TCP Network Coding with Enhanced Retransmission for Heavy and Bursty Loss

PAPER

TCP Network Coding with Enhanced Retransmission for Heavy and Bursty Loss

Nguyen VIET HA[†a)], ***Student Member***, **Kazumi KUMAZOE**[††b)], ***and*** **Masato TSURU**[†c)], ***Members***

SUMMARY In general, Transmission Control Protocol (TCP), e.g., TCP NewReno, considers all losses to be a sign of congestion. It decreases the sending rate whenever a loss is detected. Integrating the network coding (NC) into protocol stack and making it cooperate with TCP (TCP/NC) would provide the benefit of masking packet losses in lossy networks, e.g., wireless networks. TCP/NC complements the packet loss recovery capability without retransmission at a sink by sending the redundant combination packets which are encoded at the source. However, TCP/NC is less effective under heavy and bursty loss which often occurs in fast fading channel because the retransmission mechanism of the TCP/NC entirely relies on the TCP layer. Our solution is TCP/NC with enhanced retransmission (TCP/NCwER), for which a new retransmission mechanism is developed to retransmit more than one lost packet quickly and efficiently, to allow encoding the retransmitted packets for reducing the repeated losses, and to handle the dependent combination packets for avoiding the decoding failure. We implement and test our proposal in Network Simulator 3. The results show that TCP/NCwER overcomes the deficiencies of the original TCP/NC and improves the TCP goodput under both random loss and burst loss channels.

key words: *TCP, Network coding, enhanced retransmission, dependence retransmission, heavy loss, bursty loss*

1. Introduction

The transmission control protocol (TCP) remains the dominant transport protocol for reliable end-to-end data transmission. However, its performance is considerably degraded in lossy networks for two main reasons. First, loss-based congestion control recognizes packet losses as a network congestion signal although non-congestion origin packet losses often occur in lossy networks. Thus, TCP reduces the sending rate mistakenly. Second, long latency often occurs in wireless networks that results in a large round trip time (RTT) in the TCP process, which yields a slow recovery from the reduced sending rate and packet loss. To overcome these problems, while a wide array of wireless network-oriented TCP variants have been proposed, e.g., TCP Westwood+; a different promising approach, TCP with network coding (TCP/NC), has also been presented [1] which is more dominant than TCP variants (mentioned in Sect. 4).

Manuscript received March 14, 2016.
Manuscript revised July 4, 2016.
Manuscript publicized August 9, 2016.

[†]The authors are with the Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology, Iizuka-shi, 820-8502 Japan.
[††]The author is with the Network Design Research Center, Kyushu Institute of Technology, Iizuka-shi, 820-8502 Japan.
a) E-mail: nvha@infonet.cse.kyutech.ac.jp
b) E-mail: kuma@ndrc.kyutech.ac.jp
c) E-mail: tsuru@cse.kyutech.ac.jp

DOI: 10.1587/transcom.2016EBP3101

TCP/NC complement the packet loss recovery capacity by allowing the source to send m combination packets (referred to as combinations) created from n original packets with $m \geq n$. We expect that a sink can recover the lost packets without retransmission by using the remaining combinations. The combination and the recovering processes are called encoding and decoding. The main advantages of TCP/NC are a high degree of robustness to packet loss and potential throughput improvement for TCP. Within the scope of the current study, TCP/NC is considered as a type of TCP with a forward error correction coding. Nevertheless, we use the term TCP/NC here according to existing studies. Some studies consider the encoding and the decoding processes at the end nodes and the relay nodes [2]. In this paper, we focus only on processing at the source and the sink, not at the intermediate nodes.

In [3], the authors proposed a basic model of TCP/NC and its architecture. They added a new layer between the TCP and internet layer called NC layer shown in Fig. 1. Due to the fact that TCP/NC is a type of block coding, it divides all original packets to smaller groups called coding window (CW) that carry n packets and processes the encoding as well as decoding independently on each CW. Two basic parameters affecting the efficiency of TCP/NC are the redundancy factor (R) and the recovery capacity of one CW (k). $R=\frac{m}{n}$ is the ratio of combination to original packet. In other words, it shows the recovery ability of the system. R can be chosen based on the link loss rate. If the link loss rate is high, a large R can improve the goodput performance; whereas, for a low link loss rate, a large R incurs the unnecessary redundancy and reduces the goodput.

The recovery capacity of one CW ($k=m-n$) is also understood as the number of accepted losses of each CW which can be recovered without retransmission. It can be called briefly as "the CW recovery capacity". k is chosen based on the types of channels, the time taken by the sink to wait

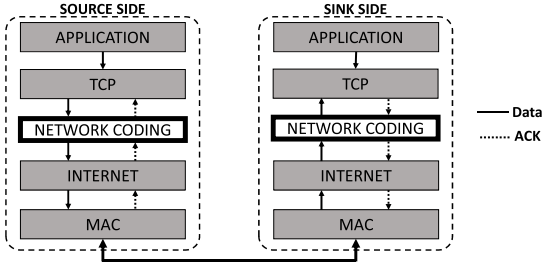

Fig. 1 The network coding Sub-layer in the TCP/IP model.

for decoding (decoding delay) and the hardware limitations. In the fast fading wireless channel, the occurrence probability of the burst loss and the situation are increased, in which the number of losses in each *CW* exceeding *k*. Therefore, a large *k* can improve throughput. However, a large *k* implies that *m* and *n* are large; thus, the buffer of the NC layer (NC buffer), the combination overhead, the processing complexity and the decoding delay become large [4]. Furthermore, if *k* is too large, the source cannot be aware of congestion even though the congestion origin burst packet losses occurs; thus, it cannot reduce its sending rate in a timely manner, which may cause more congestion and further performance degradation. Therefore, choosing *k* should be considered to ensure a balance among the burst loss recovery ability and other limitations.

Since the setting of a large *k* is not always the solution, an efficient retransmission mechanism for burst loss with the number of lost packets beyond *k* is of practical importance for TCP/NC. However, the retransmission mechanism of TCP/NC is not efficient because it entirely relies on TCP layer which can only retransmit the lost packets one at a time. The sink must wait for a long period to receive a sufficient number of the retransmitted packets, then recover all original packets and release the NC buffer.

Another shortcoming of TCP/NC is the dependent combinations which are generated by the random coefficients selection. Dependent combinations will more likely happen when a large *k* is used to increase the burst loss recover ability. Consequently, the sink cannot decode all packets even though it receives enough combinations.

In our work, to make TCP/NC more effective in practice, to implement two new retransmissions schemes, forward retransmission (FR) and dependence retransmission (DR). With FR scheme, the system can recognize exactly which packets are lost and retransmit multiple lost packets sequentially and efficiently rather than sending only one lost packet in one RTT as original TCP/NC. With the DR scheme, the system can determine and retransmit the suitable packets to solve the dependence problem. Thus, the *k* can be increased to a large value to increase the burst loss recovery ability. Besides the two retransmission schemes, we propose to encode all retransmitted packets to prevent the repeated packet losses which cause an increase of the number of TCP timeout situations. The combination of FR, DR and encoding the retransmitted packets schemes are called enhanced retransmission (ER). The definition of all protocols used in this paper are described in Table 1.

The ER is controlled by NC layer. To limit the adjustment of the system and for future evolution, we do not directly change the original TCP/NC system. We implement new modules that operate in parallel with the old system. Note that one of the functional objectives of the proposed scheme is similar to TCP selective acknowledgment (TCP-SACK). However, it is difficult for TCP layer to exactly determine packet losses on the sink side in a timely manner owing to the structure of TCP/NC.

Table 1 Definition of protocol terms.

Term	Definition
TCP/NC or original TCP/NC	TCP with network coding [1],[3]
TCP/NCwDR	TCP/NC with dependence retransmission
TCP/NCwFR	TCP/NCwDR with forward retransmission
TCP/NCwER	TCP/NC with enhanced retransmission (the combination of TCP/NCwFR and encoding the retransmitted packets)

The remainder of this paper is organized as follows. In Sect. 2, we describe the fundamental concept of NC in TCP. The proposed protocol is presented in Sect. 3. Simulations and results are described in Sect. 4 and the conclusions is discussed in Sect. 5.

2. TCP with Network Coding Overview

2.1 Integrating Network Coding to Protocol Stack

The study of TCP/NC in [1], [3] proposed the implementation of NC in the protocol stack with minor changes by adding the NC layer between the TCP and internet layer, as shown in Fig. 1. The NC layer executes the encoding/decoding process and sets the ACK number of ACK packets by using the degree of freedom concept and the seen/unseen definition [5]. This layer is designed to be completely transparent with higher and lower layers. Thus, if all losses can be recovered by NC layer, TCP is unaware of any loss events occurring and keeps increasing the congestion window (CWND) to improve the performance.

2.2 Recovering the Lost Packet by Network Coding

2.2.1 Encoding

Encoding is a combination of *n* TCP segments to produce *m* combinations with $m \geq n$. Based on the channel states, *m* and *n* are defined as constant numbers or variable numbers [6]. *n* is also referred as the size of one coding window. Each combination can contain one or many TCP segments, and each TCP segment is multiplied by coefficient α. Combination *C* is created by Eq. (1). The coefficients can be chosen depended on the purpose of use. TCP/NC system uses random linear network coding (RLNC [7]) algorithm; thus, the coefficients are selected randomly. On the other hand, whenever receiving a new TCP segment, TCP/NC uses a sliding method to create one combination. The maximum sliding size, i.e., the coding sliding window, equals the maximum number of TCP segments in one combination or equals $k+1$. For example, in Fig. 2, four packets ($n=4$) are encoded to six combinations ($m=6$) at the source. Hence the *CW* size is four and the coding sliding window is three.

$$C[i] = \sum_{j=1}^{n} \alpha_{ij} p_j \; ; \quad i = 1, 2, 3, ..., m \tag{1}$$

2.2.2 Decoding

At any time, when the sink receives sufficient combinations,

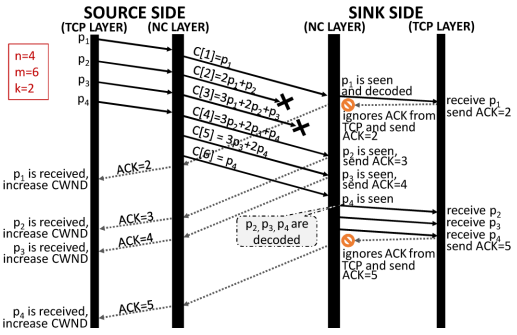

Fig. 2 Network coding process.

it will attempt to decode by using an inverse matrix function, e.g., the Gauss-Jordan elimination method. In case the received combination cannot be decoded to the original packets, the sink will store them in NC buffer and wait for the new combinations. In Fig. 2, $C[1]$ to $C[6]$ are sent through a lossy network, $C[2]$ and $C[3]$ are lost. Until $C[6]$ is received, the sink has four equations and four unknown variables. Using the Gauss-Jordan elimination method, four packets can be decoded even though two combinations are lost.

RLNC generates coefficients randomly; thus, it cannot guarantee that all combinations are linearly independent. The sink may not decode all the original packets despite of receiving sufficient combinations. With a large coefficient value (e.g., in a $\mathbb{GF}(2^8)$), the probability of creating the linear dependent combination is negligible [8]. However, in long TCP session, this problem still happens and makes the system interrupt (the result is shown in Sect. 4.1). Therefore, we propose the solution to overcome this problem described in Sect. 3.2.

2.3 TCP Functionality

2.3.1 Acknowledgement Mechanism

In TCP/NC system, in cases of a lost packet caused by the lossy channel, TCP layer should not aware of this event but should continue increasing the CWND. This process achieves a high stability and fast data transfer. Note that the number of lost packets must be less than the *CW* recovery capacity k. Otherwise, TCP needs to know the losses and retransmits them. TCP/NC uses the seen definition to redefine the ACK number in the ACK packet. TCP/NC does not send an ACK packet for a received packet or a decoded packet; it sends ACK packets for the degree of freedom [5].

Definition 1 (seeing a packet). *A node is said to have seen a packet p if it has enough information to compute a linear combination of the form (p+q), where q is itself a linear combination involving only packets that arrived after p at the sender [5].*

In Fig. 2, when combination $C[i]$ is received, p_i is always expressed to the set of p_k with $k>i$; thus, all packets are seen.

When C[1] is received: $p_1 = p_1 + 0p_2$
When C[4] is received: $p_2 = \frac{1}{3}(C[4] - 2p_3 - p_4)$

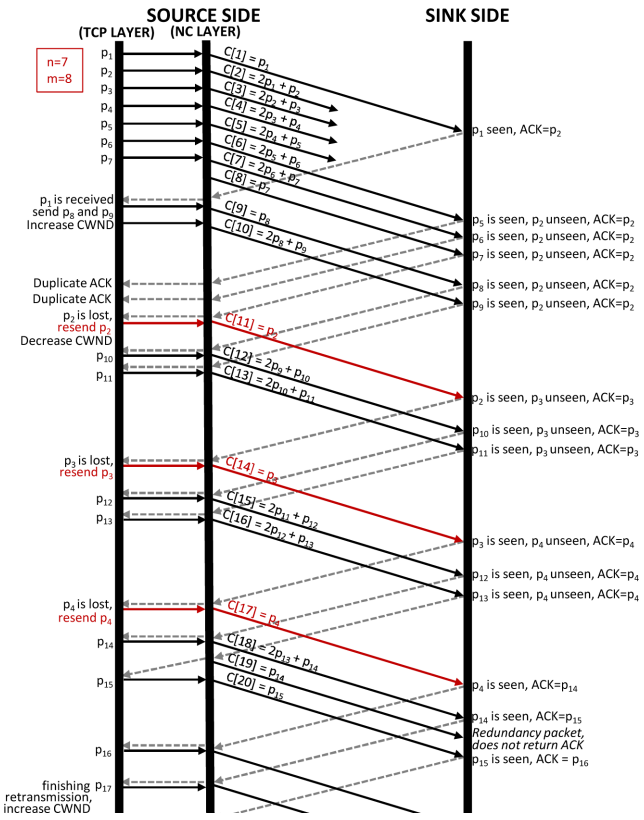

Fig. 3 Retransmission mechanism in original TCP/NC system.

When C[5] is received: $p_3 = \frac{1}{3}(C[5] - 2p_4)$
When C[6] is received: $p_4 = C[6] - 0p_5$

For every new seen packet, the sink returns an ACK packet whose ACK number corresponds to the smallest unseen packet even if all seen packets have not been decoded. The source simply forwards this ACK number to TCP layer; therefore, the TCP layer is unaware that a loss has occurred and it continues increasing the CWND. In Fig. 2, the k is equal to 2; thus, the system can work normally with two lost combinations. As a result, although $C[2]$ and $C[3]$ are lost, all packets are seen sequentially; thus, the CWND continues to increase. When the last combination $C[6]$ is received, all original packets are decoded.

2.3.2 Retransmission and Congestion Control Mechanism

TCP/NC just attempts to use the original retransmission and congestion control mechanisms of TCP layer. In other words, NC layer is completely transparent in these mechanisms.

When the number of lost packets exceeds the *CW* recovery capacity, the sink will return ACK packets with the same value of the smallest sequence number of an unseen packet. At the source, NC layer simply forwards this information to TCP layer which makes retransmission decisions based on triple duplicate ACKs or timeout events. Figure 3 shows the standard retransmission mechanism in four lost combinations. This example assumes that k is 1; thus,

three packets must be retransmitted to decode all the original packets. TCP/NC needs to retransmit the three unseen packets p_2, p_3 and p_4. The retransmission process finishes after the source receives the ACK packet used to confirm p_{14}. Increasing and decreasing CWND for controlling congestion are processed by TCP layer. In this example, TCP/NC needs three times to retransmit the combinations. It takes a long time to start increasing CWND. To shorten the waiting time, we propose the new retransmission technique described in Sect. 3.1.

3. Enhanced Retransmission Scheme

The simplest idea of the enhanced retransmission has been presented in [9]. It is brought back in this paper with new improvements, which make it become more realistic. The enhanced retransmission includes three functions, forward retransmission, the encoding of retransmitted packets and dependence retransmission.

3.1 Forward Retransmission Scheme

In this section, we discuss the forward retransmission scheme, which can quickly adapt to the burst losses. If the k is larger than 1, TCP/NC can recover the burst losses. However, as mentioned in Sect. 1, k must be limited. If the size of the burst losses is large, TCP layer should retransmit the oldest unseen packets and decrease the CWND to avoid congestion. However, the original TCP/NC does not have an independent retransmission mechanism; it depends entirely on TCP layer. The retransmitted packets are sent at different RTTs. Therefore, TCP/NC takes a long time for retransmission. Return to the example in Fig. 3, the original TCP/NC takes three RTTs to retransmit all the unseen packets p_2, p_3 and p_4.

TCP must retransmit all lost packets; however, TCP/NC needs to retransmit only the unseen packets. Technically, the system has to know how many lost combinations and how many unseen packets occur. Consequently, the system can indicate how many packets must be retransmitted and sequentially retransmit them in one RTT after receiving the first transmitted packet from TCP layer. This method helps to shorten the retransmission time to reduce the entire flow transfer time. In Fig. 4, the proposed system sequentially sends p_2, p_3 and p_4. After the NC layer receives the retransmitted p_2 from the TCP layer, it determines other unseen packets (p_3, p_4) to retransmit following p_2. The retransmission process finishes after the source receives the ACK packet used to confirm p_{10}. To achieve this goal, we must implement three additional functions, i.e., determining the number of packet losses, enabling the retransmission process, and retransmitting lost packets in NC layer that are explained in the next subsections.

3.1.1 Determining the Number of Lost Packets

First, NC layer must know the total lost packets to determine which lost packets should be retransmitted. We propose to use some additional information besides the normal TCP header; thus, the new header called NC-ACK header is added. We add the new Packet-id (*Pid*) field to the NC header as well as the new Packet-id echo-reply field (*Pid-reply*) and redundancy-indicate flag (*R-flag*) in the NC-ACK header, as shown in Fig. 5 and Table 2. Each combination has a unique *Pid* number. With each received combination, the sink will return the ACK packet which has the *Pid-reply* number equal to the *Pid* number in this combination. After receiving the ACK packet, the source can know exactly which combination has been lost. With this loss counting

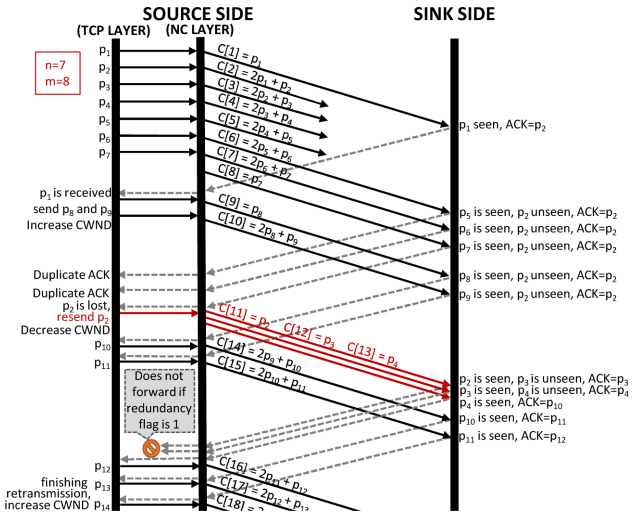

Fig. 4 Retransmission mechanism in TCP/NCwFR system.

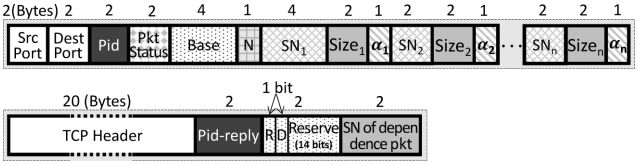

Fig. 5 The NC header (above) and the NC-ACK header.

Table 2 The description of the NC header and NC-ACK header fields.

Field name	Description
SrcPort	The source port number
DestPort	The destination port number
Pid	The packet identity
Pkt status	The packet status
Base	The sequence number (SN) of the oldest packet in the NC buffer of the source. Using for buffer management in the sink.
N	The number of original packets in the combination
SN_1	The SN of the first original packet
SN_n	Equals $SN_n - SN_1$
$Size_n$	The payload size of n-*th* packet
α_n	The n-*th* NC coefficient
Pid-reply	The packet identity echo reply
R	The redundancy indicate flag
D	The dependence indicate flag
SN of dependence pkt	The SN of the dependence packet

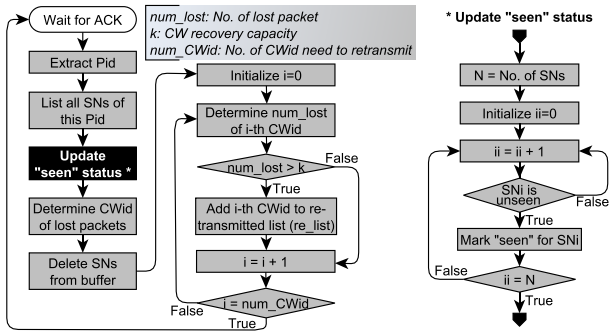

Fig. 6 The receiving ACK function at the source.

purpose, the sink has to return an ACK packet for all combinations including the redundant combinations which should be ignored by original TCP/NC. Therefore, the sink must inform the source of which ACK packet is used only for counting (rather than confirming) and does not forward to the TCP layer. The sink will set the *R-flag* to 1 in the returned ACK packet of the redundant combination.

To determine the starting time of the retransmission (in Sect. 3.1.2) and which packets have to be retransmitted (Sect. 3.1.3), NC layer stores the information of each *CW* including the number of the lost packets, the *CW* recovery capacity *k*, the *Pid*s and the sequence number of the original packets of each *Pid*. *CW*s are distinguished by the ordinal number of them (*CWid*).

3.1.2 Enabling Retransmission Process

NC layer should retransmit the unseen packets; thus, it must know the seen/unseen status of all transmitted packets. After receiving the ACK packet, based on the stored sequence numbers of original packets in each transmitted combination, the source can infer which packets had been seen or unseen by the sink. In the left flowchart in Fig. 6, when NC layer at the source receives an ACK packet, it will read the *Pid-reply* and list all the sequence numbers combined in the combination which has *Pid* equal to *Pid-reply*. Subsequently, NC layer can update the seen/unseen status of all packets, as shown in the right flowchart in Fig. 6. Besides, NC layer counts the number of lost packets and stores their *CWids*. If the number of lost packets is greater than the *CW* recovery capacity, the sink cannot decode all combinations. NC layer adds this *CWid* to a list in NC buffer control block called the retransmitted list (*re_list*) which stores all *CWs* containing the retransmitted packets. Disabling or enabling of the retransmission process will then be relied on the empty or not empty status of *re_list*. This list will be cleaned when NC layer finishes retransmission process.

3.1.3 Retransmitting the Lost Packets

As mentioned previously, we use the advantages of the congestion control of TCP. In our propose, the forward retransmission scheme only processes after TCP layer decreases CWND and retransmits a packet. The forward retransmission process is shown in Fig. 7. Basically, NC layer determines the *CWid* of the retransmitted packet (step 1), calculates the number of the retransmitted packets in each *CWid* (step 2) and forwards them to the internet layer after adding NC header (step 4).

As the main purpose of forward retransmission is to retransmit multiple lost packets and to force the TCP layer to assume that only one packet has been lost, NC layer forwards only one ACK packet that confirms the last unseen packet in all retransmitted unseen packets to TCP layer. In other words, NC layer will ignore all ACK packets until receiving the ACK packet of the last retransmitted packet as shown in Fig. 4. To know which ACK packet is needed to forward to TCP layer, we manipulate the *R-flag* of NC-ACK header that is used for indicating the ACK packet of redundant combination. The sink will set this flag to 1 if the received combination is the retransmitted combination, but this combination is not the last one. The key issue is how the sink determines which combination is the retransmitted one and this is the last one. Thus, we propose to add a new information to NC header called a packet status field. When NC layer creates the combination for the retransmitted packets, the packet status of the last retransmitted combination is set to 2 in NC layer. For other retransmitted combinations, the packet status is set to 1. If this is a normal combination, the field is 0. Setting 1 or 0 for *R-flag* in NC-ACK header will depend on the packet status field of 1 or 2 respectively.

There are three situations in which NC layer receives a retransmitted packet from the TCP layer. First, the source has already retransmitted the lost packets but the last one is lost. This situation is called "lost the last" in Fig. 7. The source just receives ACK packets whose *R-flag* is 1; hence TCP layer has never received an ACK packet. In this situation, TCP layer retransmits the first combination of the group of retransmitted combinations even if this packet has been seen at the sink. When this packet arrives at NC layer, NC layer creates and returns an ACK packet which has ACK number equal to the sequence number of the unseen packet. After that, TCP layer retransmits the actual lost packet.

The second situation is retransmission timeout. When the retransmission time is exceeded, TCP layer will retransmit a packet. NC layer checks whether the packet receiving from TCP layer is in the retransmitted packet list (*re_p_list*) or not, which is the list of retransmitted packets. If not, NC layer simply adds it in the *re_p_list* shown in Fig. 7 at step 3.

Finally, in a normal situation, NC layer will determine the number of *CWids* (*num_CWid*) that has the packets needed to retransmit (in Fig. 7 at step 1), and the number of the retransmitted packets (at step 2). Moreover, the NC layer needs to list all the retransmitted packets into *re_p_list* for each *CWids*. After that, NC layer simply retransmits the packets in *re_p_list* of each *CWids* sequentially (at step 4).

3.1.4 Encoding Retransmitted Packet Scheme

One of the reasons that causes a decrease in performance

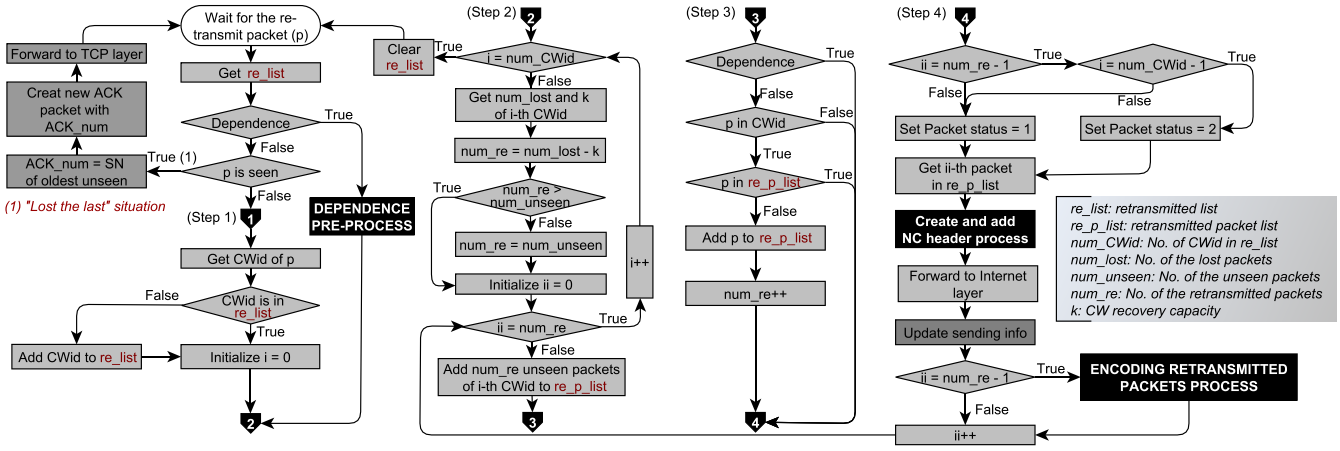

Fig. 7 The forward retransmission function at the source.

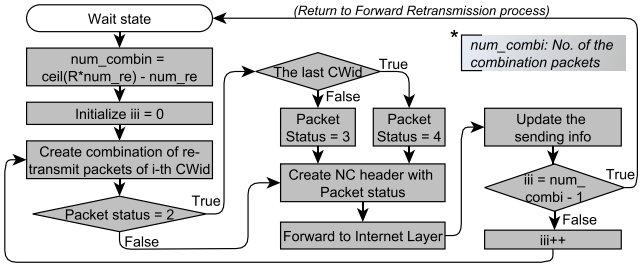

Fig. 8 Encoding the retransmitted packets function at the source.

Table 3 Packet status description.

Packet status	Description
0	The normal combination
1	The coded/uncoded retransmitted combination which is not in the last *CW*
2	The last uncoded retransmitted combination of the last *CW*
3	The coded retransmitted combination of the last *CW*
4	The last coded retransmitted combination of the last *CW*

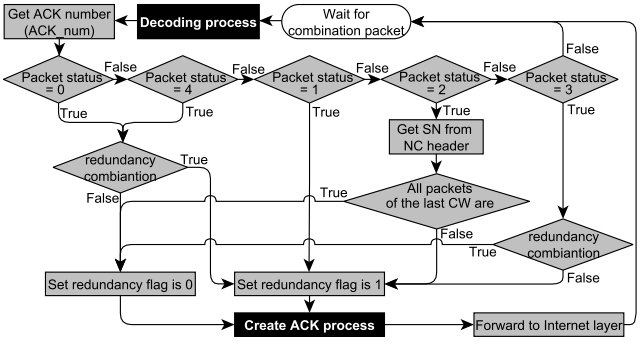

Fig. 9 Packet status processing at the sink.

is the repeated loss which leads to an increase of the number of timeout situations. A larger value of link loss rate is, the more repeated losses occur. In the forward retransmission scheme, if the last retransmitted combination is lost, the timeout situation will be happened. To limit the repeated loss, we propose the encoding of the retransmitted packets.

The encoding process of the retransmitted packets is shown in Fig. 8. As our system handles packets in separate *CW*, the encoding process is executed for each *CW*. In this section, *CW* is referred to only those that contain the retransmitted packets. In each *CW*, all the retransmit packets are sent one by one as the normal combinations which have only one original packet. Then, all the retransmitted packets will be encoded into $\lceil R \times num\_re \rceil - num\_re$ combinations where *num_re* is the number of retransmitted packets. To avoid confusion, the combination which is not created from the encoding process is called an uncoded retransmitted combination. The other is called a coded retransmitted combination. Since there are some new types of the retransmitted combinations, we supplement two states for packet status variables increasing the total number of packet status to five states which are described in Table 3.

Depend on the operation of the forward retransmission, the source forwards only one ACK packet to the TCP layer based on the *R-flag* in NC-ACK header. At the sink, instead of assigning the last ACK packet as a forwarding ACK packet, the new scheme has more choices which are determined based on the received packet status as shown in Fig. 9.

If the packet status is 0, the received combination is not the retransmitted combination. The sink simply checks whether it is the redundancy combination or not and sets the *R-flag* to 1 or 0, respectively.

If the packet status is 1, the received combination is the retransmitted combination which has not created from the retransmitted packets of the last *CW*. Because only one ACK packet can be forwarded to TCP layer, the sink should wait for the remaining retransmitted combinations. The *R-flag* is always set to 1.

In the case of packet status is 2, the received combination is the last uncoded retransmitted combination of the last *CW*. The system will check whether all the retransmitted packets of this *CW* are decoded or not. If not, some uncoded retransmitted combinations are lost; thus, NC layer waits for the coded retransmitted combinations and set *R-flag* to be 1. Otherwise, the *R-flag* is set 0.

If packet status is 3, the received combination is the

coded retransmitted combination of the last *CW* which is used in case the uncoded retransmitted packets are lost. This combination is combined from all the retransmitted packets of the last *CW*. Thus, if it can be used to decode to new packets, some uncoded retransmitted combinations, which was lost, are recovered. Hence NC layer does not need to wait for the other retransmitted combinations. The *R-flag* is set to be 0. Conversely, it is set to 1.

Finally, the packet status is 4. The received combination is the last coded retransmitted combination. If this combination can be used for decoding, it is not the redundant combination. It means no ACK packet is forwarded to TCP layer. Thus, the *R-flag* must be set to 0. Otherwise, it is the redundant combination. The *R-flag* is set to 1.

In all cases the ACK number will be set to the smallest sequence number of the lost combinations.

3.2 Dependence Retransmission Scheme

This part, we describe how to retransmit the suitable packets to solve problem of dependent combinations which cause the failure of the decoding process. The type of retransmission and the retransmitted packet are called "dependence retransmission" and "dependence packet", respectively.

As discussed above, to work well with a heavy and bursty loss, a value of k should be higher than 1. In RLNC method, the k that is greater than 2 can cause the dependence combinations. The system cannot decode to original packets even though all packets are "seen" at the sink. Thus, the system must determine and retransmit the suitable packets to solve this problem. However, it has two challenges. First, the packet that is the origin of the dependence situation, might have been seen at the sink; hence the returned ACK number will never equal to the sequence number of this packet. The second is how to make TCP layer decrease the CWND to avoid buffer overloading. The normal retransmission method which uses only the ACK number to determine the retransmitted packet cannot be operated. Therefore, we propose the dependence retransmission method which adds to the NC-ACK header a one bit Dependent indicates flag (*D-flag*) and 2 bytes SN as a dependence packet field (*Dependence-SN*). *D-flag* is used to inform the source about the status of the dependence situation. If *D-flag* is set, it means the dependence situation is happening; thus, the source retransmits the packet having the sequence number equal to *Dependence-SN*. Otherwise, the source operates normally.

3.2.1 Determining the Suitable Packet to be Retransmitted

To specify a dependence packet, the sink uses an inverse matrix method, e.g., the Gauss-Jordan elimination, to simplify the coefficient matrix which is generated from the received combinations. If the dependence situation happens, this has some all-zero rows corresponding with the packets which cannot be decoded and needed to be retransmitted. After determining the dependence packets, sink sets *D-flag* to 1 and

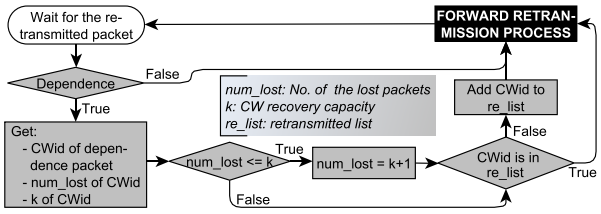

Fig. 10 Preprocessing for retransmitted dependence packet at the source.

Dependence-SN to the smallest sequence number of dependence packets in all returned ACK packets until the dependence situation is solved. Moreover, to avoid the NC buffer overloading caused by the inability of releasing the buffer, the sink has to assign the ACK number to the last ACK number even though the packet, which has the sequence number equal to this ACK number, might have been seen. Source always stores *D-flag* and *Dependence-SN* whenever receiving the ACK packet. When the triple duplicate ACK condition is matched, TCP layer retransmits a packet indicated from the received ACK number. If this packet arrives at NC layer and the *D-flag* is 1, a suitable packet which has the sequence number equal to *Dependence-SN* will be sent. In case of the retransmitted packet is seen, it will be ignored. By waiting for the retransmitted packet from TCP layer to retransmit the dependence packet, our proposed can take advantage of TCP congestion control to prevent the buffer overloading.

3.2.2 Retransmitting the Suitable Packet

Basically, the dependence retransmission scheme uses the retransmission mechanism of the forward retransmission scheme to retransmit the dependence packet, which was discussed in the Sect. 3.1. Therefore, the main process of the dependence retransmission is to force the forward retransmission to retransmit the dependence packet even though no loss packets needed to be retransmitted, as shown in Fig. 10. When NC layer receives the retransmitted packet from TCP layer, it specifies the dependence situation status through the stored *D-flag* from the last ACK packet. If the flag is 0, this packet is a normal retransmission; hence NC layer just simply forwards this packet to the forward retransmission process. Otherwise, this packet is the dependence packet. That means the dependence problem is happening. In the case of the *CWid* of this packet is not in the retransmitted list (*re_list*) because there is no retransmitted packet, NC layer adds this *CWid* to *re_list* and sets the number of lost packet of this *CWid* to k+1. Finally, the process will turn to the forward retransmission process to retransmit the dependence packet and the lost packets if there is any.

4. Simulation Result

The combination of the forward retransmission, encoding the retransmitted packet and the dependence retransmission is called the enhanced retransmission (TCP/NCwER). With k>2, the original TCP/NC cannot work because of the linear

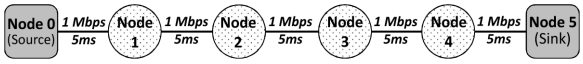

Fig. 11 Simulation topology 1: single flow.

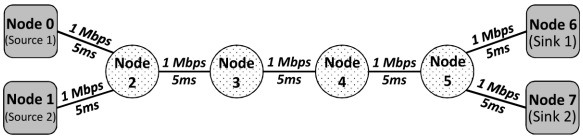

Fig. 12 Simulation topology 2: multiple flows.

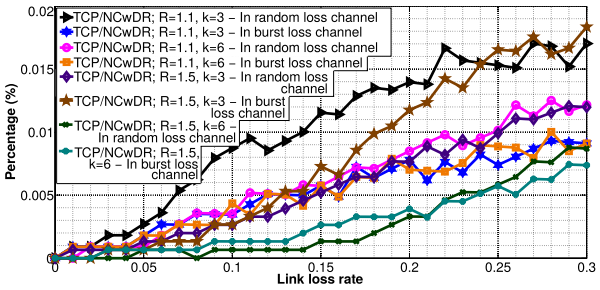

Fig. 13 The dependence situations vs the link loss rate for TCP/NCwDR.

dependent combinations. Thus, TCP/NCwDR is used to replace it. Their performance are equal in the case that no dependence problem occurs. To compare the goodput between without encoding and encoding the retransmitted packets, we use TCP/NCwFR and TCP/NCwER. The dependence retransmission scheme is integrated in both of them.

The implementation of TCP/NCwER was accomplished using Network Simulator 3 (ns-3) [11]. The topology of the simulation was a tandem network consisting of four routers. In a first scenario, one source and one sink were connected to four routers, as shown in Fig. 11. In a second scenario, two sources and two sinks were employed, as shown in Fig. 12. The source and the sink were at opposite ends of the chains. All links had a bandwidth of 1 Mbps and a propagation delay of 5 ms. The TCP type used in the simulation was NewReno, and the packet size was 1000 bytes. The transferred data size was 100 Mbytes. In all simulations, we performed the system at least 10 times to obtain the average value.

4.1 Benefit of Using Dependence Retransmission

The first scenario (Fig. 11) is used in this simulation. In this scenario, the link loss rate was set from 0 to 0.3. To evaluate the goodput performance of the variants of TCP/NC over such a wide range of link loss rate, we choose two values of redundancy factor R, one is small (R=1.1) and the other is large (R=1.5) unless otherwise noted. With the fixed R, n can be calculated corresponding to k based on the equation $n=\frac{k}{R-1}$.

Figure 13 shows the evolution of dependence situations versus the lost rate. The Y-axis represents the percentage of the number of dependence situation over the number of sending combination packets. Although the result is the average value of 20 iterations, it is not convergence. The reason is that the number of the dependence situations is very huge. Therefore, the simulation of the completed dependence situation seems impossible. However, the result can show the increasing trend of the dependence situation. It is greater than 0 if the loss rate is increased. With the dependence retransmission, our simulation can run to transfer completely all 100 Mbytes data with link loss rate is 0.3.

4.2 Performance of Single Flow

The first scenario which has the topology shown in Fig. 11 continues to be used to evaluate the fundamental effectiveness of our proposed. The simulation was run in 2 types of channel, random loss and burst loss. TCP NewReno and TCP Westwood+ were used to compare to our proposed.

4.2.1 Random Loss Channel

The random loss channel simulates a slow fading channel which causes the separated losses. At first, Fig. 14 and Fig. 15 show all TCP/NC variants which have the recovery ability significantly outperform TCP NewReno and TCP Westwood+. Besides, Fig. 14 also indicates the efficiency of increasing k. When k is increased from 1 to 6, the goodput will be improved. Because the losses are dispersed, the performance of system is not much improved when k is larger than 2. The results indicate that a higher k is better.

In Fig. 15, we compare the goodput of three protocols, TCP/NCwDR, TCP/NCwFR and TCP/NCwER. As mentioned in Sect. 1, k should be limited; thus we set to 3 in this simulation. The goodput of TCP/NCwDR and TCP/NCwFR are slightly different because the losses are separated. However, when the link loss rate is increased, separated losses exceed the CW recovery capacity. Related to the retransmission capacity for multiple packets at once, TCP/NCwFR performs better than TCP/NCwDR. While, if TCP/NCwER is used, repeated losses is limited. The number of timeout situations is degraded which is shown in Fig. 16. Therefore, the goodput is the best.

4.2.2 Burst Loss Channel

The burst loss channel simulates a fast fading channel e.g., mobile wireless channels. We use two types of the burst loss model, one is a default burst loss model of ns-3 simulator (NS3 burst loss model) and the other is a well-known Gilbert burst loss model [12]. In NS3 burst loss model, under a given link loss rate, the number of the continuous packet losses in one loss event is chosen randomly from 1 to the changeable number l called "the max burst loss size". In this simulation, l is set at 3, 5 and 7. In the Gilbert burst

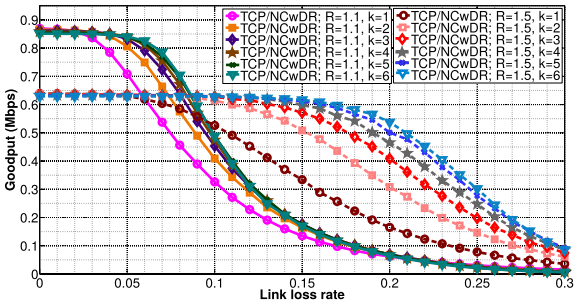

Fig. 14 The effect of k to the goodput performance in the random loss channel.

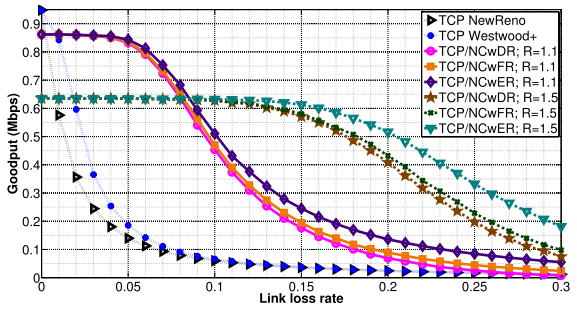

Fig. 15 Goodput comparison among protocols in random loss channel.

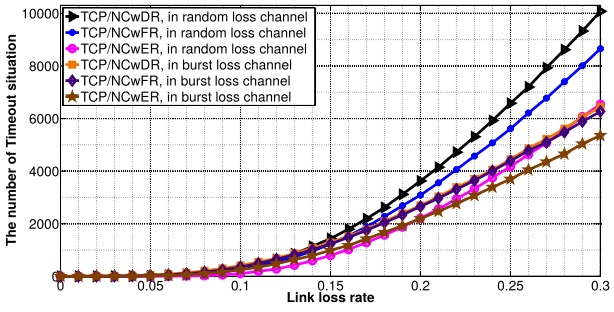

Fig. 16 The timeout situation vs the link lost rate (R=1.1, k=3).

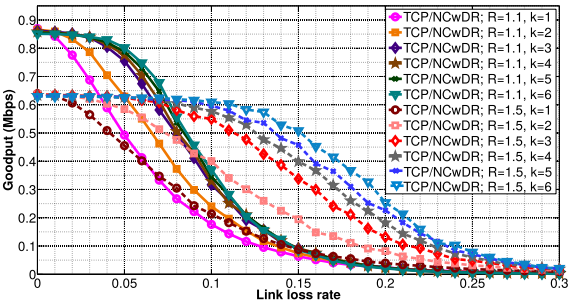

Fig. 17 The effect of k to the goodput performance in NS3 burst loss channel with l=3.

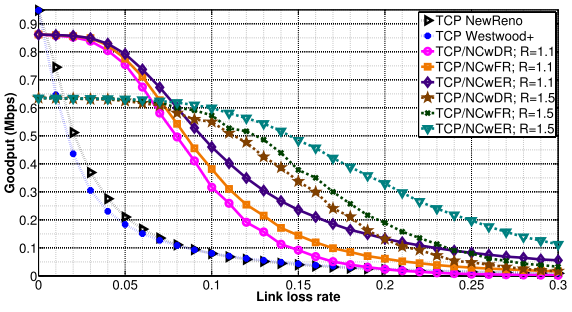

Fig. 18 Goodput comparison among protocols in NS3 burst loss channel with l=3.

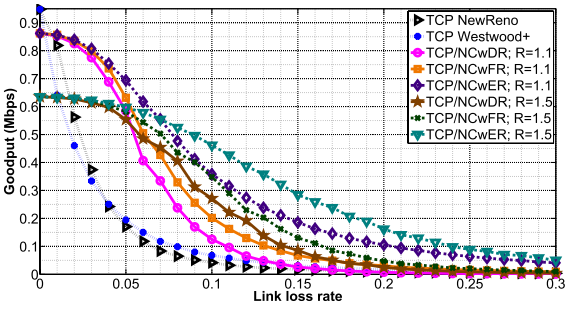

Fig. 19 Goodput comparison among protocols in NS3 burst loss channel with l=5.

loss model, we cannot control the max burst loss size in contrast to NS3 burst loss model. The number of the continuous packet losses in one loss event is totally random; thus, we can only set the average burst loss size (L) under a given link loss rate. To compare two types of burst loss model with a "similar" setting, we calculate L of Gilbert burst loss model based on l of NS3 burst loss model under the same link loss rate. In the first case, we choose L so that the average burst loss size of Gilbert burst loss model is equal to that of NS3 burst loss model with l, i.e., L=(l+1)/2. In the second case, we choose L so that the 90-percentile of burst loss size of Gilbert burst loss model is equal to that of NS3 burst loss model with l, i.e., L=1/(1−0.1$^{1/(0.9\times l)}$).

NS3 burst loss channel: In this simulation, the max burst loss size (l) is set at 3. The results are shown in Fig. 17 and Fig. 18. In Fig. 17, we can see the efficiency of increasing k. Since the number of continuous losses is up to 3, the goodput is significantly improved when the k is set to 3 or greater compared to 2 or less. While the performance of TCP/NCwDR and TCP/NCwFR are nearly the same in the random loss channel in Fig. 15, their performance are clearly different in the burst loss channel in Fig. 18. When encoding the retransmitted packets is applied, the goodput is improved significantly. We increase l to 5 or 7 in Fig. 19 and Fig. 20. The results remain constant, i.e., TCP/NCwER always performs better than TCP/NCwDR, TCP NewReno or TCP Westwood+.

Gilbert burst loss channel: l=5 is used to calculate L in Gilbert burst loss channel. In case 1 mentioned above, L is equal to 3. In case 2, L is equal to 2.4968. Fig. 21, and Fig. 22 show the goodput comparison of TCP/NCwDR and TCP/NCwER in three cases of the burst loss channel, which are NS3 burst loss channel (NS3), Gilbert burst loss channel with L=3 (Gilbert1) and Gilbert burst loss channel with L=2.4968 (Gilbert2), by applying two different redundancy

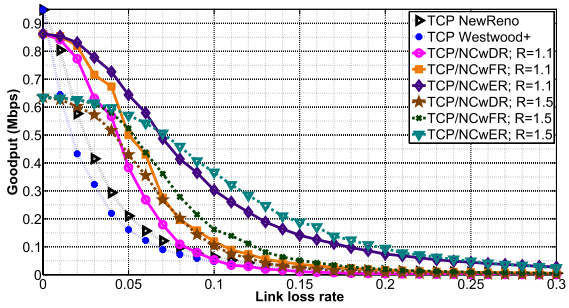

Fig. 20 Goodput comparison among protocols in NS3 burst loss channel with l=7.

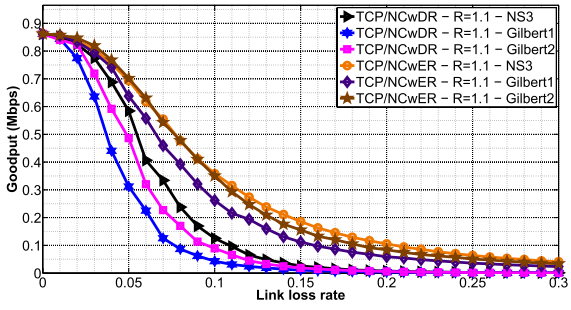

Fig. 21 Goodput comparison between TCP/NCwDR and TCP/NCwER (R=1.1, k=3) in the burst loss channel.

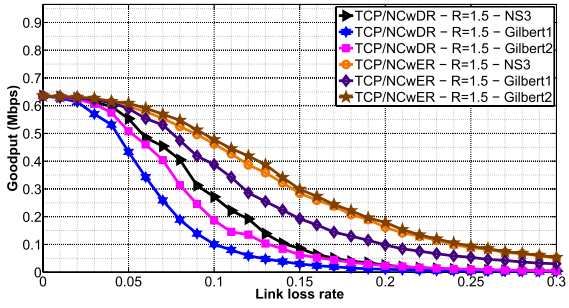

Fig. 22 Goodput comparison between TCP/NCwDR and TCP/NCwER (R=1.5, k=3) in the burst loss channel.

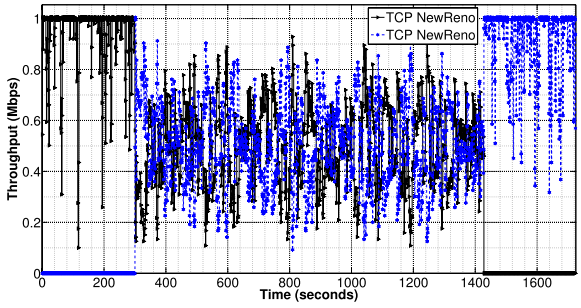

Fig. 23 Network congestion - two TCP NewReno flows in lossy network.

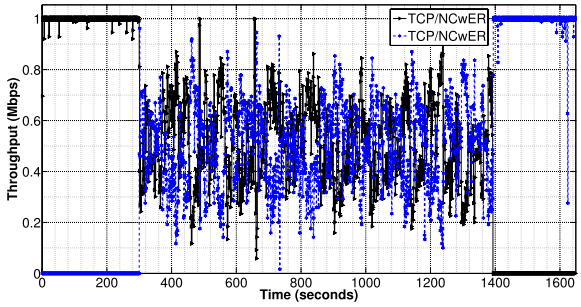

Fig. 24 Network congestion - two TCP/NCwER flows in lossy network.

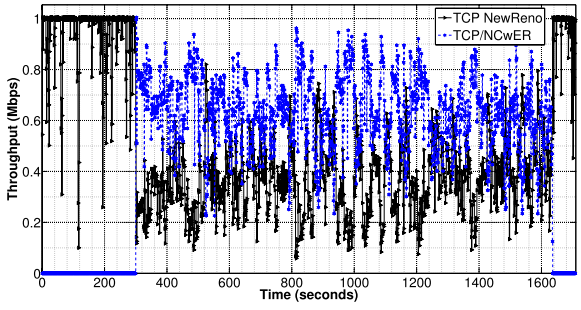

Fig. 25 Network congestion - one TCP and one TCP/NCwER flow in lossy network.

factor R, respectively. Although the conditions of channels are different, the tendency of the goodput performance is almost the same. The goodput of TCP/NCwER is always better than TCP/NCwDR.

4.3 Performance of Multiple Flows

The topology shown in Fig. 12 is used to evaluate the proposed scheme with multiple flows. To show a compatibility of TCP/NCwER to TCP NewReno, a moderate link loss rate is chosen to be 0.01. To recovery the losses, R is set at 1.01 and k is set at 1. The buffer size of the links was set at 100 packets. Source 1 began at t=0s and source 2 began at t=300s. Every flow stops transferring data if it finishes sending 100 Mbytes data. The current throughput was calculated at intervals of 1s. We examined three cases. In case 1 and 2, two flows use the same protocol, TCP NewReno or TCP/NCwER. In case 3, the TCP/NCwER flow competes with the TCP NewReno flow.

Figure 23 and Fig. 24 show that the fairness is satisfied. The time-averaged throughput over congestion period, which is calculated from 300s until one flow finishes to transfer, is nearly the same in two competitive flows. In Fig. 23, the time-averaged throughput is 0.4983 Mbps and 0.5061 Mbps, respectively. And the time-averaged throughput is 0.4968 Mbps and 0.5112 Mbps for two TCP/NCwER flows in Fig. 24.

In Fig. 25, because of the lossy channel and the network congestion, TCP flow decreases the sending rate for every loss, on the contrary, TCP/NCwER flow can recover the losses in many cases. Therefore, the time-averaged throughput of TCP flow and TCP/NCwER over congestion period, are 0.3688 Mbps and 0.6366 Mbps. The fairness is not fully satisfied but it can be acceptable. When this sim-

ulation ran again with the link loss rate set to 0, the time-averaged throughput were 0.4346 Mbps and 0.5702 Mbps. Base on the results of both loss and no loss channel cases, the compatibility can be accepted.

5. Conclusion

We proposed a method that avoids the two key deficiencies of the original TCP/NC, the inefficiency retransmission and the limitation of k. These shortages lead to the low performance of the system in the heavy and bursty losses. Some studies focus on methods which can dynamic change R and k such as Self-Adaptive NC with TCP (SANC-TCP) [4] and Adaptive NC with TCP (ANC-TCP) [6] for the variance channels. However, they do not respond in time when the channel changes rapidly. With these unstable channels, our proposed has advantages in determining the lost packets and scheduling sequential retransmission in order to shorten the retransmission time. It also can encode all the retransmitted packets to prevent the repeated loss which causes the increase of timeout situation. Moreover, our proposed method can solve the dependence situation that causes the failure decoding at the sink by determining and retransmitting the suitable packets. We implemented the proposed TCP/NCwER using ns-3. The simulation results demonstrate that the proposed scheme can improve the goodput of a system compared to the original TCP/NC, TCP Westwood+ and TCP NewReno.

In this study, we adopted constant R and k. However, in practice, the link loss rate varies in time, depending on the channel state. In future, we will attempt to integrate our proposed into adaptive TCP/NC, such as SANC-TCP and ANC-TCP. Another enhancement direction can be considered are applying the congestion detection [13] into the system to respond well with network congestion.

This work is partly supported by JSPS KAKENHI (16K00130) and KDDI Foundation.

References

[1] J.K. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros, "Network coding meets TCP," Proc. 28th Conf. on Comput. Commun., Rio de Janeiro, Brazil, no.2, pp.280–288, April 2009. DOI: 10.1109/INFCOM.2009.5061931

[2] C.C. Chen, C. Chen, S.Y. Oh, J.S. Park, M. Gerla, and M.Y. Sanadidi, "ComboCoding: Combined intra-/inter-flow network coding for TCP over disruptive MANETs," Int. J. Advanced Research, vol.2, no.2, pp.241–252, May 2011. DOI: 10.1016/j.jare.2011.05.002

[3] J.K. Sundararajan, D. Shah, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets TCP: Theory and implementation," Proc. IEEE, vol.99, no.3, pp.490–512, Feb. 2011. DOI: 10.1109/JPROC.2010.2093850

[4] C.Y. Cheng and H.Y. Yi, "Adaptive network coding scheme for TCP over wireless sensor networks," J. Comput. Commun. and Control, vol.8, no.6, pp.800–811, Dec. 2013. DOI: 10.15837/ijccc.2013.6.26

[5] J.K. Sundararajan, D. Shah, and M. Medard, "ARQ for network coding," Proc. IEEE Int. Sym. on Info. Theory, Toronto, Canada, pp.1651–1655, July 2008. DOI: 10.1109/ISIT.2008.4595268

[6] S. Song, H. Li, K. Pan, J. Liu and S Y R Li, "Self-adaptive TCP protocol combined with network coding scheme," Proc. 6th Conf. on Sys. and Net. Commun., pp.20–25, Barcelona, Spain, Oct. 2011.

[7] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," Proc. IEEE Int. Sym. on Info. Theory, Yokohama, Japan, June 2003. DOI: 10.1109/ISIT.2003.1228459

[8] C. Fragouli, J.L. Boudec. Shah, and J. Widmer, "Network coding: an instant primer," ACM/SIGCOMM Comput. Commun. Review, vol.36, no.1, pp.63–68, Jan. 2006. DOI: 10.1145/1111322.1111337

[9] N.V. Ha, K. Kumazoe, and M. Tsuru, "TCP network coding with forward retransmission," Proc. IEEE of Asia Pacific Conf. on Wireless and Mobile, Bandung, Indonesia, pp.136–141, Aug. 2015. DOI: 10.1109/APWiMob.2015.7374970

[10] M.J. Kim, M. Medard, and J. Barros, "Modeling network coded TCP throughput: A simple model and its validation," Proc. 5th Int. ICST Conf. on Perfor.e Eval. Methodologies and Tools, Brussels, Belgium, pp.131–140, May 2011. DOI: 10.4108/icst.valuetools.2011.246509

[11] "Network simulator (ns-3)," https://www.nsnam.org/, accessed March 1. 2016.

[12] C. Jiao, L. Schwiebert, and B. Xu, "On modeling the packet error statistics in bursty channels," Proc. 27th Local Computer Networks, Florida, USA, pp.534–541, Nov. 2002. DOI: 10.1109/LCN.2002.118 1827

[13] S. Cen, P.C. Cosman, and G.M. Voelker, "End-to-end differentiation of congestion and wireless losses," J. IEEE/ACM Trans. Netw., vol.11, no.5, pp.703–717, Oct. 2003. DOI: 10.1109/TNET.2003.818187

Nguyen Viet Ha received the B.S. degree (2009), and M.S. degree (2012) in Electronics and Telecommunications, both from Ho Chi Minh University of Science, Viet Nam. He is currently a Ph.D student in Kyushu Institute of Technology, Japan. His research interest are transport layer protocols and network coding. He is a student member of IEEE and IEICE.

Kazumi Kumazoe received the B.E. degree (1993), M.E. degree (1995) and D.E. degree (2007) in computer science from Kyushu Institute of Technology, Japan. Since April 2013, she has been a visiting researcher at Network Design Research Center, Kyushu Institute of Technology. Her research interests are in various areas of computer networks, including transport layer protocols, network coding and network management. She is a member of IEICE.

Masato Tsuru received the B.E. and M.E. degrees from Kyoto University, Japan in 1983 and 1985, and then received his D.E. degree from Kyushu Institute of Technology, Japan in 2002. He has been a professor in the department of Computer Science and Electronics, Kyushu Institute of Technology since 2006. His research interests include performance measurement, modeling, and management of computer communication networks. He is a member of the ACM, IEEE, IEICE, IPSJ, and JSSST.