IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON EMERGING TRENDS, ISSUES, AND CHALLENGES IN ENERGY-EFFICIENT CLOUD COMPUTING

Received April 6, 2017, accepted May 8, 2017, date of publication May 29, 2017, date of current version June 28, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2706328

Towards Green IoT Networking: Performance Optimization of Network Coding Based Communication and Reliable Storage

JIAN LI[1], YUN LIU[1], ZHENJIANG ZHANG[1], JIAN REN[2], (Senior Member, IEEE), AND NAN ZHAO[3]

[1]School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China
[2]Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI 48824, USA
[3]School of Telecommunications Engineering, Xidian University, Xi'an 710071, China

Corresponding author: Jian Li (lijian@bjtu.edu.cn)

This work was supported in part by Beijing Jiaotong University under Talent Grant KWRC16004536, in part by the National Natural Science Foundation of China under Grant 61301171, in part by the National Key Research and Development Program of China under Grant 2016QY06X1203, and in part by the National Natural Science Foundation of China under Grant 61172072 and Grant 61401015.

ABSTRACT Internet of things (IoT) is expanding its outreach to almost every aspect of our daily life. By utilizing network coding in IoT, the IoT energy consumption can be reduced. Thus it is worthwhile studying and improving the applications in IoT, where network coding is incorporated. In this paper, we optimize the performance of network coding-based communication and reliable storage in two important components of IoT, including the IoT core network, where data is sensed and transmitted, and the distributed cloud storage, where the data generated by the IoT core network is stored. First, we propose an adaptive network coding scheme in the IoT core network to improve the transmission efficiency. We demonstrate the efficacy of the scheme and the performance advantage over existing schemes through simulations. Second, we introduce the optimal storage allocation problem in the network coding-based distributed cloud storage, which aims at searching for the most reliable allocation that distributes the n data components into N data centers, given the failure probability p of each data center. Finally, we propose a polynomial-time optimal storage allocation (OSA) scheme to solve the problem. Both the theoretical analysis and the simulation results show that the storage reliability could be greatly improved by the OSA scheme.

INDEX TERMS Internet of Things, wireless sensor networks, distributed cloud storage, green networking.

I. INTRODUCTION

Internet of things (IoT) [1], [2] is an integral part in today's development of smart city. People could remotely access and interact with a wide range of devices integrated with sensors, from home appliances, wearable electronics to environmental monitors. With such enormous coverage potential in our daily life, IoT with reduced energy consumption (the 'green' attribute) has attracted more and more attention. In recent years, energy-efficient networking and computing [3] have been extensively studied from many perspectives, such as the framework design [4], the algorithm design [5] and the resource reusing [6].

The high level view of Internet of things is shown in Fig. 1, which includes IoT core network for data sensing and transmission, distributed cloud storage [7] for storing the data generated by the core network, cloud computing [8] for processing the data. Upon these components are various applications such as e-transportation, e-heath, smart home and so on. Communication networks such as 4G and 5G networks [9] interconnect these major components.

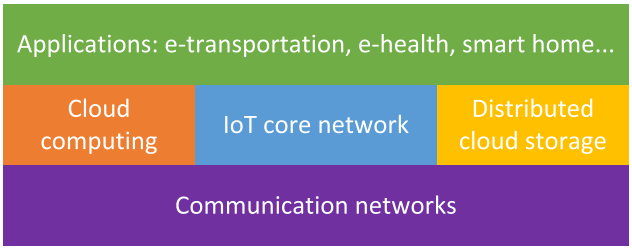


FIGURE 1. High level view of Internet of things.

The IoT core network is responsible for generating the data for Internet of things. Smart devices sense various data and send out the data through the networks constituted by these devices [10], [11]. Because the smart devices are mostly battery-driven, plenty of researches aim at devising energy-efficient schemes to prolong the operation time of the network, such as in [12]–[14]. Moreover, since the communication of the IoT core network is largely through wireless, the packet loss may be high due to fadings and interferences, which would bring unnecessary energy

2169-3536 © 2017 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

consumptions. Thanks to the emerging of software defined wireless networking [15], we can apply sophisticated algorithms to improve the communication quality of the IoT core network and conserve energy.

During the operation of Internet of things, data collected from a vast number of sensors in the IoT core network could explode. The distributed cloud storage is the best candidate to safely and reliably store these data. The distributed data storage architecture model distributes the database to multiple servers in many locations across the participating network in the storage cloud. Each location is directly and independently plugged into the Internet. If something unexpected happens to the data in one location, generally only a small amount of backed up data is impacted. Thus the data could be recovered with much less energy consumption, using the data stored in the rest of the locations. Besides distributing the data, many researches also study proactive approaches to ensure the data availability such as in [16].

Network coding provides a trade-off between communication capacity and computational complexity in network environment by enabling the intermediate relay nodes to encode the incoming packets before forwarding them. Since network coding can improve the throughput and robustness of the network, the unnecessary energy consumption due to high loss rate communication or high failure rate storage can be saved. Application of network coding in Internet of things can contribute to the 'green IoT networking'. With such desirable merit, in this paper we optimize the performance of network coding based communication and reliable storage in Internet of things. The main contributions of this paper are:

- We propose an adaptive network coding scheme (ANC scheme) for the IoT core network and demonstrate that the scheme can improve the transmission efficiency and the performance is better than existing schemes.
- For the distributed cloud storage utilizing network coding that stores the data generated by the IoT core network, we introduce the optimal storage allocation problem and propose an optimal storage allocation (OSA) scheme. Simulation results show that the storage reliability can be greatly improved.

The paper is organized as follows: in Section II we briefly review the IoT core network and the distributed cloud storage. The concept of network coding and its advantages in communication and storage are also introduced in Section II. Next we propose and analyze our adaptive network coding scheme for the IoT core network in Section III. After that we study the optimal storage allocation problem in the distributed cloud storage utilizing network coding in Section IV. At last is the Conclusion.

II. PRELIMINARIES AND RELATED WORK

A. INTERNET OF THINGS

The objective of Internet of things is to equip everything related to human beings with smart chips integrating sensors, actuators and transceivers. Smart devices equipped with smart chips within a certain range can communicate with each other and form networks. These networks can be further connected to the Internet through proper interconnecting. In this paper, we will mainly focus on the IoT core network and the distributed cloud storage which stores the data generated by the IoT core network as shown in Fig. 1.

1) IoT CORE NETWORK

IoT core network consists of the smart devices mentioned above and the networks among these devices. Several challenges of the IoT core network exists. First, it lacks of a unified infrastructure and protocol stack. Second, the monitor and control of the network lacks of flexibility. Third, the functionality of the network cannot be changed without reprogramming the smart devices when the application environment changes.

To overcome these shortcomings, software defined wireless networking (SDWN) [15] was proposed based on the paradigm of software defined networking (SDN) [17]. In the context of SDWN, the network elements in the data plane are smart devices which act as both end users and switches. The data flow is separated from the control flow. We can easily change the network behaviors through exchange of the control flow among smart devices.

Thanks to the advantages of SDWN, it is much easier to implement algorithms which can improve network performance into IoT core network. In [18] the authors propose to combine network coding and software defined networking, where the code rate of the network coding is fixed. Although this approach can improve the communication throughput, the strategy is not flexible to cope with the changing channel qualities in wireless environments. In this paper, we will show that the transmission efficiency of the IoT core network can be greatly improved through our adaptive network coding scheme, where the code rate of the network coding can be dynamically adjusted in a centralized manner with the global view of the whole network.

2) DISTRIBUTED CLOUD STORAGE

The volume of all global data will be boosted dramatically with the developing of Internet of things, where there will be hundreds of thousands of sensors deployed to create more and more data. To the year 2020, the amount of data would grow to 40 zettabytes. How to properly store the data has become a major challenge in Internet of things.

Traditional centralized data center is not suitable for the context of Internet of things. If something unexpected happens such as power outage or military actions, the precious data stored in the data center could be lost and unrecoverable. To ensure a high reliability of the data storage, a typical solution is to store the data across multiple servers in the distributed cloud storage. The main idea is that instead of storing the entire data in one server, we can split the data into n data components and store the components separately. The original data can be recovered only when the required

(threshold) number of components, say k, are collected. The storage efficiency is much higher than simply replicating the data over multiple servers. The distributed cloud storage can also increase data availability while reducing network congestion, thus leading to increased resiliency. A popular approach is to employ an (n, k) maximum distance separable (MDS) code, such as the Reed-Solomon (RS) code in the Total Recall system [19]. In later sections, we will show that by applying network coding in the distributed cloud storage we can further improve the performance of data storage.

B. NETWORK CODING

In this section, we will briefly introduce the concept of network coding and its advantages in improving the communication throughput and the distributed cloud storage reliability, which could eventually contribute to the 'green IoT networking'. Network coding was first introduced in the seminal paper by Ahlswede *et al.* [20]. By allowing the intermediate relay nodes to encode the incoming packets, the network could achieve the maximum multicast capacity.

A network is equivalent to a directed graph $G = (V, E)$, where V represents the set of vertices corresponding to the network nodes and E represents all the directed edges between vertices corresponding to the communication link. The start vertex v of an edge e is called the tail of e and written as $v = tail(e)$, while the end vertex u of an edge e is called the head of of e and written as $u = head(e)$. For a source node u, there is a set of symbols $\mathcal{X}(u) = (x_1, \ldots, x_k)$ to be sent. Each of the symbol is from the finite field $GF(2^m)$, where m is a positive integer. For a link e between intermediate nodes r_1 and r_2, written as $e = (r_1, r_2)$, the symbol y_e transmitted on it is the function of all the $y_{e'}$ such that $head(e') = r_1$. And y_e can be written as:

$$y_e = \sum_{e':head(e')=r_1} \beta_{e',e} \cdot y_{e'}, \tag{1}$$

in which the encoding coefficients $\beta_{e',e} \in GF(2^m)$. For a sink node v, there is a set of incoming symbols $y_{e'}$ $(e' : tail(e') = v)$ to be decoded.

1) NETWORK CODING IN COMMUNICATION

The main idea of network coding can be illustrated through Fig. 2. Assume the capacity of all the edges is C, the capacity of this network is $2C$ according to the max-flow min-cut theorem. Only by encoding the incoming packet symbols x_1, x_2 at node R3, this network can achieve the maximum capacity. *Since node R3 only need to send out one symbol $x_1 + x_2$ instead of two symbols, the energy consumption of R3 in data transmission could be reduced by* 50%.

In [21] and [22], the authors have shown that linear codes with random selected coefficients are sufficient to achieve the multicast capacity by coding on a large enough field. Sink nodes that have received more linear independent encoded symbols than the original symbol generated by the source nodes can easily decode the original symbols by solving a set of linear equations. Moreover, it has been demonstrated that network coding can improve the communication throughput.

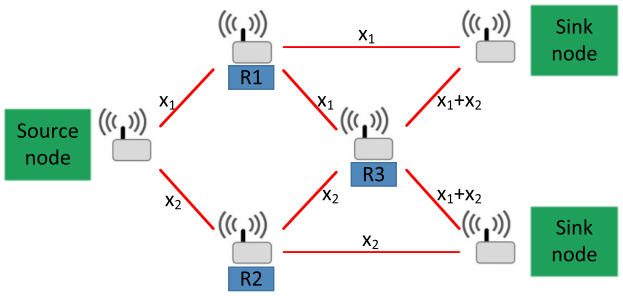


FIGURE 2. A simple example of network coding.

As an example, the authors in [23] have applied the principles of random network coding to the context of peer-to-peer (P2P) content distribution, and have shown that file downloading times can be reduced. Thus in this paper we propose to apply adaptive random linear network coding in the IoT core network to improve the network transmission efficiency. In [24], the authors develop an OpenCoding protocol to improve the network throughput through intra-flow network coding for wireless mesh networks.

2) DISTRIBUTED CLOUD STORAGE UTILIZING NETWORK CODING

When a storage node in the distributed cloud storage network that employing (n, k) RS code (such as Total Recall [19]) fails, the replacement node connects to k nodes and downloads the data of the same amount as the whole file first to decode the original file. Then the replacement node encodes the original file using the same (n, k) code to recover the encoded part of the file stored in the failed node. This approach is a waste of bandwidth because the whole file has to be downloaded to recover a fraction of it.

To overcome this drawback, Dimakis *et al.* [25] introduced the conception of $(n, k, d, \alpha, \beta, B)$ regenerating code based on the network coding. In the context of regenerating code, the contents stored in a failed node can be regenerated by the replacement node through downloading β help symbols from each of d helper nodes. This regeneration is identical to the encoding process of the intermediate nodes in network coding. The bandwidth consumption for the failed node regeneration could be far less than the whole file. *Thus the energy consumption in data regeneration could be greatly reduced.*

In [25], a tradeoff between the regeneration bandwidth $\gamma = d\beta$ and the storage requirement α was derived based on network coding theory and two extreme points were found: minimum storage regeneration (MSR) point in which the storage parameter α is minimized:

$$(\alpha_{MSR}, \gamma_{MSR}) = \left(\frac{B}{k}, \frac{Bd}{k(d-k+1)}\right), \tag{2}$$

and minimum bandwidth regeneration (MBR) point in which the bandwidth γ is minimized:

$$(\alpha_{MBR}, \gamma_{MBR}) = \left(\frac{2Bd}{2kd-k^2+k}, \frac{2Bd}{2kd-k^2+k}\right). \tag{3}$$

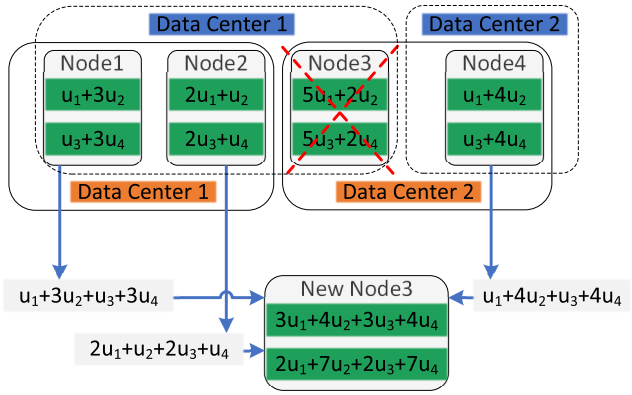


FIGURE 3. network coding based distributed cloud storage.

Fig. 3 is an illustrative example of regenerating code with parameters $n = 4, k = 2, d = 3, \alpha = 2, \beta = 1, B = 4$. 4 symbols u_1, u_2, u_3, u_4 are stored in 4 storage nodes, and can be retrieved from any 2 of the storage nodes. A failed node can be regenerated by downloading 1 symbol each from the 3 remaining nodes. Here we suppose node 3 fails. For the storage systems simply employing RS code, 4 symbols have to be downloaded first to decode the original symbols. Then we have to encode the 4 decoded symbols again to regenerate the symbols in the failed node 3. So the bandwidth needed for repairing the failed node 3 is 4. For the regenerating code solution in Fig. 3, by linearly combing the 3 downloaded symbols $u_1 + 3u_2 + u_3 + 3u_4$, $2u_1 + u_2 + 2u_3 + u_4$ and $u_1 + 4u_2 + u_3 + 4u_4$ into 2 symbols $3u_1 + 4u_2 + 3u_3 + 4u_4$ and $2u_1 + 7u_2 + 2u_3 + 7u_4$, we can regenerate a new node 3 that has the same function as the failed node 3. In the repairing process, only 3 symbols need to be downloaded. Thus the repair bandwidth is saved by 25%. In the later section, we will introduce a storage allocation problem for regenerating code in the distributed cloud storage and propose an optimal storage allocation scheme that can achieve the highest possible reliability.

III. ADAPTIVE NETWORK CODING IN THE IoT CORE NETWORK

In this section, we will show our adaptive network coding (ANC) scheme in the IoT core network. The size of the data to be transmitted in the IoT core network may be larger than the size limit of a single packet, such as new firmwares to update the smart devices on-air. So the data needs to be divided into data fragments first then transmitted in multiple packets with one data fragment per packet. A node has to correctly receive enough linearly independent packets to reassemble the original data.

A. LIMITATIONS OF EXISTING WORKS

Since the communication between smart devices are through wireless channel and there may be various fadings and interferences in the channel, some of the nodes may experience packets loss in the communication. When network coding is not utilized, retransmission is a common method to mitigate the packets loss. In some cases, certain packets may get lost most of the time so these packets have to be retransmitted many times until they are correctly received. Thus the overall transmission efficiency will be low. *Here the transmission efficiency is defined as the ratio between the minimum number of the packets needed to reassemble the original data and the number of total packets transmitted from the source node and the intermediate nodes*. When network coding is utilized, a node can retrieve the original data as long as the node can correctly receive enough number of packets. The entire transmission of the data will not be affected by lacking of certain particular packets. So the overall transmission efficiency will be higher.

However, there are still limitations for simply applying the network coding in the IoT core network, where the number of encoded packets to be generated and sent in the intermediate nodes is predetermined [18]. If too few packets are generated, the sink node may not even be able to collect enough packets to decode the original data. If too many packets are generated, the transmission efficiency will be low. Moreover, the fact that the quality of wireless channel is changing over time makes the situation even worse. As an example, when the channel quality becomes better and the packet loss rate goes lower, some of the encoded packets will be useless and the transmission efficiency could be higher. The encoding strategy should be able to dynamically adjusted according to the transmission conditions.

B. ANC SCHEME FOR THE IoT CORE NETWORK

To overcome the limitations mentioned above, we propose an adaptive network coding (ANC) scheme to further improve the transmission efficiency of the IoT core network with SDWN, as illustrated in Fig. 4. In this example, end users could communicate with the IoT core network with SDWN through Internet for monitoring/control purposes. The data transmission inside the IoT core network could benefit from our ANC scheme. In the figure, the IoT core network formed by the smart devices could be the smart appliance network at home, the surveillance camera network on streets or the emission detecting network in factories, etc. Here we only include the data plane and control plane of the SWDN to show the main idea of the ANC scheme.

For the data plane, the source node will send out linear combinations of the original packets. Each intermediate node will perform random linear network coding. The incoming packets will be linearly combined using random coefficients then sent out to succeeding nodes. *The code rate r of the network coding is defined as the ratio of the number of encoded packets to the number of incoming packets.* And the code rates of the network coding will be automatically adjusted by the SWDN controller mentioned below. The sink nodes will decode the original packets after receiving enough number of linearly independent packets.

Meanwhile, for the control plane, the smart devices will report packets receiving statistics to the SDWN controller periodically through the control path. Based on the informa-

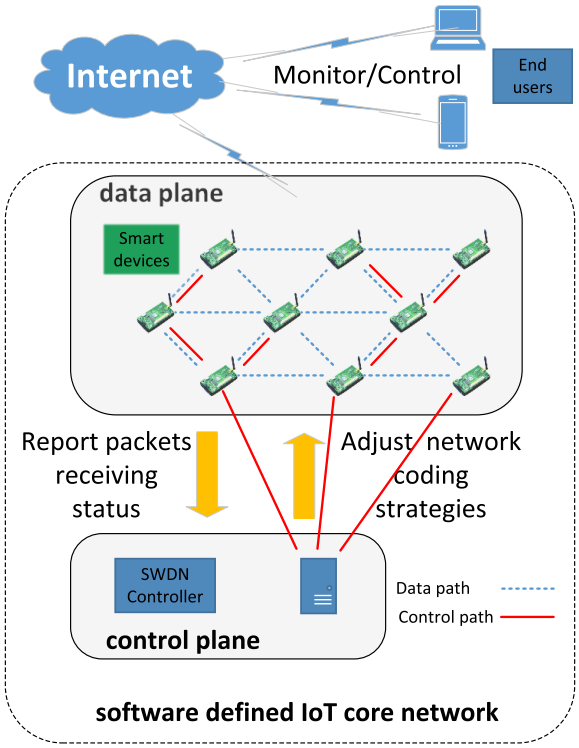


FIGURE 4. ANC Scheme for the IoT core network.

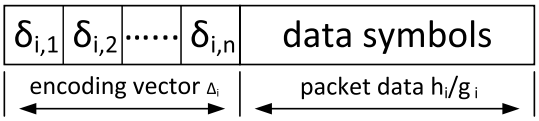


FIGURE 5. ANC packet format.

Algorithm 1 ANC Scheme - Source Node

▷ the network coding code rate r is determined/updated by the SWDN controller
for $i = 1 \rightarrow \lceil rn \rceil$ **do**
  if $i \leq n$ **then**
    repeat generate a random encoding vector $\mathbf{\Delta}_i = [\delta_{i,1}, \delta_{i,2}, \ldots, \delta_{i,n}]$
    until $\mathbf{\Delta}_i$ is linearly independent from all the $\mathbf{\Delta}_j$, $1 \leq j < i$ (except for $i = 1$)
  else
    generate a random encoding vector $\mathbf{\Delta}_i = [\delta_{i,1}, \delta_{i,2}, \ldots, \delta_{i,n}]$
  end if
  ▷ multiply each symbol of packet data $\mathbf{h}_j$ by $\delta_{i,j}$
  $\mathbf{g}_i \Leftarrow \sum_{j=1}^{n} \delta_{i,j}\mathbf{h}_j$
  send out $[\mathbf{\Delta}_i || \mathbf{g}_i]$, where "||" is the concatenation operation
end for
report the number of sent out packets to the SWDN controller

tion reported, the SDWN controller will dynamically adjust the network coding strategies to eliminate unnecessary transmissions. If the packet loss becomes higher around some node, more encoded packets will be generated in the corresponding intermediate nodes. If the packet loss becomes lower, the number of encoded packets will be decreased. Since the SDWN controller has the global information of the network, this centralized control will be more effective.

1) SOURCE NODE ALGORITHM OF THE ANC SCHEME

In the source node, the data to be transmitted will be fragmented into data packets with equal length. Every n data packets will form a coding group, in which random linear network coding will be performed. For the purpose of clarity, in the paper we assume that there is only one coding group. For each packet $\mathbf{h}_i$ in the coding group, there will be an encoding vector $\mathbf{\Delta}_i = [\delta_{i,1}, \delta_{i,2}, \ldots, \delta_{i,n}]$ ($\delta_{i,j} \in GF(2^m)$, $1 \leq i, j \leq n$) attached in front to indicate which packets participate in the encoding of $\mathbf{h}_i$. $GF(2^m)$ denotes the finite field with 2^m elements where $m \in \{8, 16, 32, 64, \ldots\}$ is determined by the symbol size. For an uncoded packet $\mathbf{h}_i$, the elements in the encoding vector will be all-zero except $\delta_{i,i} = 1$. The packet format is illustrated in Figure 5. The source node will perform Algorithm 1 to send out the encoded packets $\mathbf{g}_i$ ($1 \leq i \leq \lceil rn \rceil$) where r is the code rate determined by the SWDN controller and $\lceil rn \rceil$ is the ceiling operation to get the smallest integer that is larger than or equal to rn.

In algorithm 1, the source node generates $\lceil rn \rceil$ n-dimensional encoding vectors $\mathbf{\Delta}_i$ (first n encoding vectors are linearly independent) and uses the vector elements $\delta_{i,1}, \delta_{i,2}, \ldots, \delta_{i,n}$ as coefficients to generate and send out encoded packets from the uncoded packets $\mathbf{h}_1, \ldots, \mathbf{h}_n$.

2) INTERMEDIATE NODE ALGORITHM OF THE ANC SCHEME

For each coding group, the intermediate node will open a receiving buffer to store the incoming fresh packets from the nodes designated by the SDWN controller for encoding. The intermediate node will also record all the encoding vectors received in the incoming packets. *A packet is called a fresh packet if its encoding vector is linearly independent from all of the previously received packets'*. In order to get a trade-off between the packet diversity and communication delay, the intermediate node will encode the incoming fresh packets received during a preset interval τ which is measured by a timer then clear the receiving buffer and wait for the next incoming fresh packet to restart the timer and the buffering. At the end of each time interval, the encoding of the fresh packets in the receiving buffer is performed. For better illustration, we can split each of the n_τ fresh packets in the receiving buffer into the encoding vector $\mathbf{\Delta}_i$ and data $\mathbf{g}_i$ ($1 \leq i \leq n_\tau$). n_τ is the number of fresh packets in the receiving buffer. The intermediate node will send out $\lceil rn_\tau \rceil$ encoded packets using Algorithm 2, where r is the code rate determined by the SWDN controller. At the same time, the intermediate node will report the receiving and the sending of the packets to the SWDN controller.

Algorithm 2 ANC Scheme - Intermediate Node

▷ the network coding code rate r is determined/updated by the SWDN controller
for $i = 1 \rightarrow \lceil rn_\tau \rceil$ **do**
  if $i \leq n_\tau$ **then**
    repeat generate a random vector $\mathbf{R}_i = [r_{i,1}, r_{i,2}, \ldots, r_{i,n_\tau}]$
    until $\mathbf{R}_i$ is linearly independent from all the $\mathbf{R}_j$, $1 \leq j < i$ (except for $i = 1$)
  else
    generate a random vector $\mathbf{R}_i = [r_{i,1}, r_{i,2}, \ldots, r_{i,n_\tau}]$
  end if
  ▷ multiply each symbol of $\mathbf{\Delta}_j$ by $r_{i,j}$
  $\mathbf{\Delta}'_i \Leftarrow \sum_{j=1}^{n_\tau} r_{i,j}\mathbf{\Delta}_j$
  ▷ multiply each symbol of $\mathbf{g}_j$ by $r_{i,j}$
  $\mathbf{g}'_i \Leftarrow \sum_{j=1}^{n_\tau} r_{i,j}\mathbf{g}_j$
  send out $[\mathbf{\Delta}'_i || \mathbf{g}'_i]$, where "||" is the concatenation operation
end for
report the number of received packets from each of the other nodes/the number of sent out packets to the SWDN controller

In algorithm 2, the intermediate node generates $\lceil rn_\tau \rceil$ n_τ-dimensional vectors $\mathbf{R}_i$ (first n_τ vectors are linearly independent) and uses the vector elements $r_{i,1}, r_{i,2}, \ldots, r_{i,n_\tau}$ as coefficients to generate and send out recoded packets from the received packets $\mathbf{g}_1, \ldots, \mathbf{g}_{n_\tau}$. The corresponding encoding vectors are processed the same way.

3) SINK NODE ALGORITHM OF THE ANC SCHEME

Once the sink node receives n linearly independent packets, it can solve the following equation to decode the original packets data $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n$:

$$\begin{bmatrix} \mathbf{\Delta}_1 \\ \mathbf{\Delta}_2 \\ \vdots \\ \mathbf{\Delta}_n \end{bmatrix} \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \vdots \\ \mathbf{h}_n \end{bmatrix} = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_n \end{bmatrix}. \tag{4}$$

Then $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n$ can be concatenated to restore the original data. The sink node will also periodically report the packets receiving status to the SWDN controller.

4) SDWN CONTROLLER ALGORITHM OF THE ANC SCHEME

Since the SDWN controller receives the packets sending/receiving status from each of the nodes in the IoT core network periodically , it can adjust the code rate of the network coding for each of the nodes according to Algorithm 3. Suppose η_i is the number of packets sent by node i, $\mathbb{N}_i$ is the set of succeeding nodes receiving the packets from node i, and $\eta_j^{(i)}$ is the number of packets received by node $j \in \mathbb{N}_i$. The code rate r_i of node i can be determined by

$$r_i = \frac{\eta_i}{\max_{j \in \mathbb{N}_i}(\eta_j^{(i)})}, \tag{5}$$

Algorithm 3 ANC Scheme - SDWN Controller

for each of the source node or intermediate node i **do**
  calculate r_i according to the packet sending/receiving status of node i and $\mathbb{N}_i$
  send r_i to update the code rate of node i through the control path
end for

where max() is the operation to select the maximum element. Besides the code rate, since the SDWN controller has all of the topology information, for each node i, it can specify the succeeding relay nodes to receive the packets sent from node i.

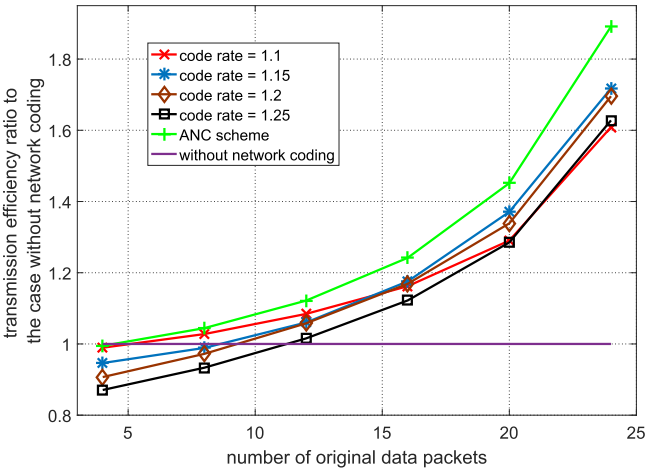


FIGURE 6. Performance of the ANC scheme.

C. PERFORMANCE EVALUATION OF THE ANC SCHEME

In Fig. 6 are the simulation results of the ANC scheme. The simulation is carried out in the NS-2 platform [26]. In the simulation, the leftmost node in Fig. 4 tries to send data to the rightmost two nodes. The qualities of the channels between the intermediate nodes are chosen randomly. We calculate the transmission efficiencies under different numbers of total original data packets (can be viewed as one coding group for network coding). For performance comparison, we also simulate the cases for no network coding (pure retransmission) and network coding with predetermined code rates as in [18]. To make the comparison more clear, we normalize the transmission efficiencies for the cases with fixed network coding and the ANC scheme by the transmission efficiency for the case without network coding. From the simulation results, we can see that the transmission efficiency for the case with network coding becomes higher than the case without network coding with the increasing of the number of original data packets. And the ANC performs best among all the cases. It can also be seen that the performance gain of the ANC scheme will increase when the number of total original data packets becomes larger.

IV. OPTIMAL STORAGE ALLOCATION IN THE DISTRIBUTED CLOUD STORAGE UTILIZING NETWORK CODING

In this section we first introduce a storage allocation problem for the distributed cloud storage utilizing network coding. Then we propose the optimal storage allocation (OSA) scheme. We also show the performance of the optimal storage allocation scheme.

A. STORAGE ALLOCATION PROBLEM

In the storage allocation problem $\mathbb{S}$, the data is encoded with an $(n, k, d, \alpha, \beta, B)$ regenerating code, so there will be n encoded parts. There are N ($N < n$) data centers in total to store these parts, each with a failure probability of p. If a data center fails, all the data stored in the data center will be lost. If the total number of encoded parts in the remaining data centers is less than k, the original data cannot be recovered any more. Since there are more encoded parts than the data centers, there will be different allocation strategies of the encoded parts with different storage reliabilities. For the problem $\mathbb{S}$, we try to find out the allocation strategy with the lowest failure probability among all the possible allocation strategies.

Definition 1: A set S with N elements $n_1, n_2, \ldots, n_N$ ($n_i > 0, 1 \le i \le N$) *is a valid allocation if* $\sum_{i=1}^{N} n_i = n$.

Definition 2: For an allocation strategy S, the failure probability P is defined as the probability that the original data cannot be recovered given the failure probabilities of individual data centers.

The problem $\mathbb{S}$ can be formulated as:

$$\text{find the allocation S among all the valid allocations,}$$
$$\text{such that } \sum_{\forall S_j \subseteq S} P\left(\sum_{n_i \in S_j} n_i \ge n-k\right) \text{ is minimal.} \quad (6)$$

As an example, for the regenerating code in Fig. 3, $n = 4$ encoded parts are stored in $N = 2$ data centers. Suppose the failure probability of each data center is $p = 0.01$. Two storage allocation strategies are shown in the figure. For the first allocation strategy $S = \{3, 1\}$ (blue data centers with dash lines), 3 encoded parts are stored in data center 1 and 1 encode part is stored in data center 2. It is easy to calculate the failure probability of this allocation strategy is 0.01. For the second allocation strategy $S = \{2, 2\}$ (orange data centers with solid lines), 2 encoded parts are stored in each of the two data centers. The failure probability of this allocation strategy is 0.0001, which is much lower than that of the first strategy.

B. OPTIMAL STORAGE ALLOCATION SCHEME

In this section, we will show our optimal storage allocation (OSA) scheme to solve the storage allocation problem. The OSA scheme includes two stages: the first stage is to find out all the possible valid allocations S and the second stage is to calculate the failure probability P for each S. Then we can output the allocation with the lowest failure probability through comparison.

1) STAGE I: FIND OUT ALL THE POSSIBLE VALID ALLOCATIONS S

The naive approach to find out all the possible valid S is to search all the possible combinations of $n_1, n_2, \ldots, n_N$ such that $\sum_{i=1}^{N} n_i = n$. However, this approach will take exponential time thus is not practical. In our OSA scheme, we first change this problem into an integer partition problem [27]: to allocate n encoded parts into N storage centers is the same as to partition an integer n into N parts. Take $n = 7, N = 3$ as an example, there are 4 ways to partition 7 into 3 parts: $\{1, 1, 5\}$, $\{1, 2, 4\}$, $\{1, 3, 3\}$ and $\{2, 2, 3\}$, which also consist all the possible valid allocations. Then we can solve the integer partition problem using dynamic programming based on the following recurrence equation:

$$\mathbb{P}(n, N) = \mathbb{P}(n-1, N-1) + \mathbb{P}(n-N, N), \quad (7)$$

where $\mathbb{P}(i, j)$ is the total number of ways of partitioning integer i into j parts. The first part of equation (7) is the subproblem where at least one 1 exists in the partition and the second part of the equation is the subproblem where no 1 exists in the partition. Thus the solution to the original problem perfectly incorporates these two subproblems, which make it feasible to solve using dynamic programming. We propose Algorithm 4 to find out all the possible valid allocations S. In the algorithm, we use $S(i, j, k)$ to represent the k^{th} valid allocation out of the $\mathbb{P}(i, j)$ allocations for i encoded parts and j storage centers. $\cup$ is the union operation between two sets. The addition between a set S and a number x is defined as the additions between every element of the set and the number:

$$S + x := \{n_i + x | n_i \in S \text{ for } 1 \le i \le N\}. \quad (8)$$

After the execution of the algorithm, we can get all the possible valid allocations $S(n, N, k)$ ($1 \le k \le \mathbb{P}(n, N)$). It is easy to see that the algorithm runs in polynomial time.

Theorem 1: Algorithm 4 can output all the valid allocations $S(n, N, l)$ *for* $1 \le l \le \mathbb{P}(n, N)$, *where* $S(n, N, l)$ *represents the* l^{th} *valid allocation out of the* $\mathbb{P}(n, N)$ *allocations for n encoded parts and N storage centers.*

Proof: Algorithm 4 calculates $S(i, j, l)$ ($1 \le l \le \mathbb{P}(i, j)$) for $1 \le j \le N$ from $i = 1$ to $i = n$ through a bottom-up manner and we can get $S(n, N, l)$ ($1 \le l \le \mathbb{P}(n, N)$) for $i = n, j = N$. For each i, line 3 to line 4 first calculate $\mathbb{P}(i, 1) = 1$ and $S(i, 1, 1) = i$, corresponding to the case of allocating i encoded data parts into one data center. Then for each $j = 2, \ldots, N$, there will be two cases:

- Line 8 to line 9 correspond to the case with $i - j < j$, where at least one storage node will be allocated only 1 encoded data part. The second part of equation (7) does not exist. So the number of ways of allocating i encoded data parts into j storage nodes will be equal to that of allocating $i - 1$ encoded data parts into $j - 1$ storage nodes: $\mathbb{P}(i, j) = \mathbb{P}(i-1, j-1)$. And each of the valid allocations $S(i, j, l)$ will be the union of each

Algorithm 4 OSA Scheme - Stage I

Input: the number of encoded parts n and the number of storage centers N

Output: all the valid allocations $S(n, N, l)$, $(1 \leq l \leq \mathbb{P}(n, N))$

1: **function** FindAllAllocations(n, N)
2: **for** $i = 1 \rightarrow n$ **do**
3: $\mathbb{P}(i, 1) \Leftarrow 1$
4: $S(i, 1, 1) \Leftarrow i$
5: **for** $j = 2 \rightarrow N$ **do**
6: **if** $i \geq j$ **then**
7: **if** $i - j < j$ **then**
8: $\mathbb{P}(i, j) \Leftarrow \mathbb{P}(i-1, j-1)$
9: $S(i, j, l) \Leftarrow S(i-1, j-1, l) \cup \{1\}$, for $1 \leq l \leq \mathbb{P}(i, j)$
10: **else**
11: $\mathbb{P}(i, j) \Leftarrow \mathbb{P}(i-1, j-1) + \mathbb{P}(i-j, j)$
12: $S(i, j, l) \Leftarrow S(i-1, j-1, l) \cup \{1\}$, for all $1 \leq l \leq \mathbb{P}(i-1, j-1)$
13: $S(i, j, \mathbb{P}(i-1, j-1) + l) \Leftarrow S(i-j, j, l)+$ 1, for all $1 \leq l \leq \mathbb{P}(i-j, j)$
14: **end if**
15: **end if**
16: **end for**
17: **end for**
18: **end function**

already calculated allocations $S(i-1, j-1, l)$ with the set $\{1\}$.

- Line 11 to line 13 correspond to the case with $i - j \geq j$, where $\mathbb{P}(i, j)$ is the summation of two previously calculated parts as shown in equation (7). The computation of the first part and the corresponding valid allocations is the same as in line 8 to line 9. The second part is the number of ways of allocating $i - j$ encoded data parts into j storage nodes $\mathbb{P}(i-j, j)$, where each of the storage node will be allocated at least 2 encoded data parts. Thus each of the valid allocations $S(i, j, \mathbb{P}(i-1, j-1) + l)$ will be each of the already calculated allocations $S(i-j, j, l)$ plus 1 as defined in equation (8).

□

Fig. 7 illustrates the algorithm for $n = 7$ encoded data parts and $N = 3$ data centers. Each (i, j) pair represent the calculation of $\mathbb{P}(i, j)$ and $S(i, j, l)$. The pairs without shades are calculated using line 8 to line 9 (the first case) while the pairs in shades are calculated using line 11 to line 13 (the second case). The solid lines correspond to the first part of equation (7) and the dashed lines correspond to the second part. From the figure we can clearly see that (7, 3) can be efficiently calculated using the results of (6, 2) and (4, 3), which have already been calculated the same way as illustrated in Fig. 7.

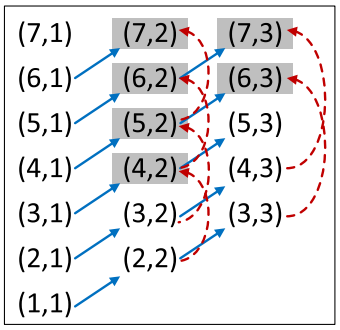


FIGURE 7. The calculation of Algorithm 4 for $n = 7$, $N = 3$.

2) STAGE II: CALCULATE THE FAILURE PROBABILITY P FOR EACH VALID ALLOCATION S

After we get all the possible valid allocations S, we can calculate the failure probability P_S for each of them. The goal function of equation (6) can be further written as:

$$P_S = \sum_{\forall S_j \subseteq S} P\left(\sum_{n_i \in S_j} n_i > n-k\right) = \sum_{\forall S_j \subseteq S,\ s.t.\ \sum_{n_i \in S_j} n_i > n-k} p^{|S_j|}(1-p)^{N-|S_j|}, \quad (9)$$

where p is the failure probability of each storage center, $|S_j|$ is the number of elements in subset S_j. If we try to directly calculate P_S for every subset $S_j \in S$, the order of the number of subsets to be calculated will be approximate to $\sum_{|S_j|=1}^{N} \binom{N}{|S_j|} \approx 2^N$, where $\binom{N}{|S_j|}$ denotes the number of $|S_j|$-combinations of the set S, thus making it infeasible to calculate in practice.

In the second stage of the OSA scheme (Algorithm 5), we propose to change the exhaust search problem into a number counting problem. More specifically, for each i $(1 \leq i \leq N)$, we count the total number of subsets $S_j^{(i)}$ such that $S_j^{(i)}$ denotes the subsets with exactly i elements and the summation of every element in $S_j^{(i)}$ is larger than $n-k$:

$$P_S = \sum_{i=1}^{N} \left| \left\{ S_j^{(i)} \mid \sum_{n_i \in S_j^{(i)}} n_i > n-k \right\} \right| p^i (1-p)^{N-i}. \quad (10)$$

In Algorithm 5, we first calculate the summations of every subset, which can be viewed as a variant of the subset-sum problem [28]. For each i $(1 \leq i \leq N)$, we merge the same-value summation results of the subsets $S_j^{(i)}$ and count the total number of $S_j^{(i)}$ which have that summation value. Then for the subsets that have summation results larger than $n-k$, we can calculate the corresponding failure probability according to equation (10). In the algorithm, T, L, R represent three auxiliary lists for subset summation. For a auxiliary list X, we use X.length to denote the number of elements of the list, X.index to denote the current index number of the list,

Algorithm 5 OSA Scheme - Stage II

Input: a valid allocation $S(n, N, l)$, $(1 \le l \le \mathbb{P}(n, N))$
Output: the failure probability P_S of the allocation

1: **function** CalculateProbability($S(n, N, l)$)
2: $\{n_1, n_2, \ldots, n_N\} \Leftarrow$ sort the allocation $S(n, N, l)$ in non-descending order
3: $L \Leftarrow \{0\}$
4: ▷ calculate summations of every subset
5: **for** $i = 1 \rightarrow N$ **do**
6: $T \Leftarrow \phi$
7: $R \Leftarrow L + n_i$, $C_R(1, 1) \Leftarrow 1$
8: $C_R(l, j) \Leftarrow C_L(l-1, j)$, for all nonzero $C_L(l, j)$, $2 \le j \le L.\mathsf{length}$, $2 \le l \le i$, $i \ge 2$
9: $L.\mathsf{index}, R.\mathsf{index} \Leftarrow 1$
10: **while** $L.\mathsf{index} \le L.\mathsf{length}$ **do**
11: **if** $V_L(L.\mathsf{index}) == V_R(R.\mathsf{index})$ **then**
12: $T \Leftarrow T \cup V_R(R.\mathsf{index})$
13: for all $1 \le l \le i$, $C_T(l, T.\mathsf{length}) \Leftarrow C_L(l, L.\mathsf{index}) + C_R(l, R.\mathsf{index})$
14: increase $L.\mathsf{index}, R.\mathsf{index}$ by 1
15: **else**
16: **if** $V_L(L.\mathsf{index}) < V_R(R.\mathsf{index})$ **then**
17: $T \Leftarrow T \cup V_L(L.\mathsf{index})$
18: $C_T(l, T.\mathsf{length}) \Leftarrow C_L(l, L.\mathsf{index})$, for all $1 \le l \le i$
19: increase $L.\mathsf{index}$ by 1
20: **else**
21: $T \Leftarrow T \cup V_R(R.\mathsf{index})$
22: $C_T(l, T.\mathsf{length}) \Leftarrow C_R(l, R.\mathsf{index})$, for all $1 \le l \le i$
23: increase $R.\mathsf{index}$ by 1
24: **end if**
25: **end if**
26: **end while**
27: $oldLn \Leftarrow T.\mathsf{length}$
28: $T \Leftarrow T \cup \{V_R(R.\mathsf{index}), V_R(R.\mathsf{index}+1), \ldots, V_R(R.\mathsf{length})\}$
29: $\{C_T(l, oldLn+1), \ldots, C_T(l, T.\mathsf{length})\} \Leftarrow \{C_R(l, R.\mathsf{index}), \ldots, C_R(l, R.\mathsf{length})\}$, $1 \le l \le i$
30: $L \Leftarrow T$
31: **end for**
32: $P_S \Leftarrow 0$
33: ▷ count the number of subsets with the summation results larger than $n-k$
34: **for** $i = 1 \rightarrow N$ **do**
35: $sum \Leftarrow 0$
36: **for** $j = 1 \rightarrow T.\mathsf{length}$ **do**
37: **if** $V_T(j) > n-k$ **then**
38: $sum \Leftarrow sum + C_T(i, j)$
39: **end if**
40: **end for**
41: $P_S \Leftarrow P_S + sum \times p^i(1-p)^{N-i}$
42: **end for**
43: **end function**

$V_X(j)$ to denote the value of j^{th} element in X, and $C_X(i, j)$ to denote the total number of subsets that have the same element number i and the same summation value $V_X(j)$. Although the total number of subsets is 2^N, Algorithm 5 is a polynomial time algorithm:

Theorem 2: The complexity of Algorithm 5 is $\mathcal{O}(nN)$.

Proof: Since the summation of a valid allocation S itself is the largest in all the summations of the subsets of S, the element number $T.\mathsf{length}$ in T cannot exceed n. Through the merge of subsets with the same summation values, each of the N for-loops has the complexity $\mathcal{O}(n)$. So the total complexity is $\mathcal{O}(nN)$. □

Theorem 3: Algorithm 5 can output the failure probability P_S for the input allocation.

Proof: In line 3 we initialize the auxiliary list L with an empty element $'0'$, representing the summation result of 0 element of the input allocation S. Line 4 to line 31 calculate the summations of every subset of the input allocation. At the beginning of each round i of the for loop $i = 1, \ldots, n$, the auxiliary list L is the list containing the summation results of every subset of the first l $(0 \le l < i)$ elements of the input allocation S. Line 7 to line 8 calculate the auxiliary list R by adding the new element n_i to L: $R = L + n_i$. Since the first element in L is the empty $'0'$, $C_R(1, 1)$ will be 1, indicating that the total number of subsets that have 1 element and summation value n_i is 1. Then the rest value of $C_R(l, j)$ will be $C_L(l-1, j)$ for $2 \le j \le L.\mathsf{length}$ because of the addition of n_i to L. The elements of allocation S are sorted in non-descending order, thus the elements in both L and R are also in non-descending order. From line 10 to line 29, we merge the elements of the auxiliary lists L and R into a temporary auxiliary list T one by one, following the rules below:

- If the value of the current element $V_L(L.\mathsf{index})$ in L is equal to the current element $V_R(R.\mathsf{index})$ in R, add the value into T. The corresponding counter $C_T(l, T.\mathsf{length})$ is equal to the sum of the two counters: $C_T(l, T.\mathsf{length}) = C_L(l, L.\mathsf{index}) + C_R(l, R.\mathsf{index})$ for $1 \le l \le i$.
- If the value of the current element $V_L(L.\mathsf{index})$ in L is smaller than the current element in R, add the element $V_L(L.\mathsf{index})$ into T. Set the counter $C_T(l, T.\mathsf{length})$ to $C_L(l, L.\mathsf{index})$ for $1 \le l \le i$.
- If the value of the current element $V_R(R.\mathsf{index})$ in R is smaller than the current element in L, add the element $V_R(R.\mathsf{index})$ into T. Set the counter $C_T(l, T.\mathsf{length})$ to $C_R(l, R.\mathsf{index})$ for $1 \le l \le i$.
- Since the last element in L is smaller than some elements in R, after merging L into T, we can directly merge the remaining elements of R into T through line 28 to line 29.

At the end of each for loop, the merged list T is assigned back to L for the next round of calculation. After N^{th} round, list T has the summation results of all the subsets in S.

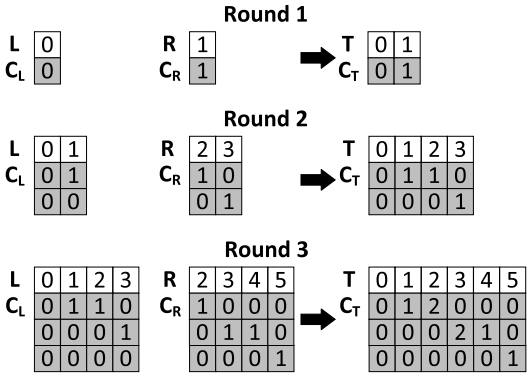


FIGURE 8. The calculation of Algorithm 5 for $S = \{1, 2, 2\}$.

Then the failure probability of S can be easily calculated from line 34 to line 42 by counting the number of subsets with the summation results larger than $n - k$. □

Fig. 8 illustrates the summations for all the subsets of $S = \{1, 2, 2\}$. For $i = 1$, $L = \{0\}$, $C_L(1, 1) = 0$, $R = \{1\}$, $C_R(1, 1) = 1$. The merged list $T = \{0, 1\}$, $C_T = \{0, 1\}$. For the second round, L, C_L are assigned the values of T, C_T. According to line 7 and line 8 of Algorithm 5, $R = L + n_2 = \{2, 3\}$ and $C_R(2, 2) = C_L(1, 2) = 1$. At the end of the third round, we can get the summation results $T = \{1, 2, 3, 4, 5\}$ and the counter matrix C_T, which correctly record the number of subsets that have the same summation value. As an example, $C_T(2, 3) = 2$, indicating that there are two 2-element subsets ($\{n_1 = 1, n_2 = 2\}, \{n_1 = 1, n_3 = 2\}$) that have the same summation value $V_T(3) = 3$.

3) OSA SCHEME

Based on the algorithms of the two stages, we can achieve the optimal storage allocation through Algorithm 6. And it is straightforward to see:

Theorem 4: The OSA scheme is a polynomial time algorithm.

Algorithm 6 OSA Scheme

Input: the number of encoded parts n and the number of storage centers N

Output: the allocation with the lowest failure probability

function OSA(n, N)
  $S(n, N, l) \Leftarrow$ FindAllAllocations(n, N) ($1 \le l \le \mathbb{P}(n, N)$)
  for $l = 1 \rightarrow \mathbb{P}(n, N)$ **do**
    $P_S \Leftarrow$ CalculateProbability($S(n, N, l)$)
  end for
  output the allocation with the lowest P_S
end function

C. SIMULATION RESULTS FOR THE OSA SCHEME

In this section, we will show the performance of the OSA scheme for the given regenerating code with parameters $(n, k, d, \alpha, \beta, B)$ and number of data centers N.

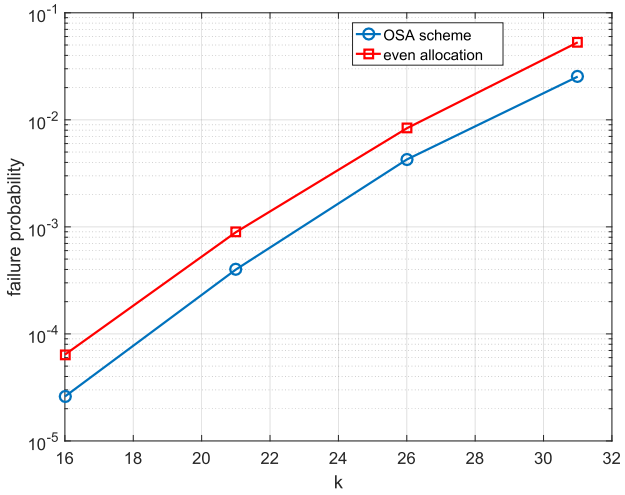


FIGURE 9. Performance of the optimal storage allocation for different k.

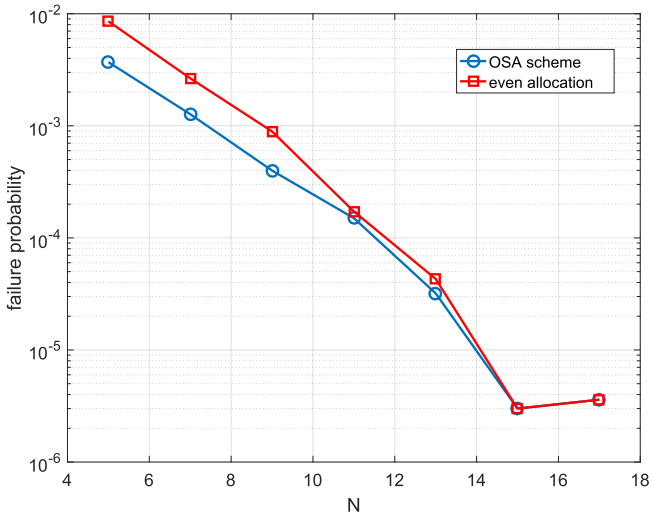


FIGURE 10. Performance of the optimal storage allocation for different number of storage centers.

In Fig. 9 are the simulation results for $n = 45$, $k = \{16, 21, 26, 31\}$, $N = 9$ and $p = 0.1$. For performance comparison, we also plot the results for the even allocation as defined in equation (11), where $\lfloor n/N \rfloor$ is the floor operation to get the largest integer that is less or equal to n/N, $mod(n, N)$ is the modulo operation to find the remainder of the division of n by N.

$$n_i = \begin{cases} \lfloor n/N \rfloor + 1, & 1 \le i \le mod(n, N) \\ \lfloor n/N \rfloor, & mod(n, N) < i < N \end{cases} \tag{11}$$

The even allocation is a natural allocation scheme to store equal number of data blocks into each storage center. From the figure we can see that the failure probability of the OSA scheme is about half order of magnitude lower than the even allocation. Both of the probabilities will become higher when k increases since there is less redundancy in the distributed cloud storage to recover the failed storage centers.

In Fig. 10 are the simulation results for $n = 45$, $k = 21$, $p = 0.1$. In this simulation, we change the number of storage

centers N to study its impact to the failure probability. From the figure we can see that the failure probability will become lower when the number of storage centers increases. And the performance gap of the even allocation and the OSA scheme will diminish with the increasing of the number of storage centers.

V. CONCLUSION

Applying the network coding in Internet of things could save energy and contribute to the 'green IoT networking'. Motivated by this advantage, in this paper we optimize network coding based communication and reliable storage in Internet of things. We propose an adaptive network coding (ANC) scheme in the IoT core network with software defined wireless network (SDWN). Simulation results have demonstrated that the ANC scheme can achieve higher transmission efficiency than existing schemes. Then we introduce the optimal storage allocation problem for the distributed cloud storage that utilizes network coding, which stores the data generated by the IoT core network. we propose an optimal storage allocation (OSA) scheme to solve the problem in polynomial time. We also conduct simulations to show that the OSA scheme can greatly improve the storage reliability.

REFERENCES

[1] H. Yue, L. Guo, R. Li, H. Asaeda, and Y. Fang, "DataClouds: Enabling community-based data-centric services over the Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 5, pp. 472–482, Oct. 2014.
[2] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, to be published.
[3] J. Baliga, R. W. A. Ayre, K. Hinton, and R. S. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proc. IEEE*, vol. 99, no. 1, pp. 149–167, Jan. 2011.
[4] Y. Chen *et al.*, "Fundamental trade-offs on green wireless networks," *IEEE Commun. Mag.*, vol. 49, no. 6, pp. 30–37, Jun. 2011.
[5] G. Jia, G. Han, J. Jiang, and L. Liu, "Dynamic adaptive replacement policy in shared last-level cache of dram/pcm hybrid memory for big data storage," *IEEE Trans. Ind. Informat.*, to be published.
[6] Z. Zhou, Z. Cheng, L.-J. Zhang, W. Gaaloul, and K. Ning, "Scientific workflow clustering and recommendation leveraging layer hierarchical analysis," *IEEE Trans. Serv. Comput.*, to be published.
[7] J. Li, T. Li, and J. Ren, "Beyond the mds bound in distributed cloud storage," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 307–315.
[8] C. Wang, K. Ren, and J. Wang, "Secure optimization computation outsourcing in cloud computing: A case study of linear programming," *IEEE Trans. Comput.*, vol. 65, no. 1, pp. 216–229, Jan. 2016.
[9] B. Bangerter, S. Talwar, R. Arefi, and K. Stewart, "Networks and devices for the 5G era," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 90–96, Feb. 2014.
[10] (2016). *The Zigbee Alliance*. [Online]. Availbale: http://www.zigbee.org
[11] G. Mulligan, "The 6LoWPAN architecture," in *Proc. 4th Workshop Embedded Netw. Sensors (EmNets)*, Jun. 2007, pp. 78–82.
[12] G. Han, L. Liu, J. Jiang, L. Shu, and G. Hancke, "Analysis of energy-efficient connected target coverage algorithms for industrial wireless sensor networks," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 135–143, Feb. 2017.
[13] G. Han, J. Jiang, M. Guizani, and J. J. P. C. Rodrigues, "Green routing protocols for wireless multimedia sensor networks," *IEEE Wireless Commun.*, vol. 23, no. 6, pp. 140–146, Dec. 2016.
[14] Z. Zhou, W. Fang, J. Niu, L. Shu, and M. Mukherjee, "Energy-efficient event determination in underwater wsns leveraging practical data prediction," *IEEE Trans. Ind. Informat.*, to be published.
[15] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling SDNs," in *Proc. Eur. Workshop Softw. Defined Netw. (EWSDN)*, Oct. 2012, pp. 1–6.
[16] Z. Chen *et al.*, "A cloud computing based network monitoring and threat detection system for critical infrastructures," *Big Data Res.*, vol. 3, pp. 10–23, Apr. 2016.
[17] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, 3rd Quart., 2014.
[18] J. Hansen, D. E. Lucani, J. Krigslund, M. Médard, and F. H. P. Fitzek, "Network coded software defined networking: Enabling 5G transmission and storage networks," *IEEE Commun. Mag.*, vol. 53, no. 9, pp. 100–107, Sep. 2015.
[19] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *Proc. Symp. Netw. Syst. Design Implement.*, 2004, pp. 337–350.
[20] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1205–1216, Jul. 2000.
[21] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 782–795, Oct. 2003.
[22] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. R. Karger, "Byzantine modification detection in multicast networks using randomized network coding," in *Proc. Int. Symp. Inf. Theory (ISIT)*, Jul. 2004, p. 144.
[23] C. Gkantsidis and P. Rodriguez, "Cooperative security for network coding file distribution," in *Proc. IEEE INFOCOM*, Apr. 2006, pp. 1–13.
[24] D. Zhu, X. Yang, P. Zhao, and W. Yu, "Towards effective intra-flow network coding in software defined wireless mesh networks," in *Proc. 24th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2015, pp. 1–8.
[25] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.
[26] (2010). *Ns2 Wiki*. [Online]. Available: http://nsnam.isi.edu/nsnam/index.php/Main_Page
[27] A. Nijenhuis, H. S. Wilf, and W. Rheinboldt, *Combinatorial Algorithms for Computers and Calculators*, 2nd ed. San Diego, CA, USA: Academic, 1978.
[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.



JIAN LI received the B.S. and M.S. degrees from Tsinghua University, China, in 2005 and 2008, respectively, and the Ph.D. degree from Michigan State University, East Lansing, in 2015, all in electrical engineering. He is currently an Assistant Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University, China. His current research interests include network security, cloud storage, wireless sensor network in Internet of Things, privacy-preserving communications, and cognitive networks.



YUN LIU received the Ph.D. degree in electronic engineering from Beijing Jiaotong University, China. She is currently a Professor and the Director of the Department of Communication and Information System, and the Director of the Key Laboratory of Communication and Information Systems, Beijing Municipal Commission of Education, Beijing Jiaotong University. Her current research interests include telecommunication, computer networks, network security, intelligent transportation system, and social dynamics. She has edited many books and authored over 200 papers and book chapters, and participated in many international academic activities, including the organization of several international conferences. She is an IET fellow.



ZHENJIANG ZHANG received the Ph.D. degree in communication and information systems from Beijing Jiaotong University, Beijing, China, in 2008. Since 2014, he has been a Professor with the Department of Electronic and Information Engineering, Beijing Jiaotong University. He currently serves as the Assistant Dean of the School of Software Engineering, Beijing Jiaotong University. He has authored about 160 professional research papers. His research interests include wireless sensor network techniques, and wearable sensor network techniques, including multisource data fusion, security and privacy, routing, and energy management. He is a Guest Editor of *Journal of Electrical and Computer Engineering*.



JIAN REN (SM'15) received the B.S. and M.S. degrees in mathematics from Shaanxi Normal University, China, and the Ph.D. degree in electrical engineering from Xidian University, China. He is currently an Associate Professor with the Department of Electrical Communication Engineering, Michigan State University, East Lansing. His current research interests include cryptography, network security, energy efficient sensor network security protocol design, privacy-preserving communications, and cognitive networks. He received the U.S. National Science Foundation Faculty Early Career Development (CAREER) Award in 2009.



NAN ZHAO is currently an Associate Professor with Xidian University, China. His research area includes optical communication, quantum communication, and quantum information.