; $1 \le p \le \lvert C_{\delta_{\max}^{v_{ij}}}\rvert$; $1 \le k \le \lvert D_{\delta_{\max}^{v_{ij}}}\rvert$;
FCFS_N	$t_{v_{ij}}$	
EDF_N	$\frac{1}{T_{v_{ij}}}$	
LWF_N	$\sum_{v_{pk} \in \delta_{\max}^{v_{ij}}} t_{v_{pk}}$	
RXW_N	$\lvert\delta_{\max}^{v_{ij}}\rvert \times t_{v_{ij}}$	
SIN_N	$\lvert\delta_{\max}^{v_{ij}}\rvert \times \frac{1}{T_{v_{ij}}}$	
LTSF_N	$\frac{\sum_{v_{pk} \in \delta_{\max}^{v_{ij}}} t_{v_{pk}}}{max_{v_{pk} \in \delta_{\max}^{v_{ij}}} \{\frac{l_{v_{pk}}}{B}\}}$	
STOBS_N	$\frac{\lvert\delta_{\max}^{v_{ij}}\rvert \times t_{v_{ij}}}{max_{v_{pk} \in \delta_{\max}^{v_{ij}}} l_{v_{pk}}}$	

$\delta \subseteq V(G)$ will be satisfied. That is, the number of satisfied clients is equal to the number of vertices in the clique (clique size). Hence the problem of satisfying the maximum number of clients by an encoded packet in a broadcast unit can be transformed into the problem of finding the maximum clique, $\delta_{\max}$ in the CR-graph G. Note that a maximum clique is a clique with the largest number of vertices in G. On the other hand, a maximal clique is a clique that cannot be extended by including one more adjacent vertex in G. A maximum clique is always maximal but the converse does not hold. Finding the maximum clique in a graph is NP-hard and many efficient approximation algorithms have been proposed in literature. We adopt the approach presented in [13] for finding the approximate maximum clique through finding the maximal cliques in polynomial time.

4. A straightforward implementation

4.1. Network coding implementation of the existing scheduling algorithms

Different scheduling algorithms have defined different QoS (quality of service) metrics which assign different scheduling priorities to pending requests. Assume that the priority of a request is a function of certain number of combined QoS metrics. Common QoS metrics considered in the literature for on-demand broadcast include the waiting time of a request, the slack time of a request, the size of requested data item, and the data productivity, which are denoted by t, T, l, r, respectively. So, the priority of a request P can be expressed in terms of a function of one or a number of these QoS metrics, i.e., priority $P = f(t, T, l, r)$. Note that this notation can be generalized to include other QoS metrics. A vertex v_{ij} (which corresponds to a client c_i asking for data item d_j) is associated with four attributes, namely $t_{v_{ij}}$, $T_{v_{ij}}$, $l_{v_{ij}}$, $r_{v_{ij}}$, which represent the waiting time, the slack time, the data item size and the productivity corresponding to client c_i, respectively.

Definition 2. Let $P_{v_{ij}}$ be the priority of a vertex v_{ij}. Then, $P_{v_{ij}}$ can be expressed by a function of QoS metrics of a scheduling algorithm, namely, $P_{v_{ij}}$ is calculated according to the QoS requirement of a scheduling algorithm.

For instance, for MRF_N, $P_{v_{ij}} \leftarrow$ the size of a maximal clique covering v_{ij} ($\lvert\delta_{\max}^{v_{ij}}\rvert$). For R × W, $P_{v_{ij}} \leftarrow \lvert\delta_{\max}^{v_{ij}}\rvert \times t_{v_{ij}}$. Note that $\delta_{\max}^{v_{ij}}$ is a maximal clique in the CR-graph G which covers the vertex v_{ij}. The identified $\delta_{\max}^{v_{ij}}$ with the highest priority $P_{v_{ij}}$ is used to generate the encoded packet γ.

Based on the graph model presented in Section 3.2, and the corresponding priority calculation requirement (Definition 2), we examine eight representative scheduling algorithms, and describe their priority calculation in the straightforward implementation. The summary is shown is shown in Table 2. The corresponding discussions are stated below:

- MRF [34]: MRF considers the data item productivity as QoS metric. MRF broadcasts the data item with the highest productivity. The network coding implementation of MRF, denoted by MRF_N, needs to find the maximal clique which includes the maximum number of vertices, namely the maximum clique $\delta_{\max}$. Hence, MRF_N examines $P_{v_{ij}}$ of $\forall v_{ij} \in V(G)$ to find the highest priority $P_{v_{ij}}$, where $P_{v_{ij}} \leftarrow \lvert\delta_{\max}^{v_{ij}}\rvert$.
- FCFS [35]: FCFS considers the waiting time of request as QoS metric. FCFS serves the pending requests with the longest waiting time. The network coding implementation of FCFS, denoted by FCFS_N needs to find the maximal clique $\delta_{\max}^{v_{ij}}$ of the candidate vertex v_{ij} where v_{ij} is the vertex with the longest waiting time ($t_{v_{ij}}$) in G. Hence, FCFS_N examines $P_{v_{ij}}$ of $\forall v_{ij} \in V(G)$ to find the highest priority $P_{v_{ij}}$, where $P_{v_{ij}} \leftarrow t_{v_{ij}}$.
- EDF [38]: EDF considers the urgency of request as QoS metric. EDF serves the request with the shortest slack time, where slack time is the remaining deadline of the request. The network coding implementation of EDF, denoted by EDF_N needs to find the maximal clique $\delta_{\max}^{v_{ij}}$ of the candidate vertex v_{ij} where v_{ij} is with the shortest slack time in G. Hence, EDF_N examines $P_{v_{ij}}$ of $\forall v_{ij} \in V(G)$ to find the highest priority $P_{v_{ij}}$, where $P_{v_{ij}} \leftarrow \frac{1}{T_{v_{ij}}}$.
- LWF [34]: LWF considers two QoS metrics, the waiting time of request and the productivity of data item. LWF

broadcasts the data item with the largest summed waiting time of the pending requests for that data item. The network coding implementation of LWF, denoted by LWF_N, needs to find the maximal clique with the maximum summed waiting time of vertices contained by the clique. Hence, LWF_N examines $P_{v_{ij}}$ of $\forall v_{ij} \in V(G)$ to find the highest priority $P_{v_{ij}}$, where $P_{v_{ij}} \leftarrow \sum_{v_{pk} \in \delta_{\max}^{v_{ij}}} t_{v_{pk}}$, $1 \le p \le |C_{\delta_{\max}^{v_{ij}}}|$, $1 \le k \le |D_{\delta_{\max}^{v_{ij}}}|$.

- R × W [3]: R × W considers two QoS metrics, the productivity of data item and the waiting time of request. R × W broadcasts the data item with the maximum R × W value, where R is the number of pending requests for the requested data item and W is the waiting time of the oldest pending request for that data item. The network coding implementation of R × W, denoted by R × W_N needs to find the clique $\delta_{\max}^{v_{ij}}$ with the maximum value of $|\delta_{\max}^{v_{ij}}| \times t_{v_{ij}}$, where $\delta_{\max}^{v_{ij}}$ and $t_{v_{ij}}$ are the maximal clique and the waiting time of vertex v_{ij}, respectively. Hence, R × W_N examines $P_{v_{ij}}$ of $\forall v_{ij} \in V(G)$ to find the highest priority $P_{v_{ij}}$, where $P_{v_{ij}} \leftarrow |\delta_{\max}^{v_{ij}}| \times t_{v_{ij}}$.
- SIN [37]: SIN consider two QoS metrics, the slack time of request and the productivity of data item. SIN broadcasts the item with the minimum SIN value, which is the ratio of the slack time of the most urgent pending request to the number of pending requests for that data item. The network coding implementation of SIN, denoted by SIN_N needs to find the clique $\delta_{\max}^{v_{ij}}$ with the maximum value of $|\delta_{\max}^{v_{ij}}| \times \frac{1}{T_{v_{ij}}}$, where $\delta_{\max}^{v_{ij}}$ is the maximal clique and $T_{v_{ij}}$ is the slack time of vertex v_{ij}, respectively. Hence, SIN_N examines $P_{v_{ij}}$ of $\forall v_{ij} \in V(G)$ to find the highest priority $P_{v_{ij}}$, where $P_{v_{ij}} \leftarrow |\delta_{\max}^{v_{ij}}| \times \frac{1}{T_{v_{ij}}}$.
- LTSF [1]: LTSF considers two QoS metrics, the waiting time of request, the service time of data item. LTSF broadcasts the data item with the largest total current stretch, where stretch is the ratio of the request waiting time to the service time of data item. Service time is the transmission time of a data item. The network coding implementation of LTSF, denoted by LTSF_N, needs to find the maximal clique with the maximum value of $\frac{\sum_{v_{pk} \in \delta_{\max}^{v_{ij}}} t_{v_{pk}}}{\max_{v_{pk} \in \delta_{\max}^{v_{ij}}} \{\frac{l_{v_{pk}}}{B}\}}$, where $\sum_{v_{pk} \in \delta_{\max}^{v_{ij}}} t_{v_{pk}}$ is the summed waiting time of vertices in $\delta_{\max}^{v_{ij}}$ and $\max_{v_{pk} \in \delta_{\max}^{v_{ij}}} \{\frac{l_{v_{pk}}}{B}\}$ is the maximum service time of encoded packet in $\delta_{\max}^{v_{ij}}$. Note that B denotes the transmission channel bandwidth. Hence, LTSF_N examines $P_{v_{ij}}$ of $\forall v_{ij} \in V(G)$ to find the highest priority $P_{v_{ij}}$, where $P_{v_{ij}} \leftarrow \frac{\sum_{v_{pk} \in \delta_{\max}^{v_{ij}}} t_{v_{pk}}}{\max_{v_{pk} \in \delta_{\max}^{v_{ij}}} \{\frac{l_{v_{pk}}}{B}\}}$, $1 \le p \le |C_{\delta_{\max}^{v_{ij}}}|$, $1 \le k \le |D_{\delta_{\max}^{v_{ij}}}|$.
- STOBS [27]: STOBS considers three QoS metrics, the data item productivity, the waiting time of request and the data item size. STOBS broadcasts the data item with the largest $\frac{R \times W}{S}$ value, where R, W and S represent the number of pending requests, the waiting time of the oldest pending request and the data item size, respectively. The network coding implementation of STOBS, denoted by STOBS_N needs to find the clique $\delta_{\max}^{v_{ij}}$ with the maximum value of $\frac{|\delta_{\max}^{v_{ij}}| \times t_{v_{ij}}}{\max_{v_{pk} \in \delta_{\max}^{v_{ij}}} l_{v_{pk}}}$, where $\max_{v_{pk} \in \delta_{\max}^{v_{ij}}} l_{v_{pk}}$ is the maximum data item size of the clique $\delta_{\max}^{v_{ij}}$, which also represents the size of the encoded packet. Hence, STOBS_N examines $P_{v_{ij}}$ of $\forall v_{ij} \in V(G)$ to find the highest priority $P_{v_{ij}}$, where $P_{v_{ij}} \leftarrow \frac{|\delta_{\max}^{v_{ij}}| \times t_{v_{ij}}}{\max_{v_{pk} \in \delta_{\max}^{v_{ij}}} l_{v_{pk}}}$, $1 \le p \le |C_{\delta_{\max}^{v_{ij}}}|$, $1 \le k \le |D_{\delta_{\max}^{v_{ij}}}|$.

The core of the network coding implementation is to find the vertex with the highest priority. The pseudo-code of the straightforward implementation is shown in Algorithm 1.

Algorithm 1: Straightforward network coding implementation.

Step 1: Arrival of a new request Q_i for d_k at the server/* Add the request's corresponding vertex v_{ik} into $V(G)$ in scheduling order */
$V(G) \longleftarrow V(G) + v_{ik}$;
$MAX \leftarrow 0$;
/* Initialize MAX */
for *each* $v_{ij} \in V(G)$ **do**
  $P \leftarrow$ Calculate the priority of vertex v_{ij} (Refer to Table 2);
  if $P > MAX$ **then**
    $MAX \leftarrow P$;
    $\delta_{\max} \leftarrow$ the maximal clique of the vertex with MAX;
Step 2: Generation of encoded packet/* Generate the encoded packet γ according to $\delta_{\max}$ */
Compute $D_{\delta_{\max}}$ of $\delta_{\max}$;
$\gamma = d_{\delta_{max(1)}} \oplus d_{\delta_{max(2)}} \oplus \cdots \oplus d_{\delta_{max(|D_{\delta_{\max}}|)}}$ where $d_{\delta_{max(i)}} \in D_{\delta_{\max}}$;
Broadcast the encoded packet γ;
After the encoded packet γ has been broadcast;
/* Update V(G) according to the vertices in $\delta_{\max}$ */
for *each* $v_{ij} \in \delta_{\max}$ **do**
  $V(G) \leftarrow V(G) - v_{ij}$;

4.2. Complexity analysis of the straightforward implementation

As discussed in Section 4.1, to exploit network coding in on-demand broadcast, a scheduling algorithm from a constructed CR-graph needs to find the maximum weighted clique, or equivalently to find the maximal clique of the vertex corresponds to the highest priority according to QoS requirements. However, the problem of finding the maximum clique in a graph is a NP-complete problem [16]. Hence, we use the approximation algorithm presented in [13], which reduces the time complexity to polynomial. According to Dharwadker [13], the approximate algorithm has complexity of $O(n^5)$ for finding the maximal clique of a given vertex and it has the complexity of $O(n^8)$ for finding an approximate maximum clique $\delta_{\max}$. However, a straightforward network coding implementation may require to search n maximal cliques for finding the maximal clique with the highest priority according to the QoS requirement of a scheduling algorithm (Table 2), where n is the upper bound of number of pending requests.Note that the maximal clique, which covers the vertex with the highest priority will be used to generate the encoded packet. When n is large, the straightforward implementation requires high computational cost, which may become the hurdle of system scalability. To have efficient bandwidth utilization, the time taken for scheduling must be less than the transmission time of a data item. Therefore, it is imperative to focus on the efficient network coding implementation without compromising

the performance. We aim to reduce the computation overhead of the straightforward implementation by proposing an efficient implementation of network coding mechanisms in on-demand broadcast.

5. An efficient implementation

5.1. Efficient coding implementation

With n clients, the straightforward coding implementation requires to construct and search n maximal cliques to find the maximal clique with the highest priority, which may cause high computational cost [7]. If n is large, applying network coding may be impractical in on-demand broadcast where requests have deadlines or the system needs quick response. In this regard, we propose a generalized efficient implementation of network coding which are applicable to all the existing on-demand scheduling algorithms. This not only reduces the searching cost but also achieves the as good performance as the straightforward implementation. From the straightforward network coding implementation, we observe that the major reason of high computation overhead is the straightforward searching for the maximal clique with the highest priority. We reduce this computation overhead by adopting the following four mechanisms.

- First, we consider the vertex degree (Dg) when computing the priority of a vertex. It can be adopted to skip the computation of maximal cliques of some vertices (Section 5.1.1).
- Second, instead of constructing the whole CR-graph G and searching the maximum clique, we construct a sub-graph G' of a particular vertex and search the maximal clique covering that vertex. This reduces the graph construction and maximum clique searching overhead (Section 5.1.2).
- Third, we propose an efficient data structure and a pruning technique for reducing the search space, which are applicable for all the exiting scheduling algorithms (Table 3).
- Fourth, we integrate the above three concepts so that it benefits to all the scheduling algorithms (Section 5.1.4).

5.1.1. Degree of vertex

Definition 3. Let $Dg_{v_{ij}}$ be the degree of vertex v_{ij}, which is the number of vertices connected to v_{ij} in G. Let $\delta_{Dg_{v_{ij}}} = \{v_{i_1 j_1}, v_{i_2 j_2}, \cdots, v_{i_k j_k}\}$ be the set of vertices which are connected to v_{ij}. $\sum_{v_{ij} \in \delta_{Dg_{v_{ij}}}} t_{v_{ij}}$ is the summed waiting time of all vertices connected to v_{ij}. $\frac{\sum_{v_{ij} \in \delta_{Dg_{v_{ij}}}} t_{v_{ij}}}{Dg_{v_{ij}}}$ is the average waiting time of a vertex in $\delta_{Dg_{v_{ij}}}$, which is denoted by $t^{avg}_{Dg_{v_{ij}}}$. Note that $Dg_{v_{ij}}$ is the upper bound of a maximal clique size of G covering v_{ij}.

For the coding implementation, we calculate the priority of vertices for finding the vertex with the highest priority. Recalling that for the productivity based scheduling algorithms (e.g., MRF, SIN, $R \times W$ etc.), the priority calculation of a vertex depends on finding the maximal clique covering the vertex. The straightforward implementation may need to compute all the possible n maximal cliques for n vertices. By sorting the vertices in descending order according to their degrees in a list, the maximal clique calculation of some vertices can be skipped. Note that the degree of vertex is the upper bound of its maximal clique size. Hence, in a descending order shorted list, if the maximal clique size of a vertex is higher than the degree of the following vertex, then there is no need to compute the maximal cliques of all the subsequent following vertices in that list.

5.1.2. Sub-graph

Note that the CR-graph construction may be invoked in each scheduling round, as new requests may arrive or some requests may be served in the service queue. As constructing the whole CR-graph requires to check the relationships of each vertex with all other vertices, the complexity is $O(n^2)$. However, we can reduce this complexity by forming a sub-graph G' for a given vertex, instead of constructing a complete CR-graph G. The sub-graph G' will cover the given vertex and all other connected vertices to the candidate vertex. Note that as the sub-graph construction only requires to check the relationships of a given vertex with the other vertices, so the complexity of sub-graph construction is $O(n)$.

Note that for finding the maximal clique of a given vertex, we only require those vertices connected with that given vertex (degree of vertex). Hence from the constructed sub-graph G', we can find the maximal clique of a given vertex. Hence, we propose a coding strategy AC (adaptive coding), which constructs a sub-graph for a given vertex and finds the maximal clique from the constructed sub-graph. The pseudo-code of AC is shown in lines 50–68 in Algorithm 2. Note that although AC searches the maximal clique in the sub-graph not in the complete CR-graph, AC is still able to find the largest clique for a given vertex. This is because, AC constructs the new sub-graph before searching the maximal clique. That means AC is dynamic and adaptive to the changes of the system environment, i.e., adding a new request or removing a served request.

5.1.3. Efficient data structure and pruning technique

For reducing the computation overhead, we need to reduce the search space. Therefore, we propose a two-list efficient data structure so that it can benefit to all the scheduling algorithms. Recalling that different scheduling algorithms have different QoS metrics. The two-list efficient structure is described as follows. Assume M is a set of QoS metrics required by a scheduling

Table 3
Inputs of D-list and Z-list for efficient implementation.

Algorithms	D-list	Z-list (P_{M-r})	Comments	Description
MRF_N	$Dg_{v_{ij}}$	*NULL*	Algorithms with only D-list has entry. $P \leftarrow \delta_{\max}$	$v_{ij} \in V'(G')$; $1 \le i \le n$; $1 \le j \le m$; $t^{avg}_{Dg_{v_{ij}}} \leftarrow \frac{\sum_{v_{ij} \in \delta_{Dg_{v_{ij}}}} t_{v_{ij}}}{Dg_{v_{ij}}}$
FCFS_N	*NULL*	$t_{v_{ij}}$	Algorithms with only Z-list has entry. $P \leftarrow P_{M-r}$	
EDF _N	*NULL*	$\frac{1}{T_{v_{ij}}}$		
R × W_N	$Dg_{v_{ij}}$	$t_{v_{ij}}$	Algorithms with both D-list and Z-list have entry. $P \leftarrow \delta_{\max} \times P_{M-r}$	
SIN_N	$Dg_{v_{ij}}$	$\frac{1}{T_{v_{ij}}}$		
LWF_N	$Dg_{v_{ij}}$	$t^{avg}_{Dg_{v_{ij}}}$		
STOBS_N	$Dg_{v_{ij}}$	$\frac{t_{v_{ij}}}{T_{v_{ij}}}$		
LTSF_N	$Dg_{v_{ij}}$	$\frac{t^{avg}_{Dg_{v_{ij}}}}{\frac{T_{v_{ij}}}{R}}$		

Algorithm 2 Efficient network coding implementation.

Step 1: Arrival of a new request Q_i for d_k at the server
/* Add the corresponding vertex v_{ik} into $V(G)$ */
$V(G) \longleftarrow V(G) + v_{ik}$;
$M \longleftarrow$ set of QoS metrics (e.g., t, T, l, r);
$M - r \longleftarrow M - \{r\}$;
if $\{M - r\} == M$ **then**
    /* The algorithms do not consider data productivity as QoS metric, e.g., EDF, FCFS etc. */
    Compute P_{M-r} of v_{ik} for QoS metric(s) $M - r$;
else
    /* The algorithms consider data productivity as QoS metric, e.g., MRF, R×W, SIN etc. */
    if $\{M - r\} == NULL$ **then**
        /* The scheduling algorithm considers data productivity as only QoS metric, e.g., MRF */
        Compute $Dg_{v_{ik}}$ of vertex v_{ik};
    else
        /* The algorithms consider data productivity in addition of other QoS metric(s), e.g., R×W */
        Compute P_{M-r} of v_{ik} for QoS metric(s) $M - r$;
        Compute $Dg_{v_{ik}}$ of vertex v_{ik};
if $\{M - r\} == M$ **then**
    Link v_{ik} and its P_{M-r} value in the Z-list in descending order of P_{M-r};
    D-list $\leftarrow NULL$;
else if $\{M - r\} == NULL$ **then**
    Link v_{ik} and its Dg value in the D-list in descending order of Dg value;
    Z-list $\leftarrow NULL$;
else
    Link v_{ik} and its Dg value in the D-list in descending order of Dg value;
    Link v_{ik} and its P_{M-r} value in the Z-list in descending order of P_{M-r};

Step 2: Generation of encoded packet
/* Find the maximal clique that covers the vertex with the highest priority; */
if *D-list* $== NULL$ **then**
    /* e.g. EDF, FCFS etc. */
    $pz \leftarrow$ the head of Z-list;
    Find the maximal clique δ_{max} of the vertex pointed by pz **by invoking AC**;
else if *Z-list* $== NULL$ **then**
    /* e.g. MRF */
    Find the maximal clique δ_{max} covering the vertex with the highest priority **by invoking FindMaxClique_Productivity (D-list)**;
else
    /* R×W, SIN etc. */
    Find the maximal clique δ_{max} covering the vertex with the highest priority **by invoking ADC-2**;
/* Generate the encoded packet γ according to δ_{max} */
Compute $D_{\delta_{max}}$ of δ_{max};
$\gamma = d_{\delta_{max(1)}} \oplus d_{\delta_{max(2)}} \oplus \cdots \oplus d_{\delta_{max(|D_{\delta_{max}}|)}}$ where $d_{\delta_{max(i)}} \in D_{\delta_{max}}$;

Step 3: Broadcast and update
Broadcast the encoded packet γ;
/* After the encoded packet γ has been broadcast, Update V(G) by removing the vertices in δ_{max} */
for *each* $v_{ij} \in \delta_{max}$ **do**
    $V(G) \leftarrow V(G) - v_{ij}$;

Function definition of Adaptive Coding (AC)

AC ($V(G)$, v_{pk})
begin
    /* Input: $V(G)$ and a given vertex v_{pk};
    Output: Maximal clique δ_{max} covering vertex v_{pk};
    */
    Step 1: Construct the undirected sub-graph $G'(V', E')$ for v_{pk} from $V(G)$
    /* initialize $V'(G')$ with v_{pk} */
    $V'(G') \leftarrow V'(G') + v_{pk}$;
    for *each* $v_{ij} \in V(G)$ **do**
        if $j = k$ **then**
            /* both clients request the same data item */
            $V'(G') \leftarrow V'(G') + v_{ij}$;
            $E'(G') \leftarrow E'(G') + e(v_{pk}, v_{ij})$;
        else if $d_k \in H(c_i)$ *and* $d_j \in H(c_p)$ **then**
            /* clients request different data items but cached each other requested data item */
            $V'(G') \leftarrow V'(G') + v_{ij}$;
            $E'(G') \leftarrow E'(G') + e(v_{pk}, v_{ij})$;
    Step 2: Find and return the maximal clique δ_{max} covering v_{pk} in sub-graph G', where $v_{pk} \in \delta_{max}$

Function definition of FindMaxClique_Productivity

```
FindMaxClique_Productivity (D-list)
begin
    /* Input:  D-list;
    Output:  The maximal clique δ_max covering the vertex with the highest priority;
                                                                                    */
    pd ← the head of D-list;
    MAX ← δ_max of the vertex pointed by pd by invoking AC;
    Advance pd to the next unexamined vertex in the D-list;
    while pd ≠ NULL do
        if pd → Dg > MAX then
            P ← δ_max of the vertex pointed by pd by invoking AC;
            if P > MAX then
                MAX ← P;
                δ_max ← the maximal clique of the vertex with MAX;
        else
            /* pd → Dg is equal to or smaller than MAX, so no need further searching   */
            break;
        Advance pd to the next unexamined vertex in the D-list;
    return δ_max;
```

Function definition of Adaptive Demand-oriented Coding-2 (ADC-2)

```
ADC-2 (D-list, Z-list)
begin
    /* Input:  D-list and Z-list;
    Output:  The maximal clique δ_max covering the vertex with the highest priority;
                                                                                    */
    pd ←the head of D-list;
    pz ←the head of Z-list;
    MAX ← 0;
    limit(Dg) ← 0;
    limit(P_M−r) ← 0;
    while pd ≠ NULL||pz ≠ NULL do
        if pd → Dg ≥ limit(Dg) then
            if (pd → Dg × P_M−r) > MAX then
                Invoke AC to find the maximal clique δ_max of the vertex pointed by pd;
                Calculate the priority P of the vertex pointed by pd (Refer to Table 3, P ← δ_max × P_M−r);
                if P > MAX then
                    MAX ← P;
                    δ_max ← the maximal clique of the vertex with MAX;
            Advance pd to the next unexamined vertex in the D-list;
            if pd ≠ NULL then
                limit(P_M−r) ← MAX / (pd → Dg);
        else
            break; /* pd → Dg < limit(Dg)                                           */
        if pz → P_M−r ≥ limit(P_M−r) then
            if (pz → P_M−r × Dg) > MAX then
                Invoke AC to find the maximal clique δ_max of the vertex pointed by pz;
                Calculate the priority P of the vertex pointed by pz (Refer to Table 3, P ← δ_max × P_M−r);
                if P > MAX then
                    MAX ← P;
                    δ_max ← the maximal clique of the vertex with MAX;
            Advance pz to the next unexamined vertex in the Z-list;
            if pz ≠ NULL then
                limit(Dg) ← MAX / (pz → P_M−r);
        else
            break; /* pz → P_M−r < limit(P_M−r)                                     */
    return δ_max;
```

algorithm. $M-r$ denotes the set of QoS metrics excluding the data productivity. P_{M-r} denotes the priority of a vertex without considering the data productivity. P is the priority of a vertex, calculated by $\delta_{\max} \times P_{M-r}$, where $\delta_{\max}$ is the maximal clique covering that vertex. The D-list contains the descending order of degree of vertex value Dg (Definition 3). The Z-list contains the descending order of the priority value P_{M-r}.

When a new request arrives, the corresponding vertex is added into $V(G)$. Based on the required QoS metrics of the scheduling algorithm, M and $M-r$ are calculated.

- If a scheduling algorithm does not consider the data productivity as a QoS metric such as EDF, FCFS etc., M and $M-r$ will be equal.

- If $M - r$ equals *NULL*, then the data item productivity is the only QoS metric of that algorithm such as MRF.
- Otherwise, the algorithm considers the data item productivity in addition to other QoS metrics, such as SIN, LWF, R × W etc.

The entries of the D-list and the Z-list of a particular algorithm depend on the required QoS metrics. The two-list structure follows the following rules.

- If an algorithm considers the QoS metric(s) without the data item productivity, then only the Z-list contains the descending order value of P_{M-r} while the D-list is set to *NULL*.
- If an algorithm considers only the data item productivity as the QoS metric, then the D-list contains the descending order value of *Dg* and the Z-list is set to *NULL*.
- If an algorithm considers the data item productivity in addition of other QoS metrics, the D-list contains the descending order value of *Dg* and the Z-list contains the descending order value of P_{M-r}.

Based on the different QoS requirements, different scheduling algorithms may have different P_{M-r} values. The entries of D-list and Z-list of different algorithms are shown in Table 3. Based on the two-list data structure, we can apply the efficient pruning technique to reduce the search space for finding the maximal clique $\delta_{\max}$ with the highest priority. $\delta_{\max}$ is required for generating the encoded packet γ.

- If the D-list equals *NULL*, the maximal clique for the vertex pointed by the head of Z-list is the maximal clique $\delta_{\max}$ with the highest priority. This is because, as Z-list is sorted according to the priority P_{M-r}, and P equals P_{M-r} in these type algorithms, the head vertex in the Z-list is the vertex with the highest priority. So, the maximal clique $\delta_{\max}$ which covers the head vertex will be used to generate the encoded packet γ. Hence, the searching complexity is constant, namely $O(1)$.
- If the Z-list equals *NULL*, we need to search the D-list for finding the maximal clique $\delta_{\max}$ with the highest priority. *MAX* is initialized with the priority P of the head vertex in the D-list. As Z-list equals *NULL*, P only depends on $\delta_{\max}$ calculated by invoking AC (Section 5.1.2). Recalling that AC returns the maximal clique $\delta_{\max}$ which covers the given vertex. Searching for $\delta_{\max}$ with the highest priority starts from the next vertex in the D-list. If the next vertex has the Dg value higher than *MAX*, the priority P of that vertex needs to calculate. Only if P is higher than current *MAX*, *MAX* will be updated otherwise not. However, if the next vertex does not have the Dg value higher than *MAX*, search process will stop. This is because if Dg value of the next vertex is lower than *MAX*, the $\delta_{\max}$ of the corresponding vertex definitely will not be higher than *MAX*, as *Dg* value is the upper bound of the maximal clique size of that vertex. As vertices are sorted in descending order of Dg value, no other vertices will have a higher maximal clique size than *MAX*. So, there is no need to examined the other vertices in the list. In this way, this approach will skip computation of finding the maximal clique size of the rest of the vertices in the list with Dg value lower than *MAX*. The maximal clique of the vertex with *MAX* will be used to generate the encoded packet γ for broadcasting. The pseudo-code of this implementation is shown in lines 72–91 in Algorithm 2.
- If both the D-list and Z-list have entries, namely neither of the list is empty, we need to search both the lists for finding the maximal clique $\delta_{\max}$ with the highest priority. The search starts from the head of the D-list and setting *MAX* to its priority P (Refer to Table 3, priority P is calculated according to the QoS requirement of an algorithm, which is $P \leftarrow \delta_{\max} \times P_{M-r}$), which denotes the highest priority found so far. The P_{M-r} values in the Z-list then can be bounded using the *Dg* value in the next entry in the D-list. Since the D-list is sorted in descending order of Dg value, for any unexamined vertex in the Z-list to have a priority greater than *MAX*, it must have a P_{M-r} value greater than $limit(P_{M-r}) = \frac{MAX}{Next \rightarrow Dg}$, $Next \rightarrow Dg$ is the *Dg* value of the next unexamined vertex in the D-list. Since the Z-list is sorted in descending order of P_{M-r} value, this limit landmarks a point in the Z-list, below which no vertex can exceed the current highest priority (*MAX*). Therefore, no search is required. Next, the vertex at the head of the Z-list is examined and $limit(Dg) = \frac{MAX}{Next \rightarrow P_{M-r}}$ is set in the D-list, where $Next \rightarrow P_{M-r}$ is the P_{M-r} value of the next unexamined vertex in the Z-list. Then, we continue the search in the D-list and Z-list in an alternating fashion and update the two limits accordingly. The *MAX* value is updated only when an examined vertex with a priority higher than *MAX* is found. Recalling that the *Dg* value of a vertex is the upper bound of its maximal clique size ($\delta_{\max}$), so the priority of a vertex P, ($P \leftarrow \delta_{\max} \times P_{M-r}$) never exceeds its $Dg \times P_{M-r}$ value. Hence, we can skip computation of priority (P) of those unexamined vertices with $Dg \times P_{M-r}$ values equal or smaller than *MAX*, which means that we do not need to find the maximal cliques of those unexamined vertices. The search process stops when one of the limits is passed. At this point, *MAX* is the highest priority of all the vertices in G and the maximal clique $\delta_{\max}$ with the *MAX* will be used to generate the encoded packet γ for broadcasting. The pseudo-code of this implementation (Adaptive Demand-oriented Coding (ADC-2)) is shown in lines 95–131 in Algorithm 2.

5.1.4. Efficient approach

The detailed implementation of the efficient approach consists of the following three parts.

- Arrival of a new request: If a new request arrives, the efficient approach calculates M, $M - r$, P_{M-r} and *Dg* first and then constructs the D-list and Z-list following the rules in Section 5.1.3.
- Generation of encoded packet: The efficient approach finds the maximal clique with the highest priority and then generate the encoded packet.
- Broadcast and update: The efficient approach broadcasts the encoded packet and then updates the graph G.

The pseudo code of the efficient implementation is shown in Algorithm 2.

5.2. Complexity analysis of efficient implementation

For those scheduling algorithms with only Z-list (i.e., D-list is empty), the head vertex in the Z-list is the candidate vertex with the highest priority, namely, the maximal clique covering the head vertex is the maximal clique $\delta_{\max}$ with the highest priority. Hence, in the efficient implementation for these algorithms, the searching complexity is constant. For those scheduling algorithms with only D-list (i.e., Z-list is empty), it requires to search $\delta_{\max}$ with the highest priority in the sorted list of *Dg* value. Recalling that the *Dg* value of a vertex is the upper bound of $\delta_{\max}$ value of that vertex, and hence if a $\delta_{\max}$ is found among the top entries in the list which value is higher than the *Dg* value of the next bottom entry, we can skip searching all the remaining entries. Hence the efficient implementation reduces the searching complexity. In the next section, we will show quantitatively the comparative searching complexity between the straightforward and the efficient

implementations. For those scheduling algorithms with both D-list and Z-list (both lists are non-empty), it needs to search $\delta_{\max}$ with the highest priority in the sorted entries of both lists. In this case, for finding $\delta_{\max}$, the searching technique of *ADC-2* is adopted. Using the combination of the degree of vertex concept, and the efficient data structure and the pruning technique, the scheduling overhead can be effectively reduced for ADC-2.

5.3. An approximate version

Recalling that the efficient implementation guarantees that the request with the highest priority will be served. The approximate version introduces a tuning parameter for system administrator to strike a balance between the system performance and the cost in terms of searching overhead.

Inspired by Aksoy and Franklin [3], we present a self-adapting approximate algorithm for the efficient implementation (as described in Section 5.1). The approximate algorithm can further reduce the searching overhead by making sub-optimal encoding decisions. The self-adapting approximation algorithm works based on the maximum priority value of the last encoding decision by the scheduling algorithm. The approximate algorithm is proposed based on the two insights. Firstly, in skewed data access patterns, the vertex with the highest priority is usually found near at the very top of the list (D-list or Z-list). With more clients, the difference between the top and bottom entry of the sorted list becomes large. In this case the efficient searching process may search a lot of entries even though after encountering the vertex with the highest priority. This spends substantial resources for examining entries with the more number of clients. Secondly, in fixed data item access patterns and the fixed request arrival rate, the priority value of the vertex with the highest priority converges to a constant value.

The approximate version works similar with the efficient implementation. However, rather than searching the vertex with the highest priority, it searches for the first encountered vertex which has equal or higher priority than $\alpha \times THRESHOLD$, where $\alpha \geq 0$. If there is no such vertex, the approximate algorithm searches just like regular efficient implementation. The *THRESHOLD* value is initially set to 1 and recomputed on each broadcast tick. The *THRESHOLD* value after t broadcast tick is computed by,

$$THRESHOLD(t+1) = \frac{THRESHOLD(t) + MAX(t)}{2} \quad (1)$$

where $MAX(t)$ is the maximum priority value of the vertex for the broadcast at tick t.

α is a tuning parameter which determines the tradeoff between the system performance and the searching overhead. The higher value of α is, the more vertices will be searched, hence searching overhead improves a little. On the contrary, giving a smaller value of α will reduce more of the searching overhead. Accordingly, when α equals 1.0, the approximate version works similar as the efficient implementation, and when α equals 0.0, only the top entry of the list(s) is searched and chooses the one with the highest priority. So, the searching complexity becomes constant, namely $O(1)$.

The pseudo-code of the approximate of the efficient implementation of the scheduling algorithms with two non-empty lists invoke *Approx_ADC-2* is shown in lines 4–59 in Algorithm 3. The approximate implementation for the scheduling algorithms with one non-empty list (D-list) invoke *Approx_FindMaxClique_Productivity* is shown in lines 63–87 in Algorithm 3. The difference is that the scheduling algorithms with two non-empty lists search for the first encountered vertex with the priority no less than $\alpha \times THRESHOLD$ in both the D-list and Z-list and choose the one with the highest priority. However, the scheduling algorithms with one non-empty list do this search only in one list.

6. Experiments

6.1. Simulation model

The simulation model is built based on the system architecture illustrated in Fig. 1. It is implemented by CSIM19 [26]. Unless stated otherwise, the simulation is conducted under the default settings. The primary parameters are summarized in Table 4.

It is a closed system model, where a client submits a new request only when the previous one is satisfied or misses its deadline [41]. The *THINKTIME* refers to the time interval of generating new requests, which follows the Exponential distribution. Each request asks for one data item. The data access pattern follows the commonly used Zipf distribution [43]. Accordingly, the access probability of a data item d_j is, $P(d_j) = \frac{\frac{1}{j^\theta}}{\sum_{l=1}^{m} \frac{1}{l^\theta}}$, where, m is the number of data items in the database, and θ is the data access pattern parameter, which specifies the skewness of the distribution. Specifically, when $\theta = 0$, it follows the Uniform distribution. With an increasing value of θ, the data access pattern is getting more skewed. Based on a certain scheduling algorithm, the server encodes the requested data items and disseminates them via the broadcast channel. Clients monitor the broadcast channel and store the received data items into the local cache. Each client has the same cache size and the LRU cache replacement policy is adopted.

In real-time environments, in our simulation the relative deadline of the request submitted by client c_i (RD_i) is computed by:

$$RD_i = (1 + uniform(\mu^-, \mu^+)) * T_i^{serv}$$

Accordingly, The deadline of c_i (Dl_i) is computed by:

$$Dl_i = AT_i + RD_i$$

where AT_i is the arrival time of c_i. μ^- *and* μ^+ represent the minimum and the maximum laxity, respectively. T_i^{serv} represents the required service time for c_i. A request is feasible for serving only if its slack time is longer than the service time. Infeasible requests will be removed from the service queue.

The server disseminates one unit-sized data item in each broadcast tick. The scheduling is non-preemptive, which means that the transmission of the current data item cannot be interrupted [1].

6.2. Performance metrics

- Deadline Miss Ratio (DMR): It is the ratio of the number of requests missing deadlines to the total number of submitted requests. It measures the system capability on satisfying requests before their deadlines expire. Low deadline miss ratio implies better performance on real-time services.
- Average Response Time (ART): It is the duration from the instant when a request is submitted to the time when the corresponding data item is received. It measures the responsiveness of the system. Low ART implies better system responsiveness.

Table 4
Simulation parameters.

Parameter	Default	Range	Description
ClientNum	400	100 ~ 500	Number of clients
THINKTIME	0.01	–	Request generation interval
m	1000	–	Size of the database
CacheSize	60	–	Client cache size
θ	0.4	–	Zipf distribution parameter
μ^-, μ^+	120, 200	–	Min. and max. laxity

- Average Encode Length (AEL): It is the average number of data items which are selected to construct an encoded packet. It measures the coding flexibility and the power of exploiting coding opportunity of a scheduling algorithm. If the AEL is large, more clients can decode their required data items from an encoded packet.
- Computation overhead: It is the searching overhead for finding the vertex with the highest priority. It measures how many maximal cliques are computed and compared per broadcast before taking an encoded broadcast decision. Low computational overhead indicates better scalability of the algorithm.

The results were obtained when the system was in a steady state and the simulations continued until a confidence interval of 0.95 with half-widths of less than 5% about the mean was achieved.

Algorithm 3 The approximate version of the efficient implementation.

Function definition of Approximate ADC-2

Approx_ADC-2 (D-list, Z-list)
begin
  $pd \leftarrow$the head of D-list; $pz \leftarrow$the head of Z-list; $MAX \leftarrow 0$;
  $limit(Dg) \leftarrow 0$;
  $limit(P_{M-r}) \leftarrow 0$;
  /* Initialize variables for the approximate implementation */
  $Signal(Dg) \leftarrow FALSE$; $Signal(P_{M-r}) \leftarrow FALSE$;
  $MAX\_Approx \leftarrow 0$;
  while $pd \neq NULL || pz \neq NULL$ **do**
    if $pd \rightarrow Dg \geq limit(Dg)$ **then**
      if $(pd \rightarrow Dg \times P_{M-r}) > MAX$ **then**
        Invoke AC to find the maximal clique δ_{max} of the vertex pointed by pd;
        Calculate the priority P of the vertex pointed by pd (Refer to Table 3, $P \leftarrow \delta_{max} \times P_{M-r}$);
        if $P > MAX$ **then**
          $MAX \leftarrow P$;
          $\delta_{max} \leftarrow$ the maximal clique of the vertex with MAX;
        /* Checking the approximate condition */
        if $P \geq (\alpha \times THRESHOLD)$ **then**
          $MAX\_Approx \leftarrow P$;
          $\delta_{max} \leftarrow$ the maximal clique of the vertex with MAX;
          $Signal(Dg) \leftarrow TRUE$;
      Advance pd to the next unexamined vertex in the D-list;
      if $pd \neq NULL$ **then**
        $limit(P_{M-r}) \leftarrow \frac{MAX}{pd \rightarrow Dg}$;
    else
      break; /* $pd \rightarrow Dg < limit(Dg)$ */
    if $pz \rightarrow P_{M-r} \geq limit(P_{M-r})$ **then**
      if $(pz \rightarrow P_{M-r} \times Dg) > MAX$ **then**
        Invoke AC to find the maximal clique δ_{max} of the vertex pointed by pz;
        Calculate the priority P of the vertex pointed by pz (Refer to Table 3, $P \leftarrow \delta_{max} \times P_{M-r}$);
        if $P > MAX$ **then**
          $MAX \leftarrow P$;
          $\delta_{max} \leftarrow$ the maximal clique of the vertex with MAX;
        /* Checking the approximate condition */
        if $P \geq (\alpha \times THRESHOLD) \,\&\, P > MAX\_Approx$ **then**
          /* Ensure that MAX_Approx contains the largest value of the examined two entries in the two lists */
          $MAX\_Approx \leftarrow P$;
          $\delta_{max} \leftarrow$ the maximal clique of the vertex with MAX;
          $Signal(P_{M-r}) \leftarrow TRUE$;
      Advance pz to the next unexamined vertex in the Z-list;
      if $pz \neq NULL$ **then**
        $limit(Dg) \leftarrow \frac{MAX}{pz \rightarrow P_{M-r}}$;
    else
      break; /* $pz \rightarrow P_{M-r} < limit(P_{M-r})$ */
    /* Checking the terminating condition of approximate implementation */
    if $Signal(Dg) = TRUE \,||\, Signal(t) = TRUE$ **then**
      break;/* No more searching */
  /* Update the THRESHOLD for the next schedule of the approximate implementation */
  if $Signal(Dg) = TRUE \,||\, Signal(t) = TRUE$ **then**
    /* $THRESHOLD$ is updated with MAX_Approx */
    $THRESHOLD \leftarrow \frac{THRESHOLD + MAX\_Approx}{2}$;
  else
    /* $THRESHOLD$ is updated with MAX */
  $THRESHOLD \leftarrow \frac{THRESHOLD + MAX}{2}$;
  return δ_{max} and $THRESHOLD$;

```
Function definition of Approximate FindMaxClique_Productivity

Approx_FindMaxClique_Productivity (D-list)
begin
    /* Input:  D-list;
    Output:  The maximal clique δ_max covering the vertex with the highest priority and the calculated new THRESHOLD ;
                                                                                                              */
    pd ← the head of D-list;
    MAX ← δ_max of the vertex pointed by pd by invoking AC;
    Advance pd to the next unexamined vertex in the D-list;
    while pd ≠ NULL do
        if pd → Dg > MAX then
            P ← δ_max of the vertex pointed by pd by invoking AC;
            if P ≥ (α × THRESHOLD) then
                MAX ← P;
                break;
            else if P > MAX then
                MAX ← P;
                δ_max ← the maximal clique of the vertex with MAX;
        else
            /* pd → Dg is equal to or smaller than MAX, so no need further searching                    */
            break;
        Advance pd to the next unexamined vertex in the D-list;
    /* Update THRESHOLD with MAX                                                                        */
    THRESHOLD ← (THRESHOLD+MAX)/2 ;
    return δ_max and THRESHOLD;
```

6.3. Performance analysis

In this section, firstly, we show the comparative performance of straightforward network coding implementation vs. proposed efficient implementation vs. non-network coding implementation. Secondly, we evaluate the performance of different versions of approximate implementations, e.g., for under different α values. Thirdly, we show by retaining the same system performance, how much computation overhead can be reduced in the efficient network coding implementation compared to the straightforward implementation. Lastly, we show the computation overhead gain of different approximates of efficient implementation. For the sake of clear exposure of the main contribution, we pick one representative scheduling algorithm from each category and discuss their comparative performance in different implementations. To be specific, we choose EDF for representing the scheduling algorithms groups, which do not require data productivity as QoS metrics; we choose MRF for representing the scheduling algorithms groups which require only data productivity as QoS metric; and we choose $R \times W$ for representing the scheduling algorithms, which require additional QoS metric(s) including data productivity as QoS metrics. Please note that 'EDF' stands for the traditional non-coded EDF scheduling, 'EDF_N' stands for the EDF with straightforward network coding implementation and 'EDF_N Efficient' stands for the EDF with efficient network coding implementation. The other scheduling algorithms also use the notations in the same fashion.

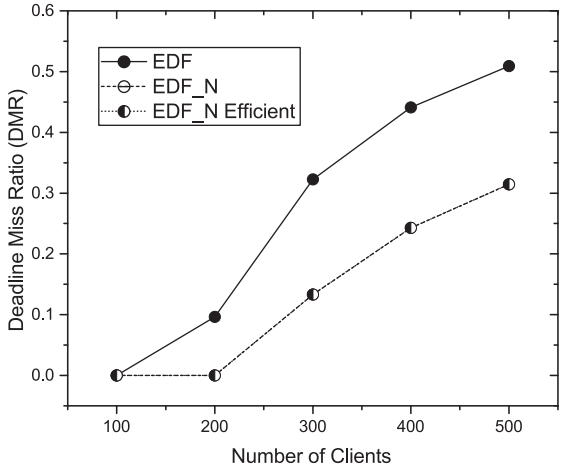
6.3.1. Straightforward coding vs. efficient coding vs. without coding

Fig. 3 shows the comparative performance of the representative scheduling algorithms with straightforward coding, with proposed efficient coding and without coding. Figs. 3(a), (c), and (e) show that as expected, with coding has better performance than without coding under diverse workloads. Note that as the proposed efficient network coding implementation does not compromise with the system performance, the system performance of the straightforward network coding implementation and the efficient implementation is the same for all the representative scheduling algorithms. However, the efficient implementation reduces the computation overhead significantly than the straightforward implementation, which will be shown in Section 6.4.

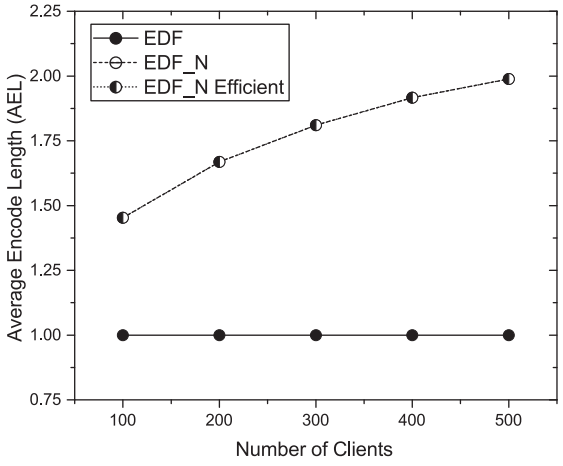
With the increasing number of clients, the performances of all the algorithms both with coding and without coding decline. This is because, under more clients, the service workload increases. However, the performance of an algorithm with coding, declines at a slower pace compared to its without coding version. This is because, coding opportunity increases with the increasing workload. When more requests are submitted, the chance of one request being cached by a client is being requested by other clients increases. Hence, more clients can be satisfied by a broadcast tick in a heavier system workload environment, which is the key of performance improvement, as shown in Figs. 3(b), (d), and (f). With the default settings, the performance improvements of EDF, MRF and R × W with network coding are 45%, 44% and 43%, respectively than their corresponding without network coding versions.

6.3.2. Efficient vs. approximate implementation

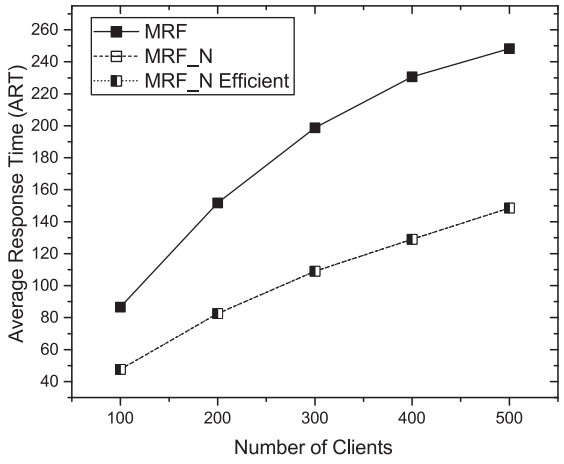
Fig. 4 shows the comparative performance of efficient network coding implementation against its approximate implementation. Figs. 4(a) and (b) show the performance of different approximates of efficient implementation of MRF_N and RXW_N, respectively. The approximate implementations reduce the search space at a possible expense of suboptimal decisions. The approximates of MRF_N with α values 1.0, 0.9, 0.8, and 0.0 are denoted by *MRF_N Approximate (1.0), MRF_N Approximate (0.9), MRF_N Approximate (0.8), and MRF_N Approximate (0.0),* respectively. Among the approximates, only the one with α equals 0.0 has a visible lower performance than the efficient implementation when workload is high; other approximates have almost the same performance of efficient implementation. However, note that still the gain of *MRF_N Approximate (0.0)* is about 40% than the without coding implementation. Nevertheless, this result will be more pronounced when we study the comparative computation overhead in the following section. Fig. 4(b) shows the performance of different approximates of efficient implementation of R × W_N. The *R × W_N Approximate (1.0)* has almost the same performance of the efficient implementation of R × W_N. *R × W_N Approximate (0.9)* pays around 2% penalty of increased response time than the efficient



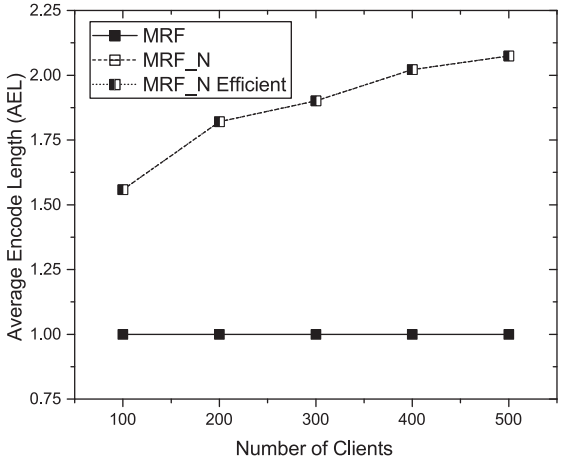
(a) Deadline miss ratio: EDF vs. EDF_N vs. EDF_N Efficient.



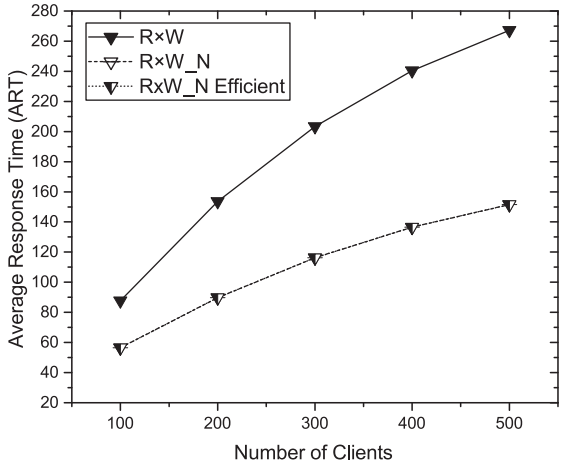
(b) Average encode length: EDF vs. EDF_N vs. EDF_N Efficient.



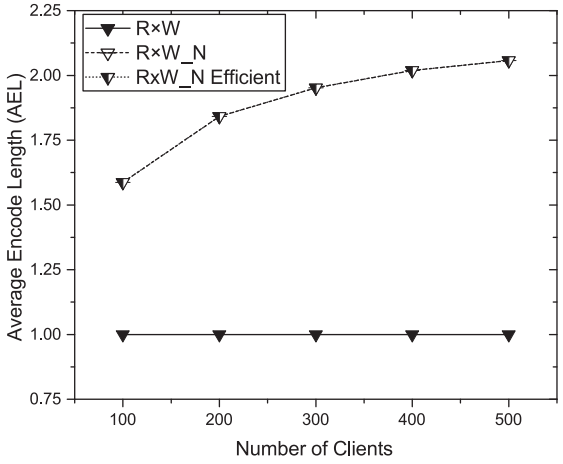
(c) Average response time: MRF vs. MRF_N vs. MRF_N Efficient.



(d) Average encode length: MRF vs. MRF_N vs. MRF_N Efficient.



(e) Average response time: R×W vs. R×W_N vs. R×W_N Efficient.



(f) Average encode length: R×W vs. R×W_N vs. R×W_N Efficient.

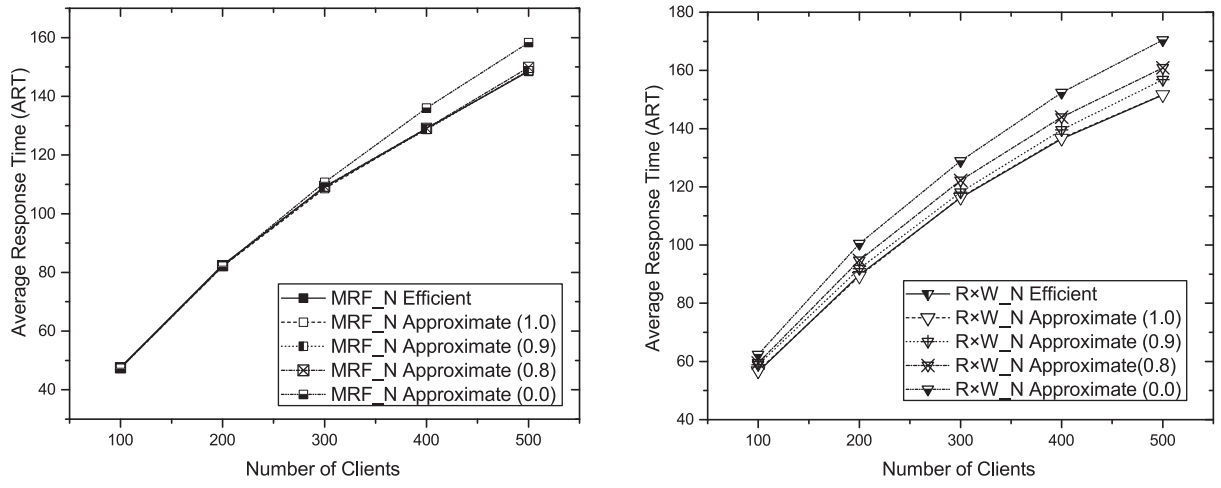
Fig. 3. Performance of without coding vs. straightforward coding vs. efficient coding.

implementation of R × W_N. Among the approximates, *R × W_N Approximate (0.0)* pays the maximum penalty in terms of increasing the response time which is around 11% from the efficient implementation. However, it still achieves a 36% lower response time than the without coding implementation.

6.4. Computation overhead analysis

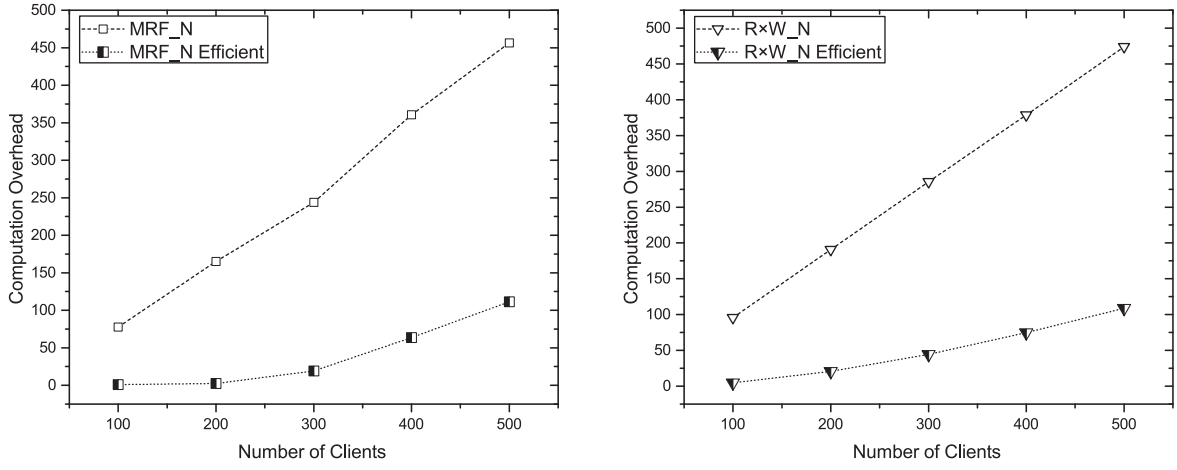
6.4.1. Straightforward vs. efficient coding implementation

Fig. 5 shows the computation overhead of the straightforward and efficient implementations. With the increasing number of



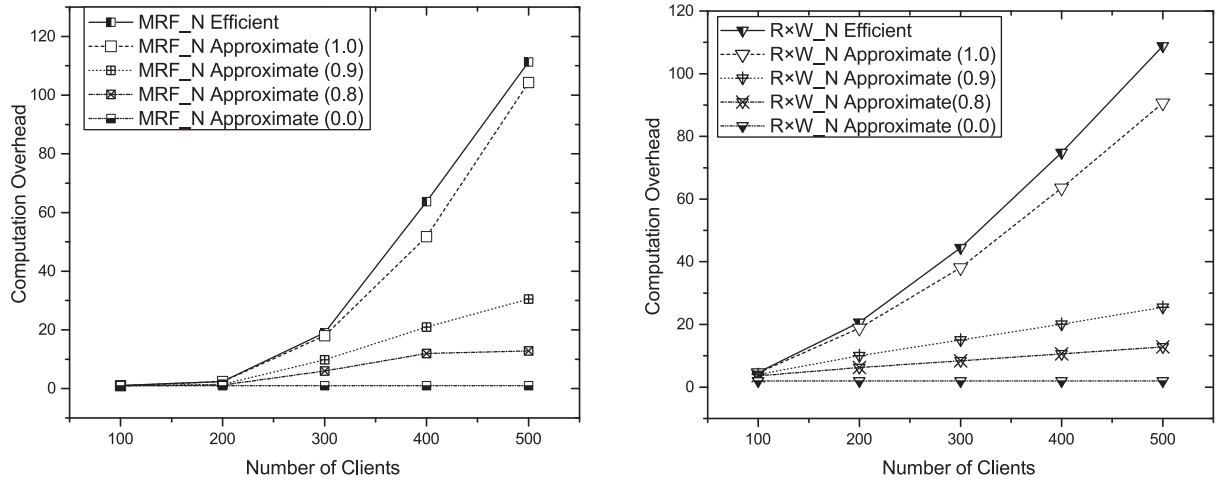
(a) MRF_N. (b) R×W_N.

Fig. 4. Performance of efficient vs. approximate implementations.



(a) Straightforward MRF_N vs. Efficient MRF_N (b) Straightforward R×W_N vs. Efficient R×W_N

Fig. 5. Computation overhead of straightforward vs. efficient implementation.



(a) MRF_N. (b) R×W_N.

Fig. 6. Computation overhead of efficient vs. approximate implementation.

clients, the number of pending requests increases which results in higher computational overhead. Fig. 5 shows that with the increasing workload, straightforward implementation endures a very high computation overhead. however, the efficient implementation can keep this rate very small. For MRF_N, when the workload is low (number of clients equals 100), while the straightforward implementation requires more than 77 maximal cliques to search the maximal clique with the highest priority, the efficient implementation completes this search just by comparing with one maximal clique. Under the default settings, the efficient implementation reduces the computation overhead by more than 75% compared with the straightforward implementation. Similar result is observed for RXW_N. These results support our claim of the effectiveness of the efficient network coding implementation. Note that for the scheduling algorithm which considers the QoS metric(s) other than the data item productivity, it has the candidate vertex with the highest priority at the top of the Z-list. Therefore, for both the straightforward and efficient network coding implementations, the searching complexity is $O(1)$.

6.4.2. Efficient vs. approximate implementation

Fig. 6 shows the comparative computation overhead of efficient and approximate implementations. The computation overhead of different approximates of efficient implementation of MRF_N are shown in Fig. 6(a). As discussed earlier, with the smaller value of α, the searching space reduces significantly. The efficient implementation with α equaling 0.0 has the constant overhead $O(1)$. In compare with the system performance and computation overhead (Figs. 4(a) and 6(a)), it is clear that with a very little expense of system performance, searching overhead can be improved significantly.

Fig. 6(b) compares the computation overhead among different approximates of the efficient implementation of R × W_N. Under the default settings, from Figs. 4(b) and 6(b), $R \times W\_N$ *Approximate (1.0)* reduces the overhead around 15% than the efficient implementation of $R \times W\_N$ with a very marginal expense (0.16%) of the increased average response time, whereas $R \times W\_N$ *Approximate (0.9)* reduces the overhead by 73% from the efficient implementation with a small increase (2.2%) in the average response time. Moreover, in compare with the straightforward implementation this overhead reduction is 95%. $R \times W\_N$ *Approximate (0.0)* has the constant searching complexity ($O(1)$), namely, the searching complexity does not increase with the increasing workload. However, it compensates by only the 11% increased response time from the efficient implementation of $R \times W\_N$.

7. Discussion and conclusion

In this study, we have verified that conventional scheduling algorithms in on-demand broadcast environments cannot best explore the bandwidth efficiency on large scale information services. Since network coding has the ability to serve multiple data items by a single broadcast by an encoded packet, multiple clients which are pending for different data items can be satisfied simultaneously. Therefore, it is imperative to explore the benefit of network coding for enhancing the QoS of different types applications, such as reducing deadline miss ratio and request response time for real-time and non-real-time applications, respectively.

Specifically, this work first investigated the performance improvement of applying network coding in on-demand broadcast over the traditional broadcast system. Second, we showed that the straightforward network coding implementation requires overwhelming computational overhead for making encoding decisions. Third, we proposed an efficient generalized network coding implementation, which achieves the same performance with straightforward network coding implementation but reduces the coding implementation overhead significantly. Fourth, we proposed an approximate of efficient network coding implementation, which can further reduce the computational overhead while achieving sub-optimal performance. The approximate implementation designs a tuning parameter for system administrator to strike a balance between the service performance and the system scalability. Finally, we conducted an extensive performance evaluation, which conclusively demonstrated the superiority of our proposed framework and solutions.

Acknowledgment

This work was supported in part by the NTU-NXP Intelligent Transport System Test-Bed Living Lab Fund S15-1105-RF-LLF from the Economic Development Board, Singapore; the National Science Foundation of China under Grant No. 61872049 and 61572088; the Fundamental Research Funds for the Central Universities (Project No. 2018CDQYJSJ0034); and the Venture and Innovation Support Program for Chongqing Overseas Returnees (Project No. cx2018016).

References

[1] S. Acharya, S. Muthukrishnan, Scheduling on-demand broadcasts: new metrics and algorithms, in: Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98), 1998, pp. 43–54.
[2] R. Ahlswede, N. Cai, S.-Y. Li, R.W. Yeung, Network information flow, IEEE Trans. Inf. Theory 46 (4) (2000) 1204–1216.
[3] D. Aksoy, M. Franklin, $R \times W$: a scheduling approach for large-scale on-demand data broadcast, J. IEEE/ACM Trans. Netw.(TON) 7 (6) (1999) 846–860.
[4] G.G.M.N. Ali, M. Noor-A-Rahim, M.A. Rahman, S.K. Samantha, P.H.J. Chong, Y.L. Guan, Efficient real-time coding-assisted heterogeneous data access in vehicular networks, IEEE Internet Things J. (2018) 1.
[5] G.G.M.N. Ali, M.A. Rahman, S.K. Samantha, Y. Gao, P.H.J. Chong, Y.L. Guan, Efficient coding based heterogeneous data access in vehicular networks, in: 2017 IEEE International Conference on Communications (ICC), 2017, pp. 1–6.
[6] Y. Birk, T. Kol, Coding on demand by an informed source (iscod) for efficient broadcast of different supplemental data to caching clients, IEEE/ACM Trans. Netw. 14 (SI) (2006) 2825–2830.
[7] M. Chaudhry, A. Sprintson, Efficient algorithms for index coding, in: INFOCOM Workshops 2008, IEEE, 2008, pp. 1–4.
[8] C.-C. Chen, C. Lee, S.-C. Wang, On optimal scheduling for time-constrained services in multi-channel data dissemination systems, Inf. Syst. 34 (1) (2009) 164–177.
[9] J. Chen, V. Lee, K. Liu, On the performance of real-time multi-item request scheduling in data broadcast environment, J. Syst. Softw. 83 (8) (2010) 1337–1345.
[10] J. Chen, V. Lee, K. Liu, J. Li, Efficient cache management for network coding assisted data broadcast, IEEE Trans. Veh. Technol. PP (99) (2016) 1.
[11] J. Chen, V.C. Lee, K. Liu, G. Ali, E. Chan, Efficient processing of requests with network coding in on-demand data broadcast environments, Inf. Sci. 232 (0) (2013) 27–43.
[12] C.-H. Chu, D.-N. Yang, M.-S. Chen, Multi-data delivery based on network coding in on-demand broadcast, in: Proceedings of the The Ninth International Conference on Mobile Data Management, 2008, pp. 181–188.
[13] A. Dharwadker, The clique algorithm, Accessed: 2018-07-20.
[14] C.-L. Hu, Fair scheduling for on-demand time-critical data broadcast, in: IEEE International Conference on Communications (ICC'07), IEEE, 2007, pp. 5831–5836.
[15] H. Ji, V.C. Lee, C.-Y. Chow, K. Liu, G. Wu, Coding-based cooperative caching in on-demand data broadcast environments, Inf. Sci. 385–386 (2017) 138–156.
[16] R. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher, J. Bohlinger (Eds.), Complexity of Computer Computations, The IBM Research Symposia Series, Springer US, 1972, pp. 85–103.
[17] C.-M. Liu, T.-C. Su, Broadcasting on-demand data with time constraints using multiple channels in wireless broadcast environments, Inf. Sci. 242 (2013) 76–91.
[18] K. Liu, L. Feng, P. Dai, V.C.S. Lee, S.H. Son, J. Cao, Coding-assisted broadcast scheduling via memetic computing in sdn-based vehicular networks, IEEE Trans. Intell. Transp. Syst. (2017) 1–12.
[19] K. Liu, V. Lee, On-demand broadcast for multiple-item requests in a multiple-channel environment, Int. J. Inf. Sci. 180 (22) (2010) 4336–4352.
[20] K. Liu, V. Lee, Adaptive data dissemination for time-constrained messages in dynamic vehicular networks, Transp.Res.Part C 21 (1) (2012) 214–229.
[21] K. Liu, J. Ng, V. Lee, S. Son, I. Stojmenovic, Cooperative data scheduling in hybrid vehicular ad hoc networks: vanet as a software defined network, Netw. IEEE/ACM Trans. 24 (3) (2016) 1759–1773.
[22] Z. Lu, W. Wu, W.W. Li, M. Pan, Efficient scheduling algorithms for on-demand wireless data broadcast, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, 2016, pp. 1–9.

[23] J. Lv, V.C. Lee, M. Li, E. Chen, Profit-based scheduling and channel allocation for multi-item requests in real-time on-demand data broadcast systems, Data Knowl. Eng. 73 (2012) 23–42.
[24] G. Nawaz Ali, P.H.J. Chong, S.K. Samantha, E. Chan, Efficient data dissemination in cooperative multi-RSU vehicular ad hoc networks (VANETs), J. Syst. Softw. 117 (C) (2016) 508–527.
[25] Y.E. Sagduyu, R.A. Berry, D. Guo, Throughput and stability for relay-assisted wireless broadcast with network coding, IEEE J. Sel. Areas Commun. 31 (8) (2013) 1506–1516.
[26] H. Schwetman, CSIM19: a powerful tool for building system models, in: Proceedings of the 33th IEEE Winter Simulation Conference, 2001. Arlington, VA, USA
[27] M.A. Sharaf, P.K. Chrysanthis, On-demand broadcast: New challenges and scheduling algorithms, First Hellenic Data Management Symposium (HDMS), 2002.
[28] S. Sorour, S. Valaee, On minimizing broadcast completion delay for instantly decodable network coding, in: IEEE International Conference on Communications (ICC'10), IEEE, 2010, pp. 1–5.
[29] Y. Sui, X. Wang, J. Wang, L. Wang, S. Hou, Deadline-aware cooperative data exchange with network coding, Comput. Netw. 97 (2016) 88–97.
[30] J.-Y. Wang, Set-based broadcast scheduling for minimizing the worst access time of multiple data items in wireless environments, Inf. Sci. 199 (2012) 93–108.
[31] S. Wang, C. Yan, Z. Yu, Efficient coding-based scheduling for multi-item requests in real-time on-demand data dissemination, Mobile Inf. Syst. 2016 (2016).
[32] S. Wang, J. Yin, Distributed relay selection with network coding for data dissemination in vehicular ad hoc networks, Int. J. Distrib. Sens. Netw. 13 (5) (2017). 1550147717708135
[33] X. Wang, C. Yuen, Y. Xu, Coding-based data broadcasting for time-critical applications with rate adaptation, IEEE Trans. Veh. Technol. 63 (5) (2014) 2429–2442.
[34] J.W. Wong, Broadcast delivery, Proc. IEEE 76 (12) (1988) 1566–1577.
[35] J.W. Wong, M.H. Ammar, Analysis of broadcast delivery in videotex system, J. IEEE Trans. Comput. 34 (9) (1985) 863–866.
[36] X. Wu, V.C. Lee, Wireless real-time on-demand data broadcast scheduling with dual deadlines, Parallel Distrib. Comput. 65 (6) (2005) 714–728.
[37] J. Xu, X. Tang, W. Lee, Time-critical on-demand data broadcast algorithms, analysis and performance evaluation, IEEE Trans. Parallel Distrib. Syst. 17 (1) (2006) 3–14.
[38] P. Xuan, S. Sen, O. Gonzalez, J. Fernandez, K. Ramamritham, Efficient and timely dissemination of data in mobile environments, in: Proceedings of the 3rd IEEE Real Time Technology and Applications Symposium (RTAS'97), 1997. Montreal, Canada
[39] D.-N. Yang, M.-S. Chen, On bandwidth-efficient data broadcast, Knowl. Data Eng. IEEE Trans. 20 (8) (2008) 1130–1144.
[40] M. Yu, P. Sadeghi, A. Sprintson, Feedback-assisted random linear network coding in wireless broadcast, in: 2016 IEEE Globecom Workshops (GC Wkshps), 2016, pp. 1–6.
[41] C. Zhan, V. Lee, J. Wang, Y. Xu, Coding-based data broadcast scheduling in on-demand broadcast, Wireless Commun. IEEE Trans. 10 (11) (2011) 3774–3783.
[42] C. Zhan, F. Xiao, Memory aware broadcast for wireless real time applications, in: 2015 IEEE 5th International Conference on Electronics Information and Emergency Communication, 2015, pp. 84–87.
[43] G.K. Zipf, Human Behaviour and the Principle of Least Effort: An Introduction to Human Ecology, Addison-Wesley, 1949.

G. G. Md. Nawaz Ali received his B.Sc. degree in Computer Science and Engineering from the Khulna University of Engineering & Technology, Bangladesh in 2006, and the Ph.D. degree in Computer Science from the City University of Hong Kong, Hong Kong in 2013 with the Outstanding Academic Performance Award. He is currently a postdoctoral research fellow with the Department of Automotive Engineering, The Clemson University - International Center for Automotive Research (CU-ICAR), Greenville, SC, USA. From October 2015 to March 2018, he was a postdoctoral research fellow with the School of Electrical and Electronic Engineering of Nanyang Technological University (NTU), Singapore. He is a reviewer of a number of international journals including the IEEE Transactions on Intelligent Transportation Systems and Magazine, IEEE Transactions on Vehicular Technology, Wireless Networks etc. His current research interests include Vehicular Cyber Physical System (VCPS), wireless broadcasting, mobile computing, and network coding.

Kai Liu received his Ph.D. degree in Computer Science from City University of Hong Kong in 2011. He is currently an Assistant Professor in the College of Computer Science, Chongqing University, China. He has been a visiting scholar in Computer Science Department at the University of Virginia, USA, from Dec. 2010 to May 2011. He has been a Postdoctoral Fellow at Singapore Nanyang Technological University, City University of Hong Kong and Hong Kong Baptist University from 2011 to 2014. His research interests include mobile computing, intelligent transportation systems and vehicular cyber-physical systems.

Victor C.S. Lee received the PhD degree in computer science from the City University of Hong Kong in 1997. He is currently an Assistant Professor in the Department of Computer Science, City University of Hong Kong. His research interests include data dissemination in vehicular networks, real-time databases and performance evaluation. He is a member of the ACM, IEEE and IEEE Computer Society. He has been the chairman of the IEEE, Hong Kong Section, Computer Chapter in 2006–2007.

Peter H. J. Chong is currently the Professor and Head of the Department of Electrical and Electronic Engineering, Auckland University of Technology, New Zealand. He received the B.Eng. (with distinction) in electrical engineering from the Technical University of Nova Scotia, Halifax, NS, Canada, in 1993, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of British Columbia, Vancouver, BC, Canada, in 1996 and 2000, respectively. Between 2000 and 2001, he worked at Agilent Technologies Canada Inc., Canada. From 2001 to 2002, he was at Nokia Research Center, Helsinki, Finland. From May 2002 to 2016, he was with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore as an Associate Professor (Tenured). He was an Assistant Head of Division of Communication Engineering between 2011 and 2013, since July 2013 to April 2016, he was the Director of Infinitus, Centre for Infocomm Technology in School of EEE. He has visited Tohoku University, Japan, as a Visiting Scientist in 2010 and Chinese University of Hong Kong (CUHK), Hong Kong, between 2011 and 2012. He is currently an Adjunct Associate Professor of CUHK.

Yong Liang Guan obtained his PhD from Imperial College of Science, Technology and Medicine (University of London), UK, in 1997, and B.Eng. degree with first class honors from the National University of Singapore in 1991. In 1991–1993, he worked in the cellular and wireless industry in Singapore. He is now an Associate Professor with the School of Electrical and Electronic Engineering, and the Director of the Positioning and Wireless Technology Center, at the Nanyang Technological University, Singapore. His research interests broadly include modulation, coding and signal processing for communication systems and information security systems. He has led 10 past and present externally-funded research projects (ASTAR, DSTA/DSO, industry) with cumulative funding of about SGD 4 million. He is an Associate Editor of the IEEE Signal Processing Letter, and has served in the organizing/technical committees of international conferences including ICC, VTC, and ICUWB. He has been granted 1 patent and has filed another 3. He has published an invited monograph, over 160 journal and conference papers, and 3 book chapters.

Jun Chen received her Ph.D degree in Computer Science from the Wuhan University of China in 2008. She was a senior research associate in the Department of Computer Science, City University of Hong Kong from 2008 to 2009. She was a visiting fellow in the Department of Information System, City University of Hong Kong in 2012. She is currently an Associate Professor in the School of Information Management of the Wuhan University in China. Her research interests include mobile computing and data management in wireless networks.