Computer Networks 56 (2012) 661–685

Contents lists available at SciVerse ScienceDirect

Computer Networks

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

A rule-based inter-session network coding scheme over IEEE 802.16(d) mesh CDS-mode networks

Shie-Yuan Wang *, Chih-Che Lin, Yu-Chi Chang

Department of Computer Science, National Chiao Tung University, 1001 University Road, Hsinchu, Taiwan

ARTICLE INFO

Article history:
Received 22 April 2010
Received in revised form 13 March 2011
Accepted 12 October 2011
Available online 31 October 2011

Keywords:
Network coding
802.16
Wireless mesh network
Hidden terminal
Inter-session network coding

ABSTRACT

In the literature, an opportunistic inter-session network coding scheme needs to exchange extra control messages to maintain traffic flow states to operate. However, such a requirement increases the implementation complexity of a network coding scheme. In this paper, we propose a rule-based network coding scheme (RNC) that performs opportunistic inter-session network coding using a stateless design. By exploiting this stateless design, the proposed coding scheme is easy to implement and deploy.

In addition, in this paper based on RNC we study the hidden terminal problems between different coding structures. Such a problem may result in severe packet collisions in a network-coding-based network and thus degrade network coding performance. To alleviate this problem, based on RNC we propose a smart handshake procedure (called RNC-SHP) over the IEEE 802.16(d) mesh coordinated distributed scheduling (CDS) mode to reduce the number of hidden terminals between pairwise network coding structures. Our simulation results show that the proposed RNC schemes can greatly outperform the original routing-based scheme on end-to-end flow goodputs and packet delays.

1. Introduction

Network coding is a packet-encoding–decoding mechanism that aims to increase the data transmission efficiency of a network. It was first proposed by Ahlswede et al. [1] and has gained much attention in recent years. A typical network coding mechanism is composed of two main operations: (1) mixing distinct outgoing packets on intermediate nodes; and (2) decoding mixed packets before (or when) they arrive at their destination nodes. With carefully finding and exploiting possible network coding opportunities, network coding can reduce the number of packet transmissions required for disseminating the same amount of data, as compared with traditional routing.

The development of network coding can be classified into two categories based on the traffic patterns that it intends to address. One deals with the traffic generated by one or more multicast flows (referred to as intra-session coding), while the other deals with the traffic generated by multiple unicast flows (referred to as inter-session coding). In this paper, we focus on the inter-session coding problem.

Developing an inter-session network coding scheme to effectively improve the performances of unicast flows is challenging. In a unicast-flow scenario, each flow has only one node as its intended receiver. In this condition, blindly coding packets may waste network bandwidth because packets may be disseminated to nodes that are not interested in them. For this reason, designing an efficient inter-session network coding scheme is difficult because it needs to consider three problems at the same time: (1) routing path selection; (2) traffic load balancing; and (3) network coding decision. In [2–5], the authors formulate the inter-session network coding problem as a resource optimization problem that considers the aforementioned three problems and use the linear programming technique to mathematically obtain the optimized solution to this problem.

* Corresponding author.
E-mail address: shieyuan@csie.nctu.edu.tw (S.-Y. Wang).

1389-1286/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2011.10.013

Such theoretical studies have explored the characteristics of the inter-session coding problem; however, they do not propose feasible distributed protocols for scheduling inter-session network coding across a network. Khreishah et al. [5] proposed a theoretical distributed framework to control inter-session coding over a virtualized wireless network model. However, their work focuses on networks with only 2 unicast flows and is linear-programming based. A linear-programming-based scheduling approach usually requires high computation complexity to find the optimal scheduling, which means that a distributed algorithm developed based on this approach needs long time to converge[1] (and may not converge under a highly-changing traffic pattern).

In addition, these previous studies assume that no packet collisions occur in a wireless network, i.e., they do not consider the performance degradation of an inter-session network coding scheme caused by the well-known "hidden terminal (HT) problem." The impacts of the HT problem can be discussed from two aspects: First, the packet collisions caused by the HT problem greatly reduce the number of packets that can be overheard by network nodes. Overhearing packets, however, is fundamental to the operation of network coding. Thus, the presence of HTs can greatly degrade the performance of network coding. Second, using virtual carrier-sensing mechanisms (e.g., the RTS-CTS mechanism used in the IEEE 802.11 network) and reservation-based mechanisms (e.g., the three-way handshake procedure used in the IEEE 802.16 mesh network) to prevent the HT problem from occurring may introduce time and bandwidth overheads for packet transmission. Such extra overheads also decrease the performance benefits achieved by network coding. Therefore, efficiently eliminating the HT problem is essential for a network coding scheme to achieve good performances in real-life networks.

The objective of this paper is to propose a stateless inter-session network coding scheme that can be easily implemented in the real world and at the same time still provide good data forwarding performances. Our work makes two contributions. First, in this paper we propose an easy-to-implement and easy-to-deploy inter-session network coding scheme that is based on an opportunistic approach. As compared with previous opportunistic coding schemes, such as COPE [7] and BFLY [8], our proposed rule-based network coding (RNC) scheme need not maintain the states of traffic flows to find coding opportunities or coding structures. Instead, our proposed scheme exploits only the routing and MAC-layer scheduling information that can be locally obtained by a node to find coding opportunities and coding structures. Due to this advantage, our proposed scheme can be easily realized in a real-life network.

Second, in this paper we point out a new "extended hidden terminal (EHT) problem," which can frequently occur in a wireless network using network coding. (In this paper, we call such a network a network-coding-based wireless network.) The EHT problem can fail the decoding of a network-coded packet, thus significantly decreasing the performance gain that can be achieved by network coding. To eliminate EHTs, mechanisms for a node to maintain the states of traffic flows and negotiate collision-free data multicasting schedules with neighboring nodes are required. Thus, this problem cannot be completely solved on top of a stateless network coding scheme. However, it is still possible to alleviate this problem over a stateless network coding scheme. In this paper we propose a smart bandwidth reservation mechanism that is extended from the one used in the IEEE 802.16(d) mesh CDS mode and aims at reducing EHTs for a stateless inter-session network coding scheme (such as RNC). Our simulation results show that, by reducing HTs and EHTs, our proposed stateless inter-session network coding scheme can increase end-to-end flow goodput and decrease the end-to-end packet delay in an interference-prone wireless network, as compared with traditional routing.

The remainder of this paper is organized as follows. We first present previous studies related to our work in Section 2 and then propose the basic design of our proposed opportunistic inter-session network coding scheme in Section 3. In Section 4, we explain why the EHT problem may occur in a network-coding-based wireless network. Our solution to this problem is presented in Section 5. In Section 6, we evaluate the performances of our proposed scheme using the NCTUns network simulator [6]. In Section 7, we discuss the applicability issue of our proposed scheme. Finally, we conclude this paper in Section 8.

2. Related work

In the literature, several papers [2–5] have theoretically studied inter-session network coding using linear programming. Although these previous studies have discussed the characteristics of inter-session network coding, turning their mathematical results into feasible distributed real-life protocols is difficult due to the huge computation complexity required by the mathematic operations of linear programming.

On the other hand, another track of inter-session network coding research is *opportunistic coding*, which aims to develop sub-optimal yet easy-to-implement and easy-to-deploy network coding schemes for unicast-flow wireless networks. In [7], Katti et al. implemented the COPE coding scheme over a real-life IEEE 802.11(b) wireless network. COPE performs network coding operation in an opportunistic manner. Its main operations are described here.

Using COPE, each node periodically broadcasts a *"reception report"* message to its neighboring nodes. A reception report message of node *i* contains the information of packets that node *i* currently possesses. Based on received reception report messages, a network node maintains a packet information table that records which packets are currently possessed by its neighboring nodes. Each time when a node is going to transmit a data packet, it first looks up the packet information table to check whether a coding opportunity exists or not. If so, this node mixes several

[1] The convergence of a network-coding protocol is defined as: the coding and routing decisions of all nodes in a network is consistent and loop-free.

packets that can be coded together to form a new network-coded packet and then sends this new network-coded packet out. If not, data packets are immediately transmitted.

COPE uses the above protocol to find coding opportunities within a node's one-hop neighborhood, meaning that using it a node can only find coding opportunities in a 3-node chain-based topology or a star-based topology. In [8,9], Omiwade et al. propose the BFLY scheme that allows nodes to find coding opportunities in a butterfly topology. The operation of BFLY is briefly described here.

Using BFLY, each node *i* should periodically broadcast a hello message to its neighboring nodes. A hello message of node *i* contains node *i*'s neighboring node list and a neighbor vector describing the relative neighboring relationship (regarding the butterfly structure) of node *i* and its neighboring nodes. With such information, each node can maintain all of its possible butterfly coding structures based on the hello messages advertized by its neighboring nodes. Upon transmitting data packets, a node first checks whether its outgoing packets can form coding opportunities over the maintained butterfly structures. If so, the node mixes these packets to generate new network-coded packets and then transmits the coded packets out instead of transmitting original data packets. Also, BFLY can collaborate with COPE (called BFLY-COPE in this paper) to find coding opportunities in chain-based, star-based, and butterfly-based topologies.

The idea of BFLY-COPE is similar to that of our work. Compared with COPE and BFLY, however, our proposed network coding scheme has several advantages. First, COPE and BFLY need to employ extra protocols to exchange the information of either packets possessed by neighboring nodes (in COPE) or butterfly-structure relationship of neighboring nodes (in BFLY). In contrast, our proposed network coding scheme utilizes only the routing and MAC-layer information to find coding opportunities and thus need not employ any extra protocols to operate. Thus, the implementation complexity of our proposed scheme can be reduced, as compared with BFLY-COPE.

Second, to the best of the authors' knowledge, previous work, such as COPE and BFLY, do not consider the influence of the HT and EHT problems, which may occur in real-life networks and can significantly degrade network coding performances. Thus, these schemes may only work well over an interference-free wireless network, which is uncommon in the real world. In contrast, our work uses a smart bandwidth reservation mechanism to reduce the occurrences of HTs and EHTs, which allows an inter-session network coding scheme to achieve better coding performances in an interference-prone wireless network than previous work.

Regarding solving the extended hidden terminal problem, the objective of [11,12] is similar to that of our work. However, there are two differences between these two studies. First, the scheme proposed in [11,12] (called CORE) uses a two-phase design that aims to provide stable and persistent data schedules to serve network coding before starting it and has to start network coding after network flows become stable (unchanged). Thus, it requires longer latencies to start network coding and is more suitable for long-lasting flows. In contrast, our work aims to propose a solution to solve the EHT problem in a one-phase manner. Data schedules obtained using our proposed approach may not be as optimal as those obtained using the CORE scheme; however, it requires shorter latencies to start network coding and is more suitable for dynamic flows. Second, the work in [11,12] discusses the problems when deploying network coding in cross-flow coding structures (i.e., chain, X, and star coding structures). In this paper, we explain why the EHT problem decreases the performances of network coding in chain and butterfly coding structures and propose a solution to solve it. We conducted proof-of-concept simulations in chain and grid topologies. Our simulation results show that the proposed scheme can effectively reduce EHTs and thus can further increase the flow goodputs of network coding.

Several previous works proposed network coding schemes for the IEEE 802.16 Point-to-multiPoint (PMP) mode network [14,15]. A PMP-mode network is essentially a cellular network, which employs a central base station to schedule the network bandwidth and completely differs from the CDS-mode network. These studies focus on adjusting some operational MAC-layer parameters and scheduling policies for PMP-mode networks, when cross-flow network coding can be used. In contrast, our work proposes enhancements to the MAC-layer designs specific to the IEEE 802.16 mesh CDS mode network, which is purely distributed and does not require a central unit to schedule link bandwidth. Such differences make our work unique to the literature. Table 1 shows the qualitative comparison between the previous works and our proposed RNC scheme. The comparison is based on the type of network for which they are designed, the used network coding

Table 1
Comparisons between previous works and our proposed RNC scheme.

	Jin and Li [14]	Zhang and Li [15]	COPE [7]	BFLY [8]	Mogre et al. [11,12]	Proposed RNC
CDS-mode network			*	*	*	*
PMP mode network	*	*	*			
Opportunistic network coding		*	*	*	*	*
Random network coding	*					
Proactive design	*	*	*	*	*	
Reactive design						*
Stateless design						*
Using butterfly structure				*		*
Solving the EHT problem						*
Seeking for optimization		*			*	
Protocol time overhead		*			*	

Table 2
Frequently used notations.

Notation	Meaning
CN	The current node
DST(p)	The destination node of packet p
FHP	The proposed basic four-way handshake procedure
NBR(A)	The set of nodes that is one-hop away from node A
NH(p)	The next-hop node of packet p
Pkt(A)	A packet transmitted by node A
SHP	The proposed smart handshake procedure
TX(p)	The transmitting node of packet p
THP	The three-way handshake procedure used in IEEE 802.16 networks

technique, design choice, and the issues that they intend to address.

3. Basic design of our proposed scheme

In this paper, we developed a bit-level network coding scheme, which can detect and perform network coding for chain and butterfly structures in an opportunistic and rule-based manner. To enable packet overhearing, which is necessary to perform network coding, we assume that each node's radio operates in the promiscuous mode. This is accomplished by allowing each node to receive all overheard packets despite their connection IDs [16] and disabling the encryption/decryption functions of the security and privacy sub-layer in the IEEE 802.16 mesh CDS-mode MAC layer.

Our proposed network coding scheme comprises two components: (1) a rule-based coding/decoding unit (RCU) and (2) a smart bandwidth reservation mechanism. The former is responsible for examining whether a coding opportunity exists and performing encoding/decoding operations, while the latter is responsible for establishing data schedules without generating HTs and EHTs in a network-coding-based wireless network. (In this paper, a data schedule refers to a set of consecutive minislots reserved for transmitting data packets. The notion of a minislot is explained later in Section 4.) In the following, we first explain the design and implementation of RCU in this section. The details of the proposed smart bandwidth reservation mechanism will be explained in Section 5 later.

RCU generates a network-coded packet by mixing two distinct fresh packets using the exclusive-or (XOR) operation. The reason why it uses the simple XOR operation to generate coded packets is explained here. The analytical results presented in [13] suggest that, when only inter-session coding is present, using a good coding function is not necessary to achieve good coding performances. Instead, other factors, such as whether good coding paths can be found in the network, are more important to the coding performances. Thus, for inter-session network coding, the coding performances that can be achieved using simple XOR operations are the same as those using more advanced coding operations, e.g., finite field operations.

The objective of our proposed coding scheme is not finding the optimal coding path in the network. This is because doing so may consume a great amount of time and computation overheads and may be infeasible in a network with fast-changing traffic. Instead, our proposed network coding scheme aims to provide an easy-to-implement and easy-to-deploy encoding/decoding approach that on average achieves better throughput and delay performances for a network in an opportunistic manner.

RCU is implemented at the MAC layer and composed of three parts: (1) the packet transmission procedure; (2) the packet reception procedure; and (3) the coding rule sets, which are explained below. For readers' convenience, we first present the notations frequently used in this paper in Table 2 and the packet header formats used by RCU in Section 3.1.

3.1. Packet header formats used by RCU

Using the proposed RNC-based scheme, each node should prepend an extra network-coding header to each outgoing packet. As shown in Fig. 1, the added network-coding header comprises two parts: (1) the Common Header (ComHdr) and (2) the Component Packet Information (CPI). Following the network-coding header are the IP header and the IP data payload (e.g., a TCP/UDP packet).

ComHdr is composed of three fields: (1) CPIN; (2) PL; and (3) TTL. The CPIN field indicates the number of CPI entries contained in this packet. The details of a CPI entry are explained later. The CPIN value is set to 2 for a network-coded packet, because our proposed coding scheme only mixes two fresh packets to generate a network-coded packet. (In this paper, we call ingredient packets that are coded together to form a network-coded packet p_{nc} as p_{nc}'s

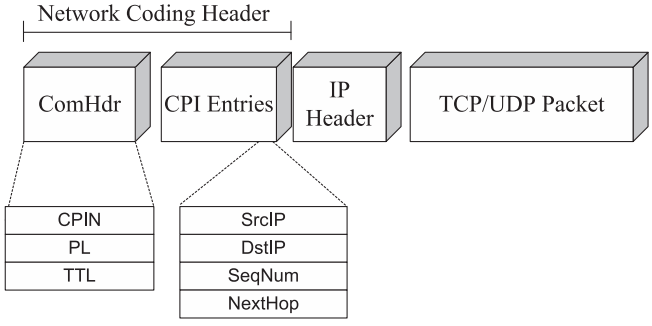

Fig. 1. The format of the network-coding header.

component packets.) For a fresh packet, the CPIN value is set to 1 because it comprises only one packet.

The PL field denotes the length of the data payload contained in this packet in bytes. For a fresh packet, the PL field is set to the length of the complete IP packet that it contains. For a network-coded packet, this value is set to the length of the longer "component IP packet" that it contains. The TTL field specifies the maximum number of hops that a packet is allowed to be forwarded. When receiving a packet, a network node should decrement the TTL value of the received packet by one. The TTL value of a packet p should be set based on the following rules: (1) it should be set to 2, if p is a coded packet that is coded using the butterfly structure coding; (2) Otherwise, it should be set to 1. (The details of the coding rules used in our schemes are explained in Section 3.4.)

The CPI part describes the information of the component packets that are coded together to form this coded packet. Each component packet is described by a CPI entry, which is composed of four fields: (1) SrcIP; (2) DstIP; (3) SeqNum; and (4) NextHop. The SrcIP field denotes the IP address of this component packet's source node; the DstIP field denotes the IP address of the component packet's destination node; the SeqNum field denotes the sequence number of this component packet assigned by its source node; and the NextHop field indicates the IP address of the next hop node to which this component packet should be transmitted. In our scheme, the SrcIP and SeqNum fields are used together to form a unique ID (SrcIP, SeqNum) for each fresh packet in the network.

3.2. Packet transmission procedure

On a transmitting node, outgoing packets (either generated by itself or forwarded) are placed into the MAC-layer output queue for transmission. The transmitting node first checks whether there is any established data schedule. If yes, it transmits the packets in the output queue using Algorithm 1. If not, the transmitting node should first establish a data schedule for the outgoing packets. Only after it has established a data schedule with a neighboring node, can it transmit data packets buffered in the output queue. The 802.16 standard defines a three-way handshake procedure (THP) for nodes to establish data schedules. However, this THP design is only suitable for routing-based wireless network and is inefficient in a network-coding-based wireless network. This issue is discussed in Section 4 and our solution to this problem is presented in Section 5.

The variables and functions used in Algorithm 1 is explained here. S_{out} denotes the set of the packets currently buffered in the MAC-layer output queue. Packets in S_{out} are sorted in the non-decreasing order based on their insertion times. M denotes the MAC-layer burst that is going to be transmitted over the forthcoming data schedule. The dequeue (S) function dequeues the first element of a sorted set S while the $p(S,i)$ function returns the ith packet in a sorted packet set S. The extract (S,i) function performs the following actions in sequence: (1) copy the ith packet pkt in a sorted packet set S to p_{tmp}; (2) remove the pkt in S; (3) re-sort S; and (4) return p_{tmp}.

The chain_coding_opp_checker (pkt_1,pkt_2) and butterfly_coding_opp_checker (pkt_1,pkt_2) functions check whether pkt_1 and pkt_2 can form a chain coding opportunity and a butterfly coding opportunity, respectively. They return "*PASSED*," if the two input packets can form a chain/butterfly coding opportunity. Otherwise, they return " FAILED." The chain_coding (pkt_1,pkt_2) and butterfly_coding (pkt_1,pkt_2) functions encode pkt_1 and pkt_2 to form a coded packet using chain/butterfly structure coding and properly set the ComHdr and CPI entries of the coded packet for its future decoding.

Initially, the transmitting node first dequeues the first element of S_{out} (denoted as p_{head}). It then iteratively checks whether any chain-structure coding opportunity exists for p_{head} and other packets in S_{out}. If such a coding opportunity exists, it first encodes these two packets to generate a new network-coded packet p_{coded} using chain structure coding and then inserts p_{coded} into M. Otherwise, it then iteratively checks whether p_{head} and other packets in S_{out} can form a butterfly-structure coding opportunity. If such a packet exists, the transmitting node first encodes p_{head} and the found packet to generate a network-coded packet p_{coded} using butterfly structure coding and then inserts p_{coded} into M. On the other hand, if no coding opportunity exists for p_{head}, the transmitting node directly inserts p_{head} into M.

Such a procedure is repeated until one of the following two conditions is satisfied: (1) S_{out} becomes empty; or (2) the total length of M has equaled or exceeded its maximum allowed length. In these conditions, the transmitting node sends M out to finish its data transmission procedure.

Algorithm 1. Packet encoding procedure for node i

```
1:  S_out := {p|∀p is a fresh packet sorted by its insertion time}
2:  M := the MAC-layer burst that is going to be transmitted out
Require: Node i is allowed to transmit M
3:  while S_out ≠ ∅ do
4:      p_head ← dequeue (S_out)
5:      res := UNPASSED
6:      for j = 1 to |S_out| do
7:          res ← chain_coding_opp_checker (p_head, p(S_out, j))
8:          if res = PASSED then
9:          p_peer ← extract (S_out, j)
10:             p_coded ← chain_coding (p_head, p_peer)
11:             Insert p_coded into M
12:             break
13:     end if
14:   end for
15:   if res ≠ PASSED then
16:     for j = 1 to |S_out| do
17:         res ← butterfly_coding_opp_checker (p_head, p(S_out, j))
18:         if res = PASSED then
19:             p_peer ← extract (S_out, j)
20:             p_coded ← butterfly_coding (p_head, p_peer)
```

(*continued on next page*)

```
21:            Insert p_coded into M
22:            break
23:          end if
24:        end for
25:      end if
26:      if res ≠ PASSED then
27:        Insert p_head into M
28:      end if
29:      if the payload length of M equals or exceeds
         the maximum allowed length then
30:        break
31:      end if
32:    end while
33:    Transmit M
34:    Return SUCCESS
```

3.3. Packet reception procedure

Upon receiving a packet (denoted as p_{recv}), a receiving node i should perform the packet decoding procedure shown in Algorithm 2. Initially, node i first checks whether p_{recv} is a fresh packet or not. If it is, node i then invokes the decoding_func() function (shown in Algorithm 3) to decode the packets stored in the coded packet pool as possible as it can, in a recursive manner. (The coded packet pool temporarily stores the received coded packets that have not yet decoded correctly.) The decoding_func() first inserts p_{recv} into the fresh packet pool (which temporarily stores fresh and decoded packets). It then iteratively tries to decode packets stored in the coded packet pool. This is accomplished by calling the network_coding_decoder() function, which is explained later.

If any packet in the coded packet pool is successfully decoded, the decoding_func() passes it to another decoding_func() instance. Such a recursive decoding process terminates when no more coded packets can be decoded further. At the end of the decoding_func(), it checks whether node i is the next-hop node of p_{recv}.[2] If it is, the decoding_func() passes p_{recv} to the upper-layer routing protocol for further route dispatching.

On the other hand, if p_{recv} is a network-coded packet, node i first tries to decode p_{recv} by invoking the network_coding_decoder() function with p_{recv} as its input. The network_coding_decoder() function iteratively searches the fresh packet pool to find whether any p_{recv}'s component packet exists. If yes, it decodes the other component packet of p_{recv} and returns the decoded fresh packet as its output. If p_{recv} cannot be decoded by the network_coding_decoder() function, node i first checks whether the TTL value of p_{recv} is larger than zero and the next-hop nodes of p_{recv}'s components are node i's neighboring nodes. If these conditions hold, it means that p_{recv} should be re-broadcast again because it is coded using the butterfly structure coding and has not yet reached the nodes possessing its remedy packets. (The details of the proposed coding rules and the usage of a packet's TTL value will be explained in Section 3.4.) Therefore, node i should re-insert p_{recv} into the MAC-layer output queue to rebroadcast it. If not, it means that the remedy packet for p_{recv} is not present in the fresh packet pool. In this condition, node i inserts p_{recv} into the coded packet pool for future possible decoding.

To prevent stored packets from consuming too much storage space, each packet in the fresh packet pool and the coded packet pool is associated with an expiry timer, which indicates the maximum time that a packet is allowed to reside in the pool. When an expiry timer expires, its associated packet is discarded immediately. In our simulations, the expiration time for a fresh packet is set to 10 s and that for a coded packet is set to 5 s.

Algorithm 2. Packet decoding procedure for node i

```
Require: Node i receives a MAC-layer data frame M,
  extracts the packets contained in M, and stores
  them into a packet set S_recv
1:   while S_recv ≠ ∅ do
2:     p_recv ← dequeue (S_recv)
3:     if p_recv is a fresh packet then
4:       decoding_func(p_recv)
5:     else
6:       p_decoded ← network_coding_decoder (p_recv)
7:       if p_decoded ≠ NULL then
8:         decoding_func(p_decoded)
9:       else
10:        if the TTL value of p_recv > 0 and NH (p_recv's
     component packets) ∈ NBR (i) then
11:          Insert p_recv to the MAC-layer connection
     queue
12:        else
13:          Insert p_recv to the coded packet pool
14:        end if
15:      end if
16:    end if
17:    Return SUCCESS
18:  end while
```

Algorithm 3. decoding_func (input: a fresh packet p_{recv})

```
1:   Insert p_recv to the fresh packet pool
2:   for all packet tmp_p in the coded packet pool do
3:     p_decoded ← network_coding_decoder (p_tmp_p)
4:     if p_decoded ≠ NULL then
5:       decoding_func(p_decoded)
6:     end if
7:   end for
8:   if node i = NH (p_recv) then
9:     Pass p_recv to the upper-layer routing protocol
     for further route dispatching
10:  end if
11:  Return SUCCESS
```

[2] Note that the next-hop node information of a packet can be obtained via retrieving current routing entries exported by the collaborative routing protocol.

3.4. Coding rule sets

The rule set used to find a chain-structure coding opportunity is called the chain-structure rule set (CRS) and that to find a butterfly-structure coding opportunity is called the butterfly-structure rule set (BRS). The details of these two rule sets are explained below.

3.4.1. CRS

The rules of CRS are listed in Table 3. Given two packets $p1$ and $p2$, if the transmitting node of $p1$ is the next-hop node of $p2$ and the transmitting node of $p2$ is the next-hop node of $p1$, then a chain-structure coding opportunity exists. In our implementation, the checking of CRS is implemented in the chain_coding_opp_checker ($p1$,$p2$) function, which is used by each node's encoding procedure. The rationale of CRS is simple and explained below.

Consider a 3-node multi-hop chain network comprising nodes A, B, and C shown in Fig. 2. In this chain network, the radio coverages of nodes A and C contain node B and themselves, respectively. Thus, data packets from node A to node C should go through node B, and vice versa. Suppose that nodes A and C transmit data packets to each other. Packets transmitted by these two nodes will go through node B.

When node B receives a packet transmitted by node A (denoted as Pkt(A)) and a packet transmitted by node C (denoted as Pkt(C)), node B can code this pair of Pkt(A) and Pkt(C) together to generate a new coded packet (Pkt(A) $\oplus$ Pkt(C)) and sends it out instead of sending out Pkt(A) and Pkt(C) separately. Since node A intends to receive Pkt(C) and possesses Pkt(A), and node C intends to receive Pkt(A) and possesses Pkt(C), upon receiving the packet (Pkt(A) $\oplus$ Pkt(C)), node A can extract Pkt(C) by XOR-ing its possessed Pkt(A) and (Pkt(A) $\oplus$ Pkt(C)). Similarly, after receiving (Pkt(A) $\oplus$ Pkt(C)), node C can extract Pkt(A) by XOR-ing Pkt(C) and the received coded packet. Using this chain structure coding to forward data packets can theoretically reduce the number of required packet transmissions from 4 to 3, increasing of the packet transmission efficiency by 33%.

3.4.2. BRS

The chain structure coding increases the transmission efficiency of a wireless network by 33% at most, while, butterfly structure coding can theoretically increase the transmission efficiency of a wireless network by 50%. Consider a 6-node grid network topology shown in Fig. 3, where each dotted circle denotes the radio coverage of a node centered on it. In this network, node A intends to deliver packets to node F and node C intends to deliver packets to node D. Due to the limited radio coverage, the packets transmitted by nodes A and C should be forwarded by nodes B and E to reach their respective destination nodes. When traditional store-and-forward routing is used, six packet transmissions are required to accomplish a packet delivery from node A to node F and that from node C to node D. (The packet transmissions required by the former are A–B, B–E, E–F, and those required by the latter are C–B, B–E, E–D.)

In contrast, when butterfly structure coding is used, only four packet transmissions are required to accomplish these two packet deliveries. Using butterfly-structure coding, upon receiving Pkt(A) and Pkt(C), node B can code these two packets to form a coded packet (Pkt(A) $\oplus$ Pkt(C)) and then transmit this packet to node E. Later on, node E can broadcast this packet out again. Thus, nodes D and F can receive this coded packet (Pkt(A) $\oplus$ Pkt(C)) at the same time. Since node D is node A's one-hop neighboring node and node F is node C's one-hop neighboring node, due to the wireless broadcast nature, node D can overhear Pkt(A) when node A is transmitting it and node F can overhear Pkt(C) when node C is transmitting it. As a result, node D will have Pkt(A) when receiving (Pkt(A) $\oplus$ Pkt(C)) and node F will have Pkt(C) when receiving (Pkt(A) $\oplus$ Pkt(C)). Thus, node D can decode Pkt(C) by XOR-ing Pkt(A) and (Pkt(A) $\oplus$ Pkt(C)) and node F can decode Pkt(A) by XOR-ing Pkt(C) and (Pkt(A) $\oplus$ Pkt(C)). The total number of required packet transmissions becomes four only (A–B, C–B, B–E, and E–DF). Using the butterfly structure coding, a network can increase its packet transmission efficiency by 50%, as compared with those using the traditional routing scheme. A more detailed comparison between the chain structure coding and the butterfly structure coding is given in the Appendix.

Note that, when the chain structure coding is used, a coded packet is decoded on its next-hop node, which is only one-hop away from its transmitting node. However, when the butterfly structure coding is used, a coded packet should be decoded on the nodes that are two-hop away from its transmitting node to obtain performance gain. Due to this fundamental difference, the encoding/decoding mechanism for the chain structure greatly differs from that for the butterfly structure. The outline of the encoding and decoding procedures for the butterfly structure coding has been presented in Algorithms 1–3. In this section, we explain the operation of these two procedures in detail and present an example to show how a butterfly structure coding is completed.

The encoding operation of our proposed butterfly structure coding is composed of two parts: (1) rule checking and (2) route designation for the next-hop node. In our implementation, the former is realized in the butterfly_coding_opp_checker() function while the latter is realized in the butterfly_coding() function.

Upon searching for a butterfly-structure coding opportunity, each node n uses the rules shown in Table 4 to check whether any pairs of packets in its MAC-layer output queue can be coded together using the butterfly structure coding. For a pair of packets $p1$ and $p2$, satisfying the first rule indicates that $p1$ and $p2$ have the same next-hop node on the ways to their respective destination nodes, and satisfying the second rule indicates that the two packets belong to different flows and are destined to different destination nodes. Satisfying these two rules means that

Table 3
The rules of CRS.

Rule ID	Rule statement
1	TX($p1$) = NH($p2$)
2	TX($p2$) = NH($p1$)

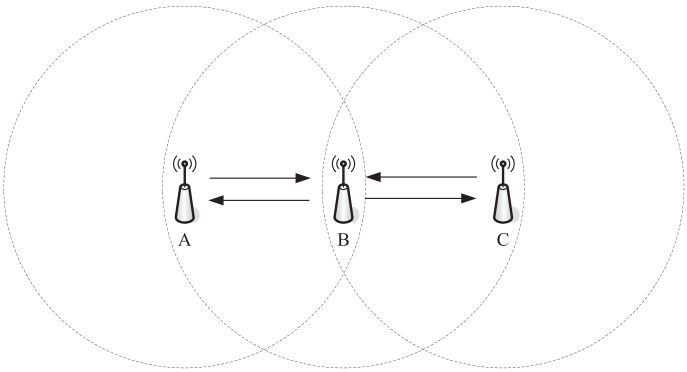

Fig. 2. An example topology suitable for the chain structure coding.

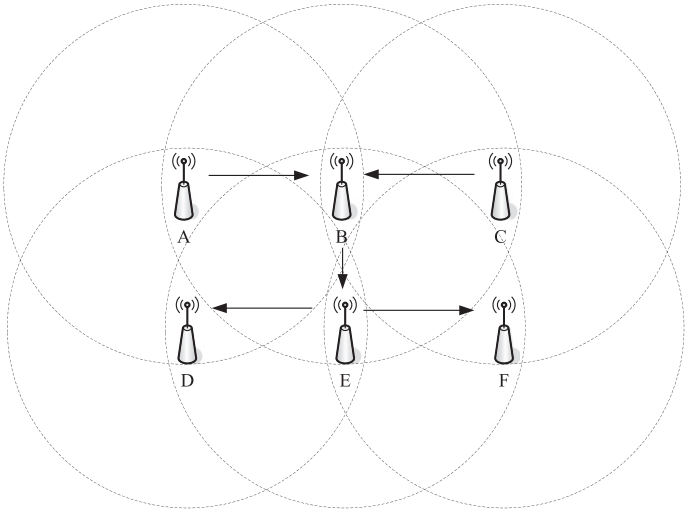

Fig. 3. An example topology suitable for the butterfly structure coding.

the routes of $p1$ and $p2$ to their respective destination nodes are likely to form a butterfly structure.

On the other hand, the third rule checks whether the one-hop neighboring node set of $p1$'s transmitting node and that of $p1$'s next-hop node have common nodes other than node n and n's one-hop neighboring nodes; the fourth rule checks whether the one-hop neighboring node set of $p2$'s transmitting node and that of $p2$'s next-hop node have common nodes other than node n and n's one-hop neighboring nodes. For $p1$ and $p2$, satisfying these two rules indicates that the coded packet formed by these two packets ($p1 \oplus p2$) is very likely to be decoded correctly later, because the remedy packets (i.e., $p1$ and $p2$) for ($p1 \oplus p2$) are very likely to be overheard by some nodes on the routes of $p1$ and $p2$.

Table 4
The rules of BRS.

Rule ID	Rule statement
1	$NH(p1) = NH(p2)$
2	$DST(p1) \neq DST(p2)$
3	$(NBR(TX(p1)) \cap NBR(NH(p1)) - (CN \cup NBR(CN))) \neq \emptyset$
4	$(NBR(TX(p2)) \cap NBR(NH(p2)) - (CN \cup NBR(CN))) \neq \emptyset$

The second part of our butterfly structure coding is designating the routes of ($p1 \oplus p2$)'s component packets used by ($p1 \oplus p2$)'s next-hop node, which is used to ensure that the decoding of ($p1 \oplus p2$) can be triggered and successfully performed. Also, the proposed route designation mechanism can effectively limit the spreading area of ($p1 \oplus p2$)'s broadcast, preventing link bandwidth from being wasted due to unnecessary broadcast packets.

This route designation is accomplished by specifying the *NextHop* field of each CPI contained in the coded packet using the rules listed in Table 5. As one sees, our butterfly structure coding chooses a node that is likely to possess $p1$ (and therefore is likely to be able to recover $p2$) as $p2$'s next-hop node. Similarly, it chooses a node that is likely to possess $p2$ (and therefore is likely to recover $p1$) as $p1$'s next-hop node.

By doing this, after node n broadcasts ($p1 \oplus p2$), it will be received by all of n's one-hop neighboring nodes. Among these nodes, however, only those that are neighboring to the next-hop nodes specified in the CPI entries of ($p1 \oplus p2$) are allowed to rebroadcast this coded packet. Thus, ($p1 \oplus p2$) can reach the nodes that possess its remedy packets (and thus get decoded correctly) while

Table 5
Route designation of the next-hop node of a coded packet using BRS.

Rule ID	Rule statement
1	$NH(p2) \in (NBR(TX(p1)) \cap NBR(NH(p1)) - (CN \cup NBR(CN)))$
2	$NH(p1) \in (NBR(TX(p2)) \cap NBR(NH(p2)) - (CN \cup NBR(CN)))$

avoiding unnecessary broadcasts for it. In addition, when using the butterfly structure coding, node n should set the TTL field of $(p1 \oplus p2)$ to 2, which explicitly indicates that $(p1 \oplus p2)$ is only allowed to be re-broadcast once. This design further reduces the number of broadcast packets for $(p1 \oplus p2)$.

We use the topology shown in Fig. 3 as an example to explain how a butterfly structure coding is accomplished. Upon receiving a Pkt(A) and a Pkt(C), node B first checks whether these two packets satisfy the rules of BRS. If they do, it means that Pkt(A) and Pkt(C) have the same next-hop node (indicated by rule (1)) but different destination nodes (indicated by rule (2)). In addition, these two packets are likely to be overheard by some nodes that are on their routes to their respective destination nodes (indicated by rules (3) and (4)).

In this condition, node B first codes Pkt(A) and Pkt(C) to generate (Pkt(A) ⊕ Pkt(C)). It then designates the routes of (Pkt(A) ⊕ Pkt(C))'s component packets (i.e., Pkt(A) and Pkt(C)) for (Pkt(A) ⊕ Pkt(C))'s next-hop node E. That is, node E should forward Pkt(A) and Pkt(C) (and thus (Pkt(A) ⊕ Pkt(C))) using the routes indicated by the *Next-Hop* fields of (Pkt(A) ⊕ Pkt(C))'s CPI entries. In this example case, for (Pkt(A) ⊕ Pkt(C))'s next-hop node E, node B designates node F to be Pkt(A)'s next-hop node and node D to be Pkt(C)'s next-hop. Lastly, node B broadcasts the coded packet (Pkt(A) ⊕ Pkt(C)) out.

After receiving the coded packet, node E will broadcast it again because node E is neighboring to nodes D and F, which are the next-hop nodes specified in the CPI entries of the coded packet. Finally, on receiving (Pkt(A) ⊕ Pkt(C)), nodes D and F can decode it to obtain Pkt(C) and Pkt(A), respectively, because the former possesses Pkt(A) and the latter possesses Pkt(C).

The operation of RCU on node n only requires (1) the neighborhood information of node n and its one-hop neighboring nodes; and (2) the next-hop node information for each packet on node n. The former can be easily obtained from the control messages of an IEEE 802.16(d) mesh network,[3] while the latter can be obtained by consulting the collaborative routing protocol. In modern operating systems, routing protocols usually have standard APIs to export their maintained route information. Therefore, it is easy to integrate RCU with the collaborative routing protocol.

4. The EHT problems

As introduced in Section 1, most of previous work assumes that no packet collisions occur in a network-coding-based network [7,10]. This assumption, however, is not always true in a real-life network. For example, the presence of HTs can result in severe packet collisions in a wireless network. To solve the HT problem, the IEEE 802.11 network employs the RTS/CTS mechanism to protect data packets from being collided. Using this mechanism, however, each pair of transmitting and receiving nodes have to exchange RTS and CTS messages before data transmission is carried out. Thus, although the RTS/CTS mechanism can avoid data packet collisions, it consumes much network bandwidth to transmit control messages. For this reason, the performance gain of inter-session network coding in a real-life 802.11(b) network can be quite low.

On the other hand, the IEEE 802.16(d) mesh network uses a reservation-based approach to schedule data transmission. In this network, control messages are transmitted over transmission opportunities (TxOpps) and data packets are transmitted over minislots. The operation of this network can be found in [16,17]. For brevity, we do not present it in this paper. Before transmitting data, the transmitting and receiving nodes have to complete a three-way handshake procedure (THP) to obtain a minislot allocation, during which the transmitting node is ready to transmit data and the receiving node is ready to receive data.

The operation of THP is explained here. First, the transmitting node transmits a request Information Element (IE) and an availability IE to the receiving node using an MSH-DSCH message. The request IE specifies the number of minislots that the requesting node needs to transmit data and the availability IE specifies a set of consecutive minislots on which the transmitting node can transmit data. That is, the request IE indicates the amount of link bandwidth that the transmitting node requests and the availability IE indicates the minislot set from which the receiving node can choose.

On receiving these two IEs, the receiving node first determines whether it can receive data from the transmitting node within the minislot set specified by the received availability IE. If not, the receiving node can simply ignore this bandwidth request. Otherwise, it should schedule a minislot allocation within the indicated minislot set and then transmit a grant IE back to the transmitting node as an acknowledgment using its MSH-DSCH message. The grant IE specifies the minislot set on which the receiving node is willing to receive data from the transmitting node. (The minislot set indicated in the grant IE must be the subset of the minislot set specified in its corresponding availability IE.)

Upon receiving the grant IE, the transmitting node then broadcasts a confirm IE using its MSH-DSCH message to complete this THP. The confirm IE is a copy of the received grant IE. It is used to notify the transmitting node's neighboring nodes of the activation information of this minislot allocation. By broadcasting grant and confirm IEs, nodes neighboring to the transmitting and receiving nodes can learn when this minislot allocation will take place and thus suspend their data transmissions in that duration. By using this three-way design, THP can eliminate HTs around the transmitting and receiving nodes. (Note that, in an 802.16(d) mesh network, transmitting

[3] The one-hop node information can also be obtained from hello messages or link-state messages received by the collaborative routing protocol.

MSH-DSCH messages is guaranteed collision-free due to the use of a distributed election algorithm [16].)

In our implementation, when network coding is used, a node that receives a confirm IE will not schedule its own data transmissions on the same minislots indicated by the received confirm IE to ensure that it will be idle at that time to overhear neighboring nodes' packets. Although this constraint may reduce the flexibility of minislot scheduling, our simulation results show that the proposed RNC-based schemes still greatly outperform traditional routing on end-to-end flow goodputs and packet delays by reducing the number of transmitted packets and decreasing packet queuing delays. (The minislot scheduling of traditional routing need not take this constraint into account.)

Unlike the RTS/CTS mechanism used in 802.11 networks, the THP used by 802.16(d) mesh networks can schedule a mini-slot allocation that lasts at most 1.28 s. Thus, the bandwidth and time overheads introduced by the THP can be amortized over time and therefore less significant, as compared with those introduced by the RTS/CTS mechanism. However, to inter-session network coding, only eliminating HTs around the transmitting and receiving nodes cannot avoid all types of packet collisions. In the following, we point out a new EHT problem which occurs only in network-coding-based wireless networks. The EHT problem can result in packet collisions different from those caused by the HT problem and significantly decrease the performance gain that can be achieved by inter-session network coding.

Fig. 4 shows an example 5-node chain network, where the dotted circles denote the transmission ranges of nodes B and D, respectively. A solid arrow from node $n1$ to node $n2$ represents that a packet transmitted by $n1$ is destined to $n2$ while a dotted arrow from node $n1$ to node $n2$ represents that a packet transmitted by $n1$ can be overheard by $n2$. (This also means that $n1$ can interfere with $n2$ if one of them transmits data and the other receives data from another node at the same time.) In this network, two greedy UDP flows are generating traffic. One is from node A to node E and the other is from node E to node A. Packets generated by these two flows have to be forwarded by nodes B, C, and D to reach their respective destination nodes.

Suppose that node B is going to forward a packet to node A and node D is going to forward a packet to node E. In a routing-based wireless network, these two packet transmissions can be scheduled to take place at the same time, because the receiving nodes A and E are not interfered with other nodes. Although the packets transmitted by nodes B and D may get collided on node C, such a packet collision does not affect the packet receptions of nodes A and E. In a network-coding-based wireless network, however, these two simultaneous packet transmissions can significantly decrease the performance of inter-session network coding. The reason is explained below.

In this chain network, when inter-session network coding is used, nodes B and D will use chain structure coding to code packets that are forwarded to different nodes. For example, node B will code Pkt(A) (destined to node C) and Pkt(C) (destined to node A) to generate a coded packet (Pkt(A) $\oplus$ Pkt(C)) and broadcast it out. Similarly, node D will code Pkt(E) (destined to node C) and Pkt(C) (destined to node E) to generate a coded packet (Pkt(E) $\oplus$ Pkt(C)) and broadcast it out. As one knows, if these two coded packets are simultaneously transmitted, they will be collided on node C. In such a condition, node C cannot overhear these two coded packets and thus cannot decode them to obtain any packets that nodes B and D intend to forward to it. Therefore, the packet forwarding performance of network coding will be greatly degraded due to these undesired packet collisions.

The EHT problem not only can occur in chain structures but also can occur in butterfly structures. We explain the latter using an example 9-node grid network shown in Fig. 5. In this example network, traffic is generated by four greedy UDP flows, whose source and destination node pairs are (A,I), (C,G), (I,A), (G,C), respectively.

The two flows (A,I) (C,G) select node B as the next-hop node of their packets and form a butterfly structure while the two flows (I,A), (G,C) select node H as the next-hop node of their packets and form another butterfly structure. In the first butterfly structure, packets generated by the flow (A,I) should be forwarded to node F to be decoded and those generated by the flow (C,G) should be forwarded to node D to be decoded. Similarly, in the second butterfly structure, packets generated by the flow (G,C) should be forwarded to node F to be decoded and those generated

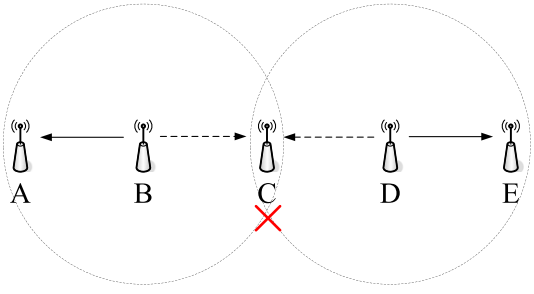

Fig. 4. The EHT problem in a chain network.

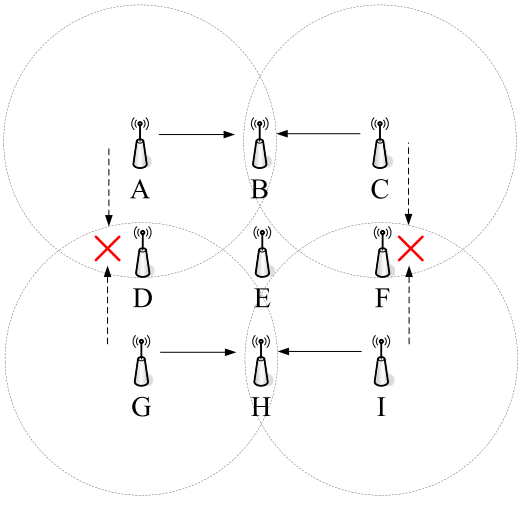

Fig. 5. The EHT problem in butterfly networks.

by the flow (I,A) should be forwarded to node D to be decoded.

To transmit data, nodes A and C need to negotiate minislot allocations with node B (denoted as MA_{AB} and MA_{CB}, respectively) using THP. Setting up MA_{AB} and MA_{CB} ensures that node B can successfully receive Pkt(A) and Pkt(C) without generating collisions. On the other hand, the flows (G,C) and (I,A) are also transmitting their data using the second butterfly structure at the same time. Similarly, nodes G and I need to negotiate minislot allocations, MA_{GH} and MA_{IH}, with node H using THP. Setting up these two minislot allocations ensures that node H can successfully receive Pkt(G) and Pkt(I) without generating collisions.

However, one should know that, for the first butterfly structure, node D is required to successfully overhear Pkt(A) to decode (Pkt(A) $\oplus$ Pkt(C)) and, for the second butterfly structure, it is also required to successfully overhear Pkt(G) to decoded (Pkt(G) $\oplus$ Pkt(I)). Similarly, for the first butterfly structure, node F is required to successfully overhear Pkt(C) to decode (Pkt(A) $\oplus$ Pkt(C)) and, for the second butterfly structure, it is required to successfully overhear Pkt(I) to decoded (Pkt(G) $\oplus$ Pkt(I)). Since MA_{AB} and MA_{CB} only guarantee the success of node B's packet reception and MA_{GH} and MA_{IH} only guarantee the success of node H's packet reception, it is not guaranteed that node D can successfully receive packets transmitted from nodes A and G and node F can successfully receive packets transmitted from nodes C and I.

In such a condition, if nodes A and G simultaneously transmit their packets, their packets will be collided on node D. Similarly, if nodes C and I transmit their packets at the same time, their packets will be collided on node F. Because nodes D and F cannot obtain the necessary remedy packets for (Pkt(A) $\oplus$ Pkt(C)) and (Pkt(G) $\oplus$ Pkt(I)), they will fail to decode these coded packets. As a result, such undesired packet collisions between different butterfly structures can greatly decrease the network goodput achieved by inter-session network coding. To solve this EHT problem, we propose a smart handshake procedure (SHP) to replace the original THP for IEEE 802.16(d) mesh networks. The design of SHP is explained in Section 5.

5. Smart handshake procedure

Our proposed SHP comprises four parts. The first is a four-way handshake procedure (abbreviated as FHP) extended from the THP defined in the IEEE 802.16 mesh-mode standard; the second is the resolution for scheduling conflicts; the third is the activation time estimation for a minislot allocation; and the last is the timing control for starting an FHP. In this section, we explain the details of these parts in sequence.

5.1. FHP

The operation of FHP is illustrated in Fig. 6, where a solid arrow from $n1$ to $n2$ denotes a message transmitted from $n1$ and destined to $n2$ and a dotted arrow from $n1$ to $n2$ denotes a message transmitted from $n1$ and can be overheard by $n2$. The operation of FHP is the same as that of THP except that, in FHP, some nodes are required to broadcast a new type of IE called "extended confirm" IE (denoted as ExtConfirm IE in the figure for brevity) to further disseminate the activation information of a scheduled minislot allocation. The detailed operation of FHP is explained below.

First, the requesting node sends the granting node a request IE. Then, the granting node acknowledges the requesting node with a grant IE. After receiving the grant IE, the requesting node broadcasts a confirm IE to accomplish the original THP. Later on, upon receiving a confirm IE, nodes other than the granting node have to broadcast extended confirm IEs to further distribute the activation information of this minislot allocation. An extended confirm IE is simply a copy of its corresponding confirm IE. By broadcasting extended confirm IEs, nodes that are two-hop away from the requesting node can know when this minislot allocation will be activated. Thus, they can suspend their data transmission during that period. Using this design, for two overlapped coding structures, their transmitting nodes can transmit their data at different times to avoid packet collisions resulting from the EHT problem.

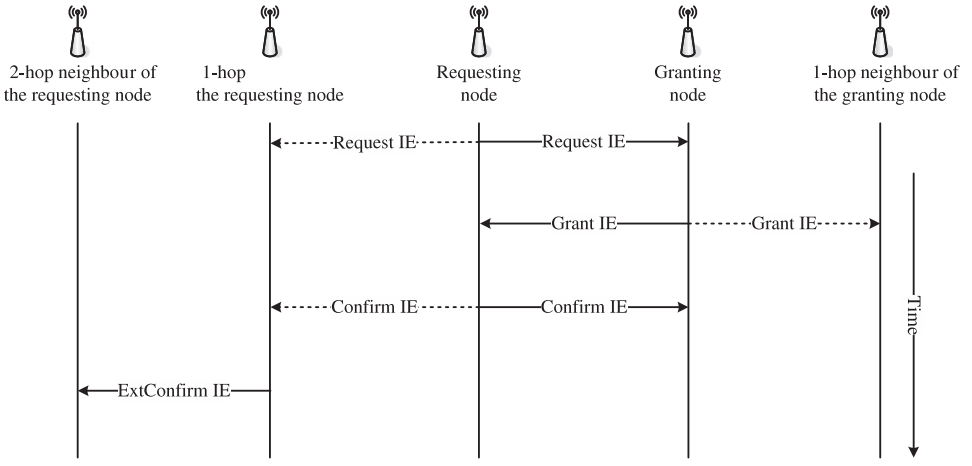

Fig. 6. The operation of FHP.

Let us revisit the example shown in Fig. 5. If FHP is used, upon receiving node A's confirm IE, node D will broadcast an extended confirm IE to notify its neighboring nodes of this minislot allocation. Thus, node G can know when node D will overhear packets transmitted by node A and thus restrain its packet transmissions at that period. Similarly, upon receiving node G's confirm IE, node D will broadcast an extended confirm IE to notify neighboring nodes of node G's minislot allocation. This allows node A to know when node D overhears node G's packets. Thus, it can restrain its packet transmissions during that period to avoid packet collisions on node D. As a result, FHP can reduce the number of EHTs and thus reduce packet collisions resulting from them in network-coding-based networks.

5.2. Scheduling conflict resolution

Due to the holdoff time design of the IEEE 802.16(d) mesh CDS mode [16], it takes some time to finish transmitting the necessary IEs of an FHP (i.e., request, grant, confirm, and extended confirm IEs). This means that two FHPs may overlap on the time axis, which can cause their data schedules to overlap with each other. That is, this may result in two ongoing FHPs scheduling minislots that conflict with each other. On receiving an extended confirm IE, which asks it to suspend transmission during the specified duration, a node may find that its ongoing FHP has chosen some part of the specified duration for transmission. Fig. 7 illustrates an example of such a scheduling conflict.

In this example network, node R1 intends to schedule a minislot allocation (denoted as MA1) with node G1 and node R2 intends to schedule a minislot allocation (denoted as MA2) with node G2. As one sees, nodes R1 and R2 independently initiate their own FHPs with nodes G1 and G2 during the same period. Therefore, node R1 does not know the existence of MA2 when scheduling MA1 because it has not yet received the extended confirm IE of MA2. Similarly, node R2 does not know the existence of MA1 when scheduling MA2 because it has not received the extended confirm IE of MA1 at that time. In this condition, nodes R1 and R2 may schedule MA1 and MA2 that overlap with each other, resulting in a scheduling conflict.

Note that, after receiving an extended confirm IE of MA2, node R1 will detect that MA1 overlaps with MA2. Similarly, after receiving an extended confirm IE of MA1, node R2 will detect that MA2 overlaps with MA1. At this point, however, the minislot allocations MA1 and MA2 have been scheduled on the sending and receiving nodes and cannot be canceled.

When a scheduling conflict occurs, it is important to determine which node can transmit data and which node should not. For this purpose, we propose a Scheduling Conflict Resolution Algorithm (SCRA). SCRA takes two minislot allocations as its inputs and returns the one that can use the overlapped minislots as its output. Its pseudo code is shown in Algorithm 4, where the function *sf* takes a minislot allocation MA as its input and returns the starting frame number of MA as its output. The functions *sm* and *nid* take a minislot allocation MA as their input and return the starting minislot number of MA within a frame and the ID of MA's transmitting node as their output, respectively. The details of SCRA are explained below. (Note that the pseudo code of SCRA shown here is for illustration purpose only. The details of SCRA for solving the frame number wrapping problem are omitted in this paper for brevity.)

Upon receiving an extended confirm IE, node i first obtains the minislot allocation MA_{ec} from the received extended confirm IE and then checks whether an existing minislot allocation conflicts with MA_{ec}. If no such a minislot allocation exists, SCRA immediately terminates and returns MA_{ec} as its output. Otherwise, SCRA sets the conflicting minislot allocation to MA_{ori} and performs the following checks: First, if the starting frame number of MA_{ec} is larger than that of MA_{ori}, SCRA returns MA_{ori} as its outputs. Otherwise, in case the starting frame numbers of the two minislot allocations are the same, SCRA returns the minislot allocation that has a larger starting minislot number within a frame as its output. In case the sf and sm values of MA_{ec} and MA_{ori} are the same, the tie is

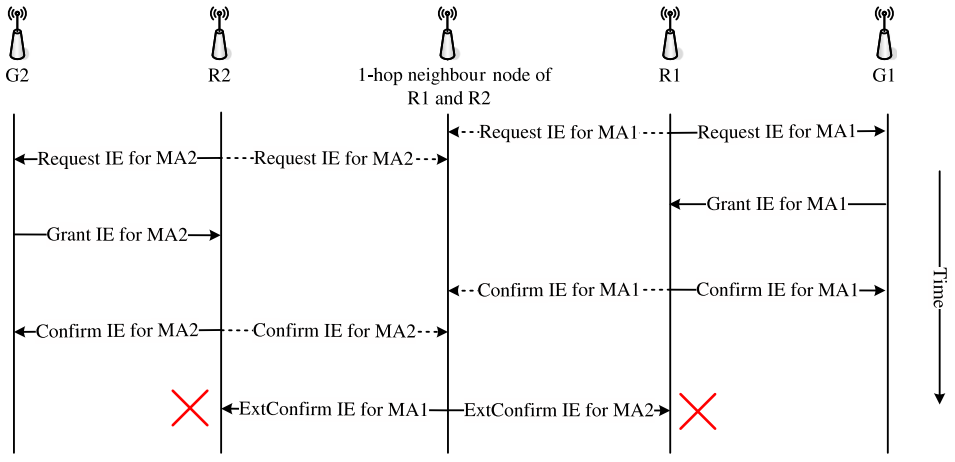

Fig. 7. Illustration of the scheduling conflict problem.

resolved by comparing the IDs of their transmitting nodes. In this condition, SCRA returns the minislot allocation whose transmitting node ID is larger as its output.

Algorithm 4. The Scheduling Conflict Resolution Algorithm for node *i*

Require: Node *i* receives an extended confirm IE
1: MA_{ec} := the minislot allocation obtained in the extended confirm IE
2: MA_{ori} := the existing minislot allocation conflicting with MA_{ec}
3: **if** $MA_{ori} = \emptyset$ **then**
4: Return MA_{ec}
5: **end if**
6: WinnerMA ← ∅
7: **if** sf (MA_{ec}) is larger than sf (MA_{ori}) **then**
8: WinnerMA ← MA_{ori}
9: **else if** (sf (MA_{ec}) = sf (MA_{ori})) and (sm (MA_{ori}) is larger than sm (MA_{ec})) **then**
10: WinnerMA ← MA_{ori}
11: **else if** (sf (MA_{ec}) = sf (MA_{ori})) and (sm (MA_{ec}) = sm (MA_{ori})) and (nid (MA_{ori}) is larger thannid (MA_{ec})) **then**
12: WinnerMA ← MA_{ori}
13: **else**
14: WinnerMA ← MA_{ec}
15: **end if**
16: Return WinnerMA

By using SCRA, after receiving an extended confirm IE that indicates minislot allocation conflicts, nodes can use FHP to resolve them. Compared with THP, the FHP design with SCRA (denoted as FHP-SCRA) can increase scheduling performances. However, it may encounter two problems that degrade its scheduling performances. We explain these two problems and our proposed solutions to below.

5.3. Activation time estimation for a minislot allocation

The first problem of FHP-SCRA is that, the bandwidth before the extended confirm IEs of conflicting minislot allocations can be broadcast out is wasted. As can be seen in Fig. 8, when using FHP-SCRA, two neighboring requesting-granting node pairs (R1,G1) and (R2,G2) may independently launch two FHPs during the same period. In such a condition, (R1,G1) and (R2,G2) may schedule their respective minislot allocations MA1 and MA2 during the same period (denoted as the rectangles shown in Fig. 8). By the aid of SCRA, nodes R1 and R2 can resolve the scheduling conflicts between MA1 and MA2 after receiving the extended confirm IEs for these two minislot allocations. However, before such extended confirm IEs can be broadcast by the one-hop neighboring nodes of R1 and R2, MA1 and MA2 may have already been activated. In this condition, R1 and R2 will transmit their data at the same time until receiving the extended confirm IEs of MA1 and MA2, causing packet collisions.

To solve this problem, we propose an Activation Time Estimation Algorithm (ATEA) for requesting nodes to estimate a good activation timing for their minislot allocations (in unit of frame number). As Fig. 9 shows, using ATEA each node can schedule a minislot allocation that is activated after its launched FHP is completed. This will reduce the bandwidth wasted for such collided packets. ATEA takes the next MSH-DSCH TxOpp information of all neighboring nodes as its inputs and returns a suitable starting frame number for a requested minislot allocation as its output. Its pseudo code is shown in Algorithm 5 and explained below. (Note that the pseudo code shown in Algorithm 5 is for illustration purpose only. The details of ATEA for solving the frame number and TxOpp number wrapping problems are omitted in this paper for brevity.)

Algorithm 5. The Activation Time Estimation Algorithm for node *i*

1: N_G := the granting node
2: MyNextTxOpp := the next MSH-DSCH TxOpp of node i
3: NbrList ← (NBR (*i*) − N_G)
4: **if** MyNextTxOpp is larger than the next MSH-DSCH TxOpp of N_G **then**
5: EstTxOpp ← MyNextTxOpp
6: **else**
7: EstTxOpp ← the next MSH-DSCH TxOpp of N_G
8: **end if**
9: **for** node $j \in$ NbrList **do**
10: $TxOpp_j$ ← the next MSH-DSCH TxOpp of node *j*
11: **if** $TxOpp_j$ is larger than EstTxOpp **then**
12: EstTxOpp = $TxOpp_j$
13: **end if**
14: **end for**
15: EstFrNum ← *fr*(EstTxOpp)
16: Return EstFrNum

To determine the starting frame number for a requested minislot allocation (on the requesting node), ATEA first compares the next MSH-DSCH TxOpp of the requesting node (i.e., the current node) and that of the granting node. If the former is larger than the latter, it means that, after receiving request/availability IEs from the requesting node, the granting node can transmit grant IEs back to the requesting node over a TxOpp that is preceding to the requesting node's next TxOpp. From this information, ATEA can estimate when the requesting node can transmit its confirm IE and when the broadcasts of the extended confirm IEs can be finished to calculate an appropriate starting frame number for the requested minislot allocation. This is accomplished by two steps: First, ATEA chooses the largest MSH-DSCH TxOpp number among the next MSH-DSCH TxOpp of the requesting node and those of all its one-hop neighboring nodes (except the granting node) as the estimated TxOpp number. Second, ATEA uses the *fr* function to compute the number of the frame that contains the estimated TxOpp number as its output. The rationale of this design is explained below.

If the next MSH-DSCH TxOpp of a neighboring node *n* is larger than that of the requesting node, it implies that, upon receiving the requesting node's confirm IE, node *n*

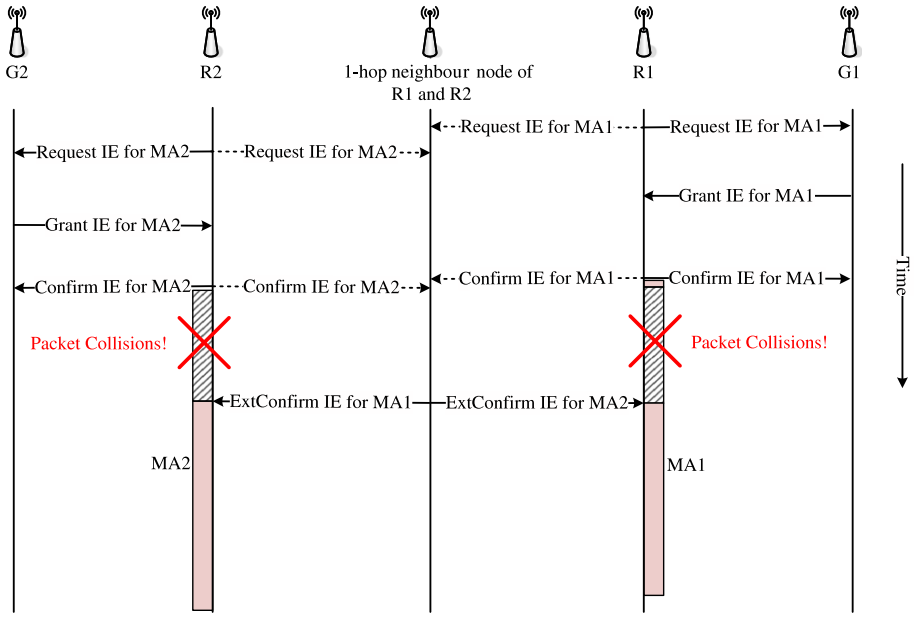

Fig. 8. Why ATEA is needed.

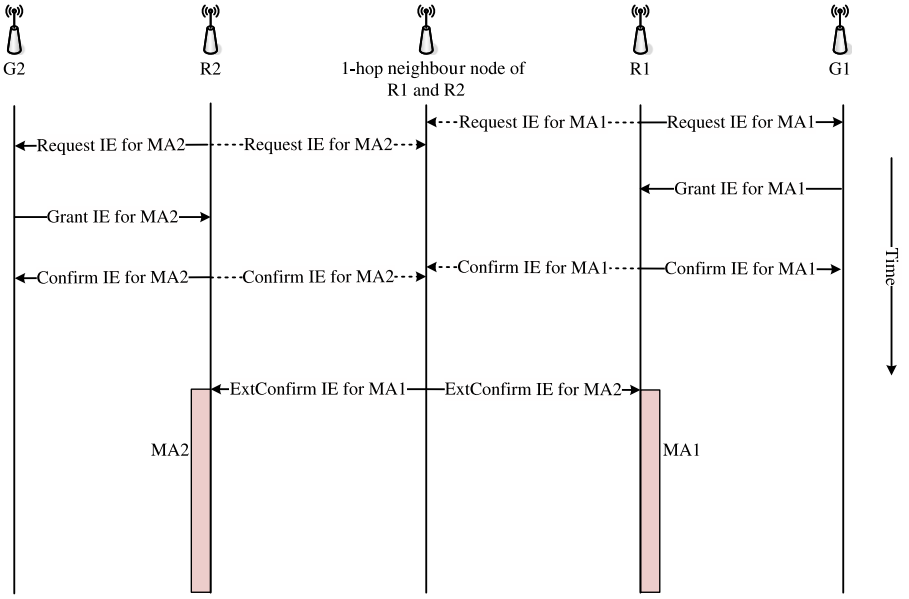

Fig. 9. A minislot scheduling example when ATEA is used.

can broadcast the corresponding extended confirm IE on its next MSH-DSCH TxOpp known by the requesting node. In such a condition, ATEA can know when node n will broadcast this extended confirm IE. By exploiting this information, ATEA can more precisely estimate when all of these neighboring nodes will finish broadcasting their extended confirm IEs. This is done by finding the largest TxOpp number among their next MSH-DSCH TxOpps already known by the requesting node.

On the other hand, if the next MSH-DSCH TxOpp of the requesting node is smaller than that of the granting node, the requesting node is not sure when it can transmit the confirm IE out because it cannot know its "next next" MSH-DSCH TxOpp at the current point of time. Although the information is not complete, ATEA still chooses the largest MSH-DSCH TxOpp number among the granting node's next MSH-DSCH TxOpp and those of all its one-hop neighboring nodes as the estimated TxOpp num-

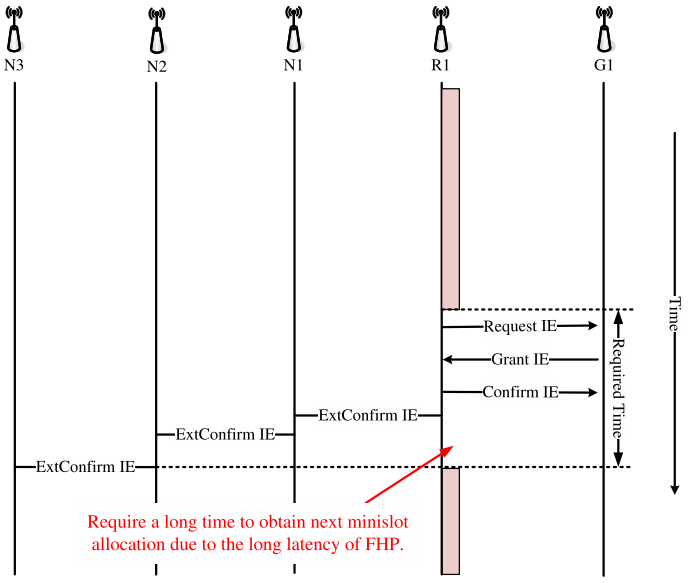

Fig. 10. Why FTA is needed.

ber and then uses the *fr* function to compute the number of the frame containing the estimated TxOpp number as its output. In such a case, ATEA cannot precisely determine the best activation time for the requested minislot allocation. However, it is still useful to mitigate the EHT problem by allowing more nodes to know the activation time of this minislot allocation before it is activated.

5.4. Timing control for starting an FHP

The second problem of FHP-SCRA is that the bandwidth is under-utilized due to its long latency. As one knows, for a requesting node, the best timing to activate its next minislot allocation (denoted as MA) is after all its one-hop neighboring nodes (except for the granting node) have broadcast their extended confirm IEs for MA. As shown in Fig. 10, however, using such a decision a requesting node has to wait a long time before its next minislot allocation can be set up. (Note that in Fig. 10, nodes N1, N2, and N3 are all one-hop neighboring nodes of R1.) Such a design will cause the network bandwidth to be under-utilized. To solve this problem, we propose an FHP Triggering Algorithm (FTA), which can smartly trigger an FHP in advance based on the estimated completion time of an FHP (determined by ATEA proposed in the previous section). As shown in Fig. 11, by using FTA each node can smartly start an FHP so that the requested minislot allocation is guaranteed (1) to be activated as soon as possible after the existing minislot allocation has elapsed and (2) not to be activated when the existing minislot allocation is still valid. In short, using FTA each node can effectively hide the long latency of FHP from traffic flows to increase bandwidth utilization.

The pseudo code of FTA is shown in Algorithm 6 and explained below. (Note that the pseudo code of FTA shown here is for illustration purpose only. The details of FTA for solving the frame number wrapping problem are omitted in this paper for brevity.)

Algorithm 6. The FHP Triggering Algorithm for node i

Require: when node i is allowed to transmit an MSH-DSCH message

frame_cur := the current frame number
MA_{req} := the minislot allocation to be requested
N_{bits_q} := the number of data bits currently buffered in the output queue
if there has been an active minislot allocation MA_{act} **then**
  N_{frame_r} := the number of frames that MA_{act} remains valid
  N_{bits_r} := the number of data bits that can be transmitted using the remaining minislots of MA_{act}
  N_{frame_dist} := ATEA (MA_{req}) − frame_cur
  if $N_{frame\_r} \leqslant N_{frame\_dist}$ and $N_{bit\_r} < N_{bit\_q}$
    Start an FHP for MA_{req}
  end if
else
  if $N_{bit\_q} > 0$ **then**
    Start an FHP for MA_{req}
  end if
end if
Return

For node i, each time when it is allowed to transmit an MSH-DSCH message, FTA first examines whether an active minislot allocation (denoted as MA_{act}) exists. If not, it then checks whether it has any data send. If so, FTA immediately starts an FHP. Otherwise, it simply returns. On the other hand, if MA_{act} exists, node i first uses ATEA to compute the estimated starting frame number of the minislot allo-

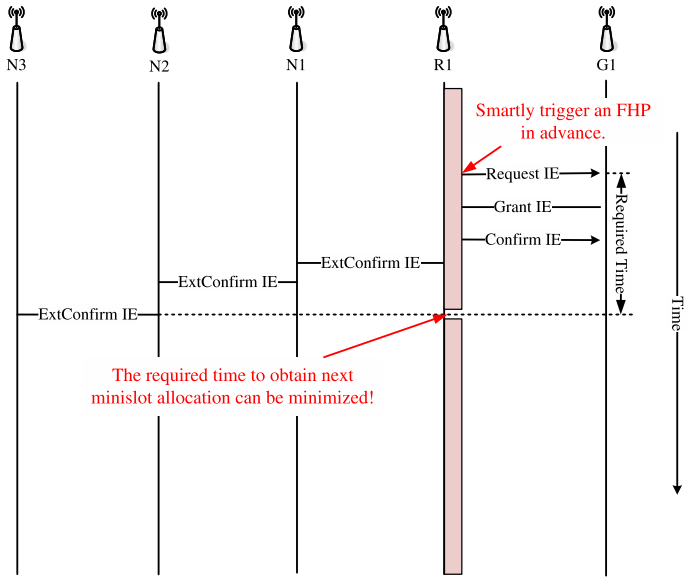

Fig. 11. A minislot allocation scheduling example when FTA is used.

cation to be requested (denoted as ATEA (MA_{req})) and then compute the frame distance between the current frame and ATEA (MA_{req}) (denoted as N_{frame_dist}). It then compares the number of MA_{act}'s remaining frames and N_{frame_dist}. If (1) the former is smaller or equal to the latter and (2) the remaining scheduled bandwidth cannot accommodate the transmission of the data currently buffered in the output queue, FTA will immediately start an FHP to obtain next minislot allocation as soon as possible. Otherwise, it defers starting the next FHP to avoid generating an MA_{req} that overlaps MA_{act}.

In summary, by using ATEA each node can more properly set the activation time of requested minislot allocations to reduce bandwidth wastage and by using FTA each node can start an FHP at a better time to increase bandwidth utilization. Doing so can achieve better performances than FHP-SCRA while effectively eliminating HTs and EHTs.

6. Performance evaluation

In this section, the data forwarding performances of three schemes are studied. The first one is the original routing-based 802.16(d) mesh-mode network (denoted as STD); the second one is our proposed rule-based network coding with the original THP design (denoted as RNC-THP); the last one is the RNC using the proposed SHP design (denoted as RNC-SHP). We use the NCTUns network simulator [6] to evaluate the performances of the three schemes. In addition, we built an analytical model to derive the optimal coding performance in IEEE 802.16 mesh CDS-mode networks in Section 6.1. Such theoretical performance results are compared with those of the three evaluated schemes.

6.1. Analytical model

The IEEE 802.16(d) mesh CDS-mode network uses a distributed handshake process to negotiate for the bandwidth resource at the MAC layer in an on-demand manner. In addition, the MAC layer of the 802.16(d) mesh CDS mode can merge/fragment upper-layer packets in a data frame. As a result, data transmissions carried out at the MAC layer are bursty. The time required for a node to complete the handshake process varies depending on several factors: the holdoff times of one-hop and two-hop nodes, the number of competing nodes, and the current frame number. Let Th_x be a random variable denoting the time required for node x to complete the handshake procedure once. The optimal performance of pairwise network coding can be considered as follows.

We use the 3-node chain network shown in Fig. 12 as an example to explain the relationship between the minislot scheduling of the forwarding node (node 2) and those of the involved traffic source nodes (nodes 1 and 3). As shown in Fig. 15, the data transmissions of a forwarding node can be divided into five intervals: I_Q, I_{NC}, I_{D1}, I_{D3}, and I_{idle}.

In the interval I_Q, node 2 is not ready for forwarding data but either node 1 or node 3 is transmitting data. Thus, node 2 has to queue the received packets within this interval and then transmit them out after it has established a minislot scheduling. In the interval I_{NC}, all of the three nodes are ready for transmitting/forwarding data. In this condition, node 2 can perform the packet mixing operation and broadcast coded packets. In the interval I_{D1} (or I_{D3}), only node 1 (or node 3) and node 2 are ready for transmit-

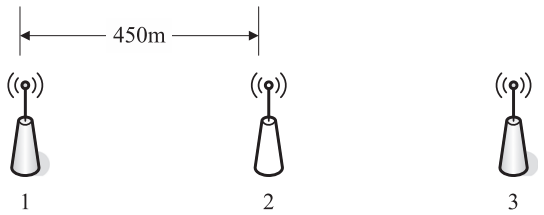

Fig. 12. The 3-node chain network used in our simulations.

ting data. In this condition, node 2 transmits the fresh received packets sent by node 1 (or node 3) without packet mixing due to the lack of the peer flow's packets. In the interval I_{idle}, neither node 1 nor node 3 is ready for transmitting data. Thus, node 2 is either idle within this interval (if no data needs to be forwarded) or forwards the remaining data in the interface output queue (if there are packets queued in the interface output queue). The packet transmitted by node 2 during the interval I_{idle} can be coded or un-coded, depending on whether coding opportunities exist among the queued packets.

Denote G_x as the average throughputs of node x. From the above observations, the average throughputs of node 2 is defined as follows:

$$G_2 = \frac{D_{NC} + D_{N1} + D_{N3} + D_{idle}}{Th_2 + T_{ds}}, \tag{1}$$

where Th_2 is a random variable denoting the time required for node 2 to obtain a minislot allocation. T_{ds} is the duration of a minislot allocation and fixed to 128 frame in this paper. D_{NC}, D_{N1}, D_{N3}, and D_{idle} denote the amounts of data that the forwarding node can transmit out in the intervals I_{NC}, I_{N1}, I_{N3}, and I_{idle}, respectively.

Let D_{fx} be the amount of data that can be transmitted by node x within a frame, when its minislot allocation is activated, and, let D_{mac} be the maximum amount of data that can be theoretically transmitted by a node within a frame, based on the number of obtained minislots per frame and the used modulation/coding scheme. For example, given that the modulation/coding scheme used by nodes is QAM-64-3/4, D_{mac} is:

$$D_{mac} = N_s * 3 * 108 \text{ (byte)}, \tag{2}$$

where N_s is the number of minislots allocated to this node in one frame. Let l_q be the length of a forwarding node's interface output queue in units of packets. When the optimal coding performance is concerned, l_q and D_{mac} are assumed to hold the following relationship:

$$l_q \geqslant \text{ceil}\left(E\left(\frac{D_{mac}}{l_{pkt}}\right) * 2\right), \tag{3}$$

where l_{pkt} is the average size of packets transmitted by flows in the network. Satisfying Eq. (3) means that, if the interface output queue of a forwarding node is empty at the beginning of a frame, the order of the minislot allocations of the traffic source nodes and forwarding nodes within a frame will not generate undesired packet losses due to the exhaustion of the interface output queue space. For example, for a network with N_s set to 50 minislots and the average packet size of flows being 1000 bytes, setting l_q larger than 38 is sufficient to avoid such undesired packet losses.

In this example network, node 2 is the forwarding node and D_{f2} is equivalent to D_{mac}. D_{NC} is thus given as follows:

$$D_{NC} = \begin{cases} 2 * D_{f2} * T_{NC}, & \text{if } D_{f1}, D_{f3} \geqslant D_{f2}, \\ 2 * D_{min} * T_{NC} + min(D_{max} - D_{min}, D_{f2} - D_{min}) * T_{NC}, & \text{if } D_{f1} < D_{f2} \text{ or } D_{f3} < D_{f2}, \end{cases} \tag{4}$$

where D_{max} is max (D_{f1},D_{f3}) and D_{min} = min(D_{f1},D_{f3}). Because a forwarding node starts to establish a minislot allocation only after it has data to forward, as shown in Fig. 15, the triggering of its THP/SHP is usually later than those of the THPs/SHPs of the traffic source nodes. On the other hand, the triggering of THPs/SHPs of nodes 1 and 3 can be very close (but cannot occur at the same time due to the design of the IEEE 802.16(d) mesh CDS mode). Let x be the random variable denoting the number of frames that Th_1 and Th_3 overlap. Then, D_{N1} and D_{N3} can be defined as follows:

$$D_{N1} = min(D_{f1}, D_{f2}) * (Th_2 + (Th_3 - x) - Th_1'). \tag{5}$$

$$D_{N3} = min(D_{f3}, D_{f2}) * (Th_3 - x). \tag{6}$$

The amount of data transmitted in I_{idle} can be considered as those packets buffered in the node 2's output queue, when node 2's minislot allocation is not ready. Thus, D_{idle} can be defined as follows:

$$D_{idle} = min(\{2 * min(l_{q1}, l_{q3}) + [max(l_{q1}, l_{q3}) - min(l_{q1}, l_{q3})]\} * l_{pkt}, T_{idle} * D_{f2}), \tag{7}$$

where T_{idle} is given as follows:

$$T_{idle} = min(Th_2 - (Th_3 - x), Th_1' - (Th_3 - x)). \tag{8}$$

The upper bound of D_{idle} is given as follows:

$$D_{idle} \leqslant min\left(\frac{l_q}{2} * l_{pkt} * 2, T_{idle} * D_{f2} * 2\right) = min(l_q * l_{pkt}, 2T_{idle} * D_{f2}). \tag{9}$$

Because flow 1 and flow 3 are activated at the same time in our experiments, nodes 1 and 3 trigger their respective THPs/SHPs nearly at the same time. Suppose that, in this condition, x is uniformly distributed from 0 to T_{ht}. The expected value of x is thus $\frac{\sum_{x=0}^{T_{ht}} x}{T_{ht}+1}$.

Denote p_{xy} as the probability of node x's data received by node y and $p_{x\to yz}$ as the probability of node x's data simultaneously received by nodes y and z. Assuming that the IDs of nodes in a n-node chain network are $\{1,\ldots,n\}$ in the increasing order from the left to the right, the optimal end-to-end aggregate goodputs for the n-node chain network is given as follows:

$$G_{chain} = min(G_2, \ldots, G_i, G_{n-1}) * (1 - R_{ql}), \quad 2 < i < n - 1, \tag{10}$$

where R_{ql} denotes the end-to-end goodput loss ratio due to the fluctuation of neighboring nodes' minislots obtained per data schedule (caused by the on-demand distributed handshake process), and, the amounts of data that can be transmitted by the two end nodes (node $i-1$ and node $i+1$) per frame for a forwarding node i are given as follows:

$$D_{f(i-1)} = min(F_{i-1}, D_{mac}) * P_{(i-1)i}, \tag{11}$$

$$D_{f(i+1)} = min(F_{i+1}, D_{mac}) * P_{(i+1)i}, \tag{12}$$

where F_x denotes the amount of data transmitted by the flow on node x. For example, the amounts of data that can be transmitted by the two end nodes (i.e., nodes 1 and n) are as follows:

$$D_{f1} = min(F_1, D_{mac}) * P_{12}, \tag{13}$$

$$D_{fn} = min(F_n, D_{mac}) * P_{n(n-1)}, \tag{14}$$

In the 802.16 mesh CDS-mode network, the 3-node chain topology is a special case of chain network topologies. Because only one coding structure exists, the success of one-hop data transmissions can be guaranteed. As a result, no packet collisions occur in the 3-node chain topology and the values of P_{12} and P_{32} can achieve 1.0. In addition, the only node that needs to buffer packets is node 2. As the length of node 2's interface queue is sufficient to buffer all incoming packets, no packets will be dropped due to fluctuation of neighboring nodes' MAC-layer throughputs. In this condition, R_{ql} is zero and the optimal end-to-end aggregate goodputs in a 3-node chain network is G_2.

Deriving R_{ql} in more complicated chain topologies is difficult because it depends on the used minislot selection algorithm, the number of requested minislots, and the range of the availability IE selected by the requesting node in a THP/SHP. For this reason, we estimate the value of R_{ql} in the 7-node chain network using the simulation results of the STD scheme as follows:

We first logged the average number of minislots obtained by nodes per THP in the simulations of 7-node chain networks with the STD scheme. Based on this information, we derived the theoretical application throughputs as a. We then computed the average of the aggregate end-to-end flow goodputs in the network as b in these simulations. Finally, we obtained $\frac{a}{b}$, which is equivalent to R_{ql} in chain networks with the STD scheme. The reason is that in the STD scheme THPs guarantee the success of one-hop data transmission. Thus, the main difference between a and b is due to queuing losses, resulting from the fluctuated MAC-layer throughputs among neighboring nodes.

In the butterfly topology, the optimal end-to-end aggregate goodputs can be defined as follows:

$$G_{btfy} = min(G_2, G_3) * P_{1\to 24} * P_{3\to 26} * (1 - R_{ql}). \quad (15)$$

In the butterfly topology, only nodes 2 and 5 need to buffer packets. Thus, we assume that R_{ql} can be ignored, when the interface queues of these two nodes are large enough to absorb temporary data bursts. For nodes 1 and 3, the successes of delivering their packets on nodes 2, 4, and 6 are independent of each other. Thus, $P_{1\to 24} = P_{12}P_{14}$ and $P_{3\to 26} = P_{32}P_{36}$. In addition, because nodes using SHPs can avoid scheduling conflicts by the use of extended confirm IEs, $P_{12}P_{14}$ and $P_{32}P_{36}$ are assumed to be 1. That is, G_{btfy} is mainly bounded by $min(G_2, G_3)$.

The optimal coding goodput results in these topologies are plotted as the Opt curves in the following performance plots and compared with those of the RNC-THP and RNC-SHP schemes obtained using simulations. For the optimal coding performances in more complicated network scenarios with four more unicast flows (e.g., the 25-node grid network scenario that will be studied in Section 6.5), we leave its analyses as our future work because the performance of inter-session coding in such complicated scenarios is still an open problem and difficult to characterize [9,18].

6.2. Simulation settings

The main parameters used in our simulations are listed in Table 6. The settings used in our simulations are explained here. In each simulation case, each traffic flow starts generating traffic after all network nodes have attached themselves to the network and the generated traffic lasts for 50 s. For each flow, the frequency of transmitting a packet is determined by the used Poisson arrival process. The mean packet generating frequency of the Poisson arrival process in each simulation scenario differs from each other and will be presented later. The payload length of each data packet is fixed to 1400 bytes in Sections 6.4 and 6.5 and varied in Section 6.6. The routing paths of each node in each simulation case are statically determined using Dijkstra's shortest path algorithm. The channel model used in the simulations is the two-ray model.

To fairly share link bandwidth, each node requests $\left\lceil \frac{N_{rs}}{|nbr_1(n)|} \right\rceil$ minislots per data schedule, where N_{rs} is the number of minislots per frame requested by node n and set to different value in different scenarios. $|nbr_1(n)|$ is the number of node n's one-hop nodes. Following this assignment, each node maximally uses N_{rs} minislots per frame.

6.3. Performance metrics

Two performance metrics are used to evaluate the performances of the proposed schemes, which are explained below.

6.3.1. Aggregate of average UDP flow throughputs (AAUFT)

The AAUFT metric is used to evaluate the throughput performance of a network under different schemes and defined as follows:

$$\text{AAUFT} = \sum_{i=1}^{N} t_i, \quad (16)$$

t_i denotes the average throughput achieved by the ith UDP flow in a simulation run. t_i is defined as $\frac{\sum_{s=n1}^{n2} th_s^i}{N}$, where th_s^i denotes the throughput logged by the receiver program of the ith UDP flow at the sth second. $n1$ denotes the traffic start time and $n2$ denotes the traffic stop time in a simulation run.

6.3.2. Average end-to-end packet delay (AEPD)

The AEPD metric is defined as follows:

$$\text{AEPD} = \frac{\sum_{i=1}^{N} d_i}{N}, \quad (17)$$

where d_i denotes the average packet delay experienced by the ith UDP flow in a simulation run and N denotes the to-

Table 6
The parameter setting used in the simulations.

Parameter name	Value
Number of TxOpps per frame	8
Number of Minislots per frame	220
Number of OFDM symbols per mini-slot	3
Requested frame length	128
Modulation/coding scheme	64QAM-3/4
Frame duration	10 ms
Fresh packet expiry time	10 s
Coded packet expiry time	5 s
Radio transmission range	500 m

tal number of UDP flows in the simulated network. d_i is calculated as follows:

$$d_i = \frac{\sum_{j=1}^{m} \text{pkt\_delay}(j)}{m}, \quad (18)$$

where pkt_delay(j) denotes the end-to-end delay of the jth packet received by flow i and m denotes the total number of packets received by flow i. Similar to AAUFT, each AEPD result presented in this paper is the average across ten simulation runs, each using a different random number seed. To avoid ambiguity, the calculation of AEPD does not take the end-to-end delay of a packet that cannot reach its destination node (e.g., being dropped by intermediate nodes due to exhausted buffer space or being collided with other packets) into consideration.

6.4. Simulation results in two classic scenarios

We first evaluate the performances of the proposed schemes in two classic scenarios in the network coding theory. The first scenario is called the Alice-and-Bob scenario, which uses a 3-node chain network shown in Fig. 12. In this chain network, each node is spaced 450 m away from each other.

Two UDP flows generate data using the Poisson arrival process with mean packet generating rate of 1 packet per 0.5 ms. One flow is from node 1 to node 3 and the other is from node 3 to node 1. In this scenario, N_{rs} is set to 40, which generates a moderate traffic load in the network. The average of minislots obtained by each node per frame in the simulations is 37.8 (minislots).

The second scenario is called the butterfly scenario, which adopts a 6-node grid network shown in Fig. 13. Similarly, in this network each node is spaced 450 m away from its neighboring nodes. Two flows are set up to generate data using the Poisson arrival process with mean packet generating rate of 1 packet per 1 ms. One flow is from node 1 to node 6 and the other is from node 3 to node 4. In this scenario, N_{rs} is set to 40 as well and the average of minislots obtained by nodes per frame is 33.5.

The traffic specified in the Alice-and-Bob scenario can trigger RCU to use chain structure coding to transmit packets and that specified in the butterfly scenario can trigger RCU to use butterfly structure coding to do it. Thus, the simulation results obtained in these two scenarios can be used to observe the efficiency of the chain structure coding and the butterfly structure coding with our proposed schemes.

Figs. 16 and 17 show the AAUFT results of the four evaluated schemes under the Alice-and-Bob and butterfly scenarios, which can be discussed from three aspects. First, our proposed RNC-based schemes can greatly outperform the routing-based scheme in terms of end-to-end flow goodputs. Second, because in each of these two scenarios only one coding structure exists in the network, no EHTs exist in the network. Thus, the performances of RNC-SHP degenerate to those of RNC-THP in these two scenarios.

Thirdly, in these two simple scenarios both of the RNC-based schemes can achieve the theoretically optimal flow goodputs, when the output queue lengths of forwarding nodes are large enough to store all incoming packets in the interval I_Q.

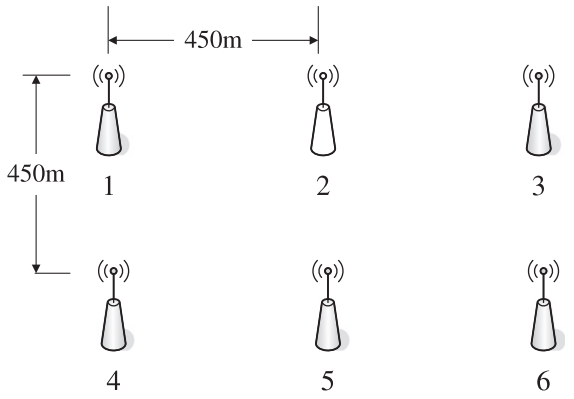

Fig. 13. The 6-node grid network used in our simulations.

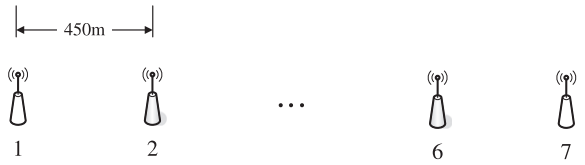

Fig. 14. The 7-node chain network topology.

The AEPD results shown in Figs. 18 and 19 also show that the RNC-based schemes can effectively increase network capacity, as compared with the routing approach. As one sees, the average end-to-end packet delay of a network using the STD scheme significantly increases when the maximum MAC-layer output queue length increases. In contrast, when the maximum MAC-layer output queue length increases, the average end-to-end packet delays of networks using the two RNC-based schemes increase at lower rates. This also shows that a network using the proposed RNC-based schemes can drain packets faster than that using the traditional routing scheme.

6.5. Simulation results in more complicated scenarios

In this section, we further study the performances of the proposed schemes under two more complicated scenarios where EHTs are likely to occur. One scenario uses a 7-node chain network topology shown in Fig. 14. In this scenario, two UDP flows generate data using the Poisson arrival process with mean packet generating rate of 1 packet per 0.5 ms. One flow is from node 1 to node 7 and the other is from node 7 to node 1.

The other scenario uses a 25-node grid network topology shown in Fig. 20. In this scenario, 16 flows generate data packets using the Poisson arrival process with mean packet generating rate of 1 packet per 1 ms. The source and destination node pairs of the eight flows among them are (2,14), (3,25), (4,24), (5,23), (6,22), (7,21), (12,16) and (11,17), respectively, while those of the remaining flows are the reverse of the former. In these two scenarios, multiple coding structures simultaneously exist. This arrange-

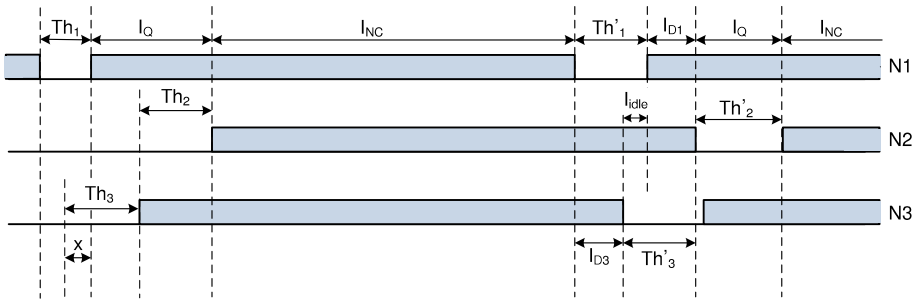

Fig. 15. The five intervals of the data transmissions of a forwarding node.

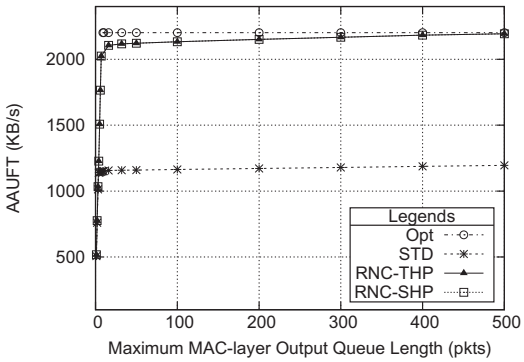

Fig. 16. The AAUFT results in the Alice-and-Bob scenario.

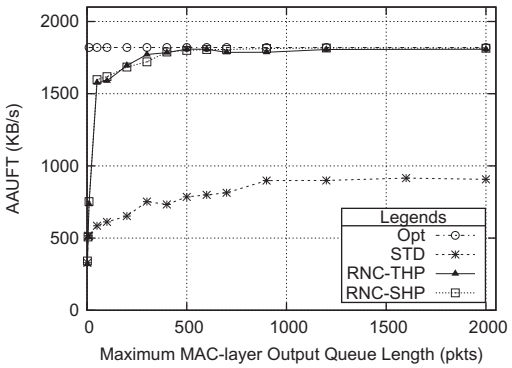

Fig. 17. The AAUFT results in the butterfly scenario.

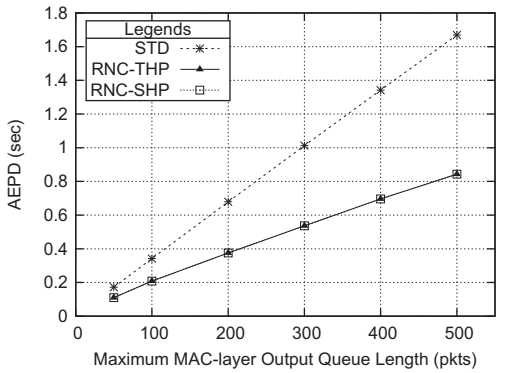

Fig. 18. The AEPD results in the Alice-and-Bob scenario.

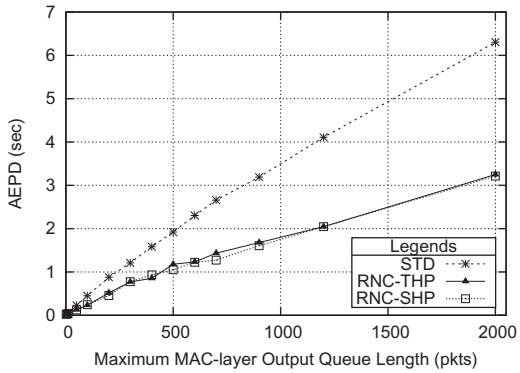

Fig. 19. The AEPD results in the butterfly scenario.

ment can result in many EHT node pairs in the simulated networks and cause the performances of network coding to degrade.

In these two scenarios, N_{rs} is set to $\frac{220}{max(\{nbr_1(i)\})}$ minislots, $\forall$node i in the network, such that one-hop neighboring nodes can fairly share and use up all the link bandwidth. Note that, due to the distributed handshake design of THP/SHP and the existence of two-hop interfering nodes, the actual MAC-layer throughput obtained by each node can greatly fluctuate over time. For example, the average of minislots obtained by nodes per frame in the 7-node chain is only 31.511, far below the value of N_{rs} (i.e., 74 in this scenario).

As shown in Fig. 21, networks using the RNC-based schemes on average outperform those using the STD scheme on end-to-end flow goodputs in the 7-node chain network. Compared with those using the STD scheme, networks using the RNC-SHP scheme and the RNC-THP scheme on average can increase the end-to-end flow goodputs by factors of 1.5 and 1.33, respectively. The reason why the RNC-SHP scheme cannot achieve the theoretical optimal performance is explained below.

As explained in Section 5.3, the ATEA algorithm used in the RNC-SHP scheme cannot precisely derive the best activation time for a minislot allocation in all cases. Some

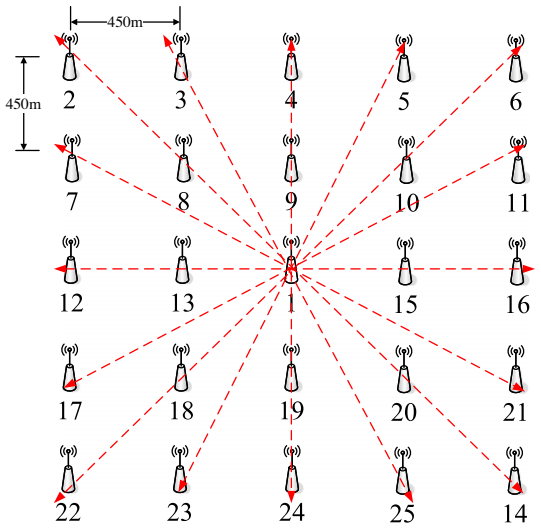

Fig. 20. The 25-node grid network topology in which a dotted arrow represents a UDP flow.

goodput losses are thus inevitable. However, as can seen in Fig. 21, although ATEA cannot generate precise estimation results at all time, the RNC-SHP scheme still achieve better end-to-end goodputs results than the RNC-THP scheme. We also logged the number of packet collisions in these simulations. The packet collision results show that, the RNC-SHP scheme on average can reduce the numbers of packet collisions to only 25.87% of those when the RNC-THP scheme is used, which shows that the RNC-SHP scheme can effectively alleviate the EHT problem.

Fig. 22 shows the AAUFT results of the evaluated schemes in the 25-node grid network scenario. The performance trends of the evaluated schemes in this scenario are similar to those in the 7-node chain scenario, except that, when the output queue length of a node exceeds 1000 packets, the end-to-end goodputs of the RNC-based schemes drastically decrease. This phenomenon is reasonable. As explained in Section 3.3, in our implementation each received packet is associated with an expiry timer and will be removed after its associated timer expires. Thus, if the queuing delay of a coded packet is larger than

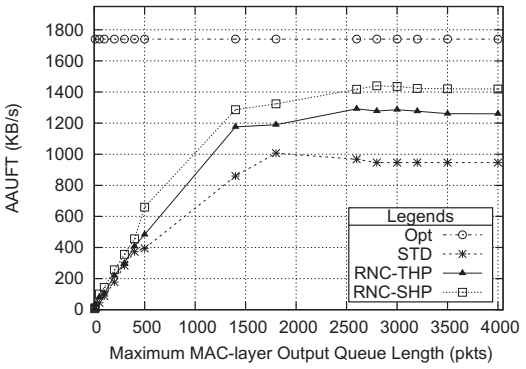

Fig. 21. The AAUFT results in the 7-node chain network (output queue length: 5–4000).

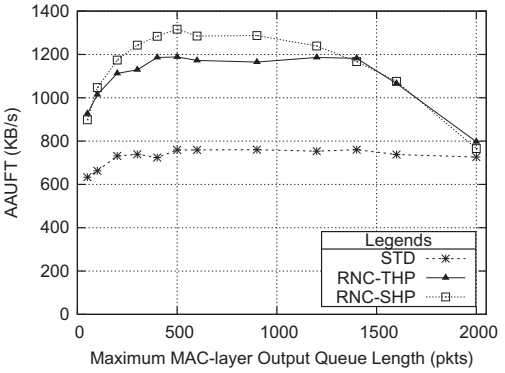

Fig. 22. The AAUFT results in the 25-node grid network.

the lifetime of its corresponding fresh (remedy) packet, when this coded packet arrives at the destination node, its corresponding fresh packet has been removed by the destination node. In this condition, these *late* coded packets consumes a large amount of MAC-layer bandwidth; however, the payloads contained in these packets cannot be recovered by the destination node, resulting in the significant decrease of end-to-end flow goodputs.

We have shown that different topology structures can affect the performance of a network coding scheme. It can be expected that the AAUFT results of the proposed RNC schemes can greatly vary in general networks. We studied this issue here. The performance of the proposed stateless RNC schemes in different topologies should be discussed case by case. To make the size of the discussion manageable, instead of creating a great number of topologies and generating results that should be separately discussed, we chose to randomly generate different routing paths for each node in the created 25-node grid network. It is known that routing paths in a network also affects the network coding performance by logically specifying the end-to-end relationship among nodes.

Fig. 23 shows the AAUFT speedup results in the 25-node grid network using 5 random route sets. The AAUFT speedup of scheme x is defined as the AAUFT value of x divided by that of the STD scheme. It is expected that the AAUFT speedup of RNC-SHP and RNC-THP greatly varies under different route sets because different route sets create different coding structures in the same network and generate different occupancy levels of interface output queues at each node. In general, the AAUFT speedup of RNC-SHP is significant and more stable than that of RNC-THP. In the case of the route set 5, where the flow traffic along the routes may exhaust some node's interface output queue space or cannot create many coding structures, the RNC-THP scheme may perform worse than the STD scheme due to the extra overheads of added network coding headers. In contrast, by exploiting ATEA and FTA, the RNC-SHP scheme can more effectively utilize link bandwidth and still slightly outperform the STD scheme. The performance of RNC-SHP and that of RNC-THP may be the same, if the route set of the network does not generate many EHTs.

We also plotted the AEPD results in these two scenarios in Figs. 24 and 25. The RNC-based schemes achieve greatly

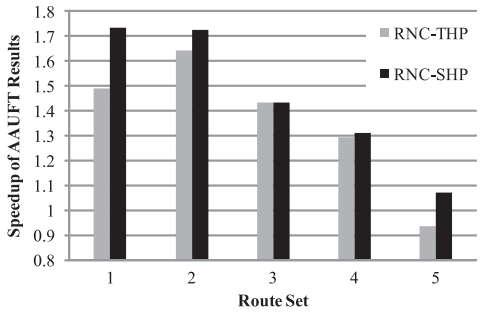

Fig. 23. The AAUFT speedup in the 25-node grid network with 5 randomly chosen route sets.

lower end-to-end packet delays than traditional routing. The end-to-end packet delay results are linear to the MAC-layer output queue length, which shows that the queuing delay of a packet dominates the overall end-to-end packet delay. One may notice that, in the 25-node grid scenario, the end-to-end packet delay decreases when the output queue length increases from 1600 (pkts) to 2000 (pkts). This is because the packet expiry timer is set to 10 s in our simulations. Thus, coded packets that reach their destination nodes with end-to-end delays larger than 10 s cannot be recovered. In this condition, only fresh packets (transmitted without network coding) and those coded packets with end-to-end delay less than 10 s can be delivered to user-level applications. Thus, the end-to-end packet delays experienced by applications are roughly bounded by the deadlines of packet expiry timers.

6.6. Coding performances with varied packet sizes

Let the lengths of packets transmitted by flows follow the exponential distribution with the mean value 1000 (bytes), the minimum value 50 (bytes), and the maximum value 1400 (bytes). The end-to-end flow goodput results of the evaluated schemes are plotted in Figs. 26 and 27. As can be expected, the forwarding efficiency of RNC-based schemes decreases due to the use of padding bits in a coded packet, if the lengths of its two component packets differ. However, because network coding can make nodes drain packets faster than traditional routing, the two

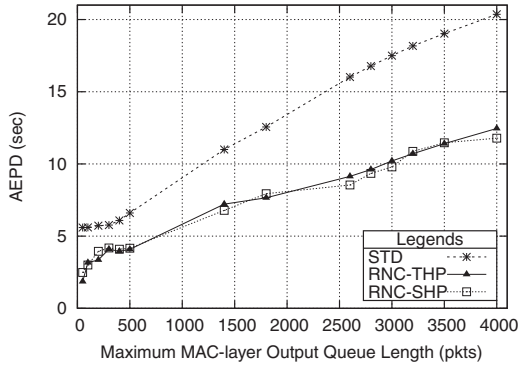

Fig. 24. The AEPD results in the 7-node chain network.

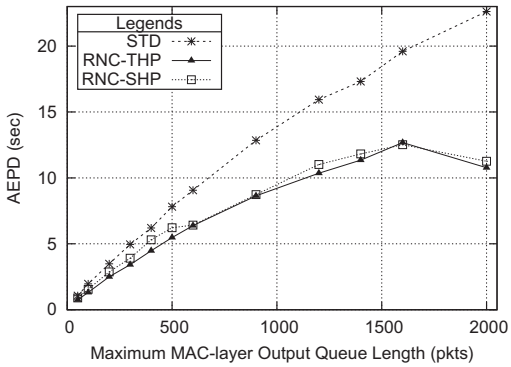

Fig. 25. The AEPD results in the 25-node grid network.

RNC-based schemes still outperform the STD scheme on end-to-end flow goodputs.

We finally studied which rule of SCRA is most frequently used to determine the winning node of an overlapped minislot. Table 7 shows the ratios of the occurrences of the minislot scheduling conflicts resolved by the three rules of SCRA to the occurrences of the total minislot scheduling conflicts. As one sees, most of the

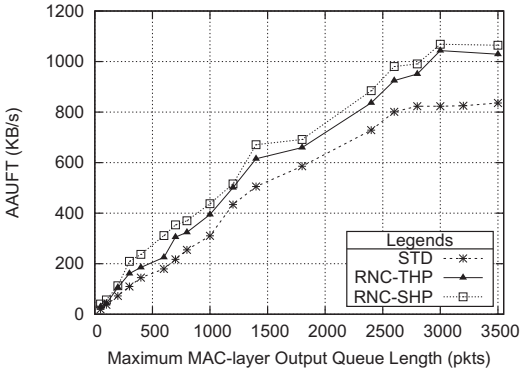

Fig. 26. The AAUFT results in the 7-node chain network with exponentially distributed packet size.

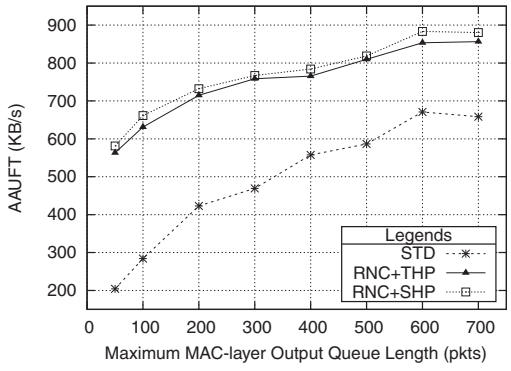

Fig. 27. The AAUFT results in the 25-node grid network with exponentially distributed packet size.

Table 7
The probabilities of the rules used in SCRA.

	Starting frame comparison (%)	Starting slot number comparison (%)	Node ID comparison (%)
7-node chain network	93.31	6.65	0.03
25-node grid network	97.97	2.00	0.04

scheduling conflicts were resolved by the first starting frame number comparison and the second starting minislot number comparison. Rare scheduling conflicts were resolved using the third node ID comparison. These empirical results show that the third rule has insignificant impacts on the fairness of resolving scheduling conflicts and most of the scheduling conflicts are resolved by the first two rules. Thus, due to the randomness of nodes' minislot scheduling behavior, using the first and second rules to resolve minislot scheduling conflicts ensures that each node on average can evenly win the use of overlapped minislots.

7. Discussion

In this section, we discuss the applicability issue of our proposed RNC-based schemes. Our proposed RNC-based schemes can efficiently increase network goodputs in an opportunistic manner. In some special network topologies, however, such RNC-based schemes may generate the "routing loop" problem, which can cause the routes of data packets to contain loops and thus data packets never have chances to reach their destination nodes. We use Fig. 28 to illustrate how this routing loop problem may occur in a special network topology, when RNC-based schemes are used.

Fig. 28 shows an example dumbbell topology, where nodes on the left and those on the right are connected only via a single route (formed by nodes 7–10). In this topology, for a flow whose source node is on the left and destination node is on the right, all of its packets have to be forwarded by nodes 7, 8, 9, and 10, in sequence.

Suppose that nodes 1 and 5 are transmitting packets to node 13 about the same time and the next-hop node of their routes to node 13 is node 3. In such a condition, node 3 will be triggered to perform butterfly structure coding to transmit the packets sent from nodes 1 and 5. In addition, to complete the butterfly structure coding, node 3 will perform route designation to set the next-hop node of node 1's packets to node 6 and that of node 5's packets to node 2, on behalf of node 4.

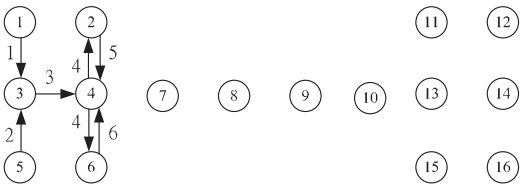

Fig. 28. An example dumbbell topology.

For the nodes on the left, however, there is only one route to reach node 13.[4] Thus, after being received by nodes 2 and 6, the packets of nodes 1 and 5 (transmitted by node 4) will be transmitted back to node 4 again, resulting in routing loops between nodes 2 and 4 and between nodes 6 and 4. In such a condition, packets transmitted by nodes 1 and 5 will never arrive at node 13 and thus the end-to-end goodputs of the two flows will drop to zero.

Fortunately, this problem can be easily solved in a mesh network where nodes are fixed and routes are not changed frequently. For a network using link-state based routing protocols, such as the Open Shortest Path First (OSPF) protocol and the Optimized Link State Routing (OLSR) protocol, each node can calculate and know the route information of other nodes. Thus, before performing the butterfly structure coding, it can check whether changing the routes of the next-hop node will generate routing loops. If not, it can safely use this butterfly structure to code packets. Otherwise, it should forbid itself from using this butterfly structure and try to use alternatives, such as using the chain structure coding or the traditional routing.

As for a network using non-link-state based routing protocols, such as the Dynamic Source Routing (DSR) protocol and the Ad-hoc On Demand Routing (AODV) protocol, this problem can be solved by sending a route-error message to the coding node, when a routing loop is detected. For example, after receiving the coded packets broadcast by node 4, nodes 2 and 6 can find that the decoded fresh packets should be forwarded to node 4 again, which is an indication of the occurrence of a routing loop. In this condition, they can send route-error messages to node 3, indicating that a routing loop is detected.

After receiving such a route-error message, node 3 can know which butterfly structures generate routing loops and thus stop using them. By using this learning mechanism, those butterfly structures that are likely to generate routing loops can be eliminated. Since a mesh network is usually deployed as a fixed core network, such detection and elimination processes are performed infrequently. Thus, the bandwidth overheads resulting from these processes and dropped data packets are insignificant.

8. Conclusion

In this paper, we propose a rule-based opportunistic inter-session network scheme (called RNC-SHP) based on IEEE 802.16(d) mesh CDS-mode networks. The proposed RNC-SHP scheme is easy-to-implement and need not maintain the states of traffic flows, which is usually required by previous work. Our simulation results show that the proposed RNC-SHP scheme can greatly outperform the traditional routing scheme on end-to-end flow goodputs and packet delays. In addition, our packet collision results show that the proposed RNC-SHP scheme can effectively

[4] The route from node 3 to node 13 is nodes 3, 4, 7, 8, 9, 10, and 13, in sequence.

reduce the number of extended hidden terminals to further increase network coding performances in an interference-prone environment.

Appendix A

In the Appendix, we use two example coding scenarios to compare the benefits of BRS and CRS on overall network throughputs. Consider an example network shown in Figs. A.29 and A.30, where six flows are generating data. The source–destination node pairs of the flows are (A,F), (C,D), (G,A), (H,C), (I,D), and (J,F), respectively. Fig. A.29 shows an example case where BRS is used in this network and Fig. A.30 shows an example case where CRS is used in this network. The concurrent transmissions allowed in these two example cases are shown in Table A.8. (The star sign denotes a transmission of the butterfly structure and the chain structure.)

As can be seen, BRS for flows (A,F) and (C,D) can be accomplished within 4 unit time slots and, by letting the transmissions A → D and C → F take place at the same unit time slot, CRS for flows (A,F) and (C,D) can be accomplished within 4 unit time slot as well. As shown in Table A.8, the number of concurrent packet transmissions in a period of 4 unit time slots in the BRS case is 10. However, because CRS essentially requires 5 packet transmissions to forward data in this example case (BRS requires 4 packet transmissions only), the number of concurrent packet transmissions in a period of 4 unit time slots in the CRS case is 8 only.

Suppose that the amount of data that can be transmitted at one unit time slot is B. The overall flow goodputs in the network in the BRS case is $\frac{10B+2B}{4} = 3B$ while that in the CRS case is $\frac{8B+2B}{4} = 2.5B$. Thus, BRS outperforms CRS on the maximum number of allowed concurrent packet transmissions in the whole network. However, because using BRS a coded packet should traverse two hops to reach its destination node, BRS is more vulnerable to network congestion. The reason is that, if the network is congested, packets are likely to be dropped at the interface output queue of a node. For BRS, if a coded packet is dropped on its way to the destination node (e.g., from node B to node E in Fig. A.29), not only the bandwidth used for the transmission B → E but those for the transmissions A → B and C → B are wasted. In addition, when the network is congested, a coded packet is more likely to reach its destination node with the end-to-end delay larger than

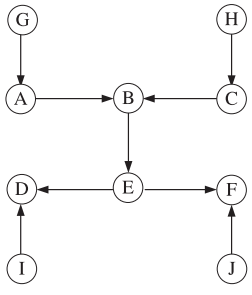

Fig. A.29. BRS.

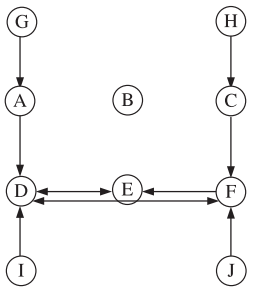

Fig. A.30. CRS.

Table A.8
Examples of the concurrent transmissions using BRS and CRS.

Tx node	BRS				CRS			
	*T*1	*T*2	*T*3	*T*4	*T*1	*T*2	*T*3	*T*4
A	*				*			
B			*					
C		*			*			
D						*		
E				*				*
F							*	
G		y	y	y		y	y	y
H	y		y	y		y	y	y
I		y	y				y	
J	y		y			y		

the deadline of its corresponding remedy packet. In this condition, even it arrives at its destination node, its contained payload cannot be recovered and delivered to the higher-layer applications. For these reasons, it is unnecessary that BRS always outperforms CRS on end-to-end flow goodputs, especially when the network is congested. However, BRS can be used as an alternative coding option when the opportunities of CRS do not exist.

References

[1] R. Ahlswede, N. Cai, S.-Y.R. Ki, R.W. Yeung, Network information flow, IEEE Transactions on Information Theory 46 (4) (2000) 1204–1216.
[2] S. Sengupta, S. Rayanchu, S. Banerjee, An analysis of wireless network coding for unicast sessions: the case for coding-aware routing, in: Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM 2007), Anchorage, Alaska, USA, May 6–12, 2007, pp. 1028–1036.
[3] A. Eryilmaz, D.S. Lun, Control for inter-session network coding, in: Proceedings of the Workshop on Network Coding, Theory and Applications, 2007.
[4] T. Ho, Y. Chang, K.J. Han, On constructive network coding for multiple unicasts, in: Invited Paper, 44th Allerton Conference on Communication, Control and Computing, 2006.
[5] A. Khreishah, C.-C. Wang, N. Shroff, Cross-layer optimization for wireless multihop networks with pairwise intersession network coding, IEEE Journal on Selected Areas in Communications 27 (5) (2009) 606–621.
[6] S.Y. Wang, C.L. Chou, C.C. Lin, The design and implementation of the NCTUns network simulation engine, Elsevier Simulation Modelling Practice and Theory 15 (2007) 5781.
[7] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Mdard, Jon Crowcroft, XORs in the air: practical wireless network coding, in: Proceedings of the Sigcomm 2006, Pisa, Italy, September 2006, pp. 243–254.
[8] S. Omiwade, R. Zheng, C. Hua, Butterflies in the mesh: lightweight localized wireless network coding, in: Proceedings of the 4th

Workshop on Network Coding, Theory, and Applications (Netcod), HongKong, China, 2008.
[9] S. Omiwade, R. Zheng, C. Hua, Practical localized network coding in wireless mesh networks, in: 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, 2008 (SECON '08), San Francisco, CA, USA, 16–20 June 2008, pp. 332–340.
[10] Anwar Al Hamra, Chadi Barakat, Thierry Turletti, Network coding for wireless mesh networks: a case study, in: Proceedings of the IEEE International Symposium on a World Networks (WoWMoM'06), 2006.
[11] P.S. Mogre et al., A note on practical deployment issues for network coding in the IEEE 802.16 MeSH mode, in: Proceedings of the 5th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2008 (SECON Workshops '08), San Francisco Bay Area, California, USA, 16-20 June, 2008, pp. 1–6.
[12] P.S. Mogre, N. d'Heureuse, M. Hollick, R. Steinmetz, CORE: centrally optimized routing extensions for efficient bandwidth management and network coding in the IEEE 802.16 MeSH mode, Wireless Communications and Mobile Computing 11 (2011) 338–356.
[13] C.-C. Wang, N.B. Shroff, Beyond the butterfly — a graph-theoretic characterization of the feasibility of network coding with two simple unicast sessions, in: Proceedings of the 2007 IEEE International Symposium on Information Theory, Nice, France, June 24–29, 2007.
[14] J. Jin, B. Li, Adaptive random network coding in WiMAX, in: Proceedings of the IEEE International Conference on Communications (ICC 2008), Beijing, China, May 19–23, 2008.
[15] X. Zhang, B. Li, Network coding aware dynamic subcarrier assignment in OFDMA wireless networks, in: Proceedings of the IEEE International Conference on Communications (ICC 2008), Beijing, China, May 19–23, 2008.
[16] IEEE Std 802.16-2004, IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems, 2004.
[17] S.Y. Wang, C.C. Lin, H.W. Chu, T.W. Hsu, K.H. Fang, Improving the performances o f distributed coordinated scheduling in IEEE 802.16 mesh networks, IEEE Transactions on Vehicular Technology 57 (4) (2008).
[18] M. Kim, M. Medard, U.-M. O'Reilly, D. Traskov, An evolutionary approach to inter-session network coding, in: INFOCOM 2009, Rio de Janeiro, Brazil, April 19–25, 2009, pp. 450–458.

Shie-Yuan Wang is a Professor of the Department of Computer Science at National Chiao Tung University, Taiwan. He received his Master and Ph.D. degree in computer science from Harvard University in 1997 and 1999, respectively. Before that, he received his Master degree in computer science from National Taiwan University in 1992 and his bachelor degree in computer science from National Taiwan Normal University in 1990. His research interests include wireless networks, Internet technologies, network simulations, and operating systems. He authors a network simulator, called NCTUns, which is now is a famous tool widely used by people all over the world.

Chih-Che Lin currently works for the Industrial Technology Research Institute of Taiwan. He received his BS degree and Ph.D. degree in computer science from National Chiao Tung University, Taiwan, in 2002 and 2010, respectively. He was a core team member of the NCTUns network simulator project during 2002 and 2010. His current research interests include wireless networking, wireless mesh networks, vehicular networks, intelligent transportation systems, and network simulation.

Yu-Chi Chang received his BS degree in computer science from Department of Computer Science, National Dong Hwa University, Taiwan, in 2006. He received his MS degree in network engineering from National Chiao Tung University, Taiwan, in 2008. He was a core team member of the NCTUns network simulator project during 2006 and 2002 and now works in High Tech Computer Corporation (HTC). His major research interests include wireless mesh networks and network simulation