TCP VON: Joint Congestion Control and Online Network Coding for Wireless Networks

Wei Bao, Vahid Shah-Mansouri, Vincent W.S. Wong, and Victor C.M. Leung
Department of Electrical and Computer Engineering
The University of British Columbia, Vancouver, Canada
e-mail: {weib, vahids, vincentw, vleung}@ece.ubc.ca

***Abstract*—In this paper, we propose TCP Vegas with online network coding (TCP VON), which incorporates online network coding into TCP. It is shown that the use of online network coding in transport layer can improve the throughput and reliability of the end-to-end communication. Compared to generation based network coding, in online network coding, packets can be decoded consecutively instead of generation by generation. Thus, online network coding incurs a low decoding delay. In TCP VON, the sender transmits redundant coded packets when it detects packet losses from acknowledgement. Otherwise, it transmits innovative coded packets. We establish a Markov chain to analytically model the average decoding delay of TCP VON. We also conduct ns-2 simulations to validate the proposed analytical model. Finally, we compare the delay and throughput performance of TCP VON and automatic repeat request (ARQ) network coding based TCP (TCP ARQNC). Simulation results show that TCP VON outperforms TCP ARQNC in terms of the average decoding delay and network throughput.**

I. Introduction

It has been shown that network coding can increase the throughput and improve the reliability of the end-to-end communication [1]–[3]. There are several related works for network coding in a practical setting [4]–[6]. In these works, the source divides native data packets into groups called *generations*. Network coding is performed within the same generation at the source and intermediate nodes. The source starts to transmit the next generation of the packets only after being acknowledged by the destination that the current generation of the packets have been decoded. The decoding delay of generation based network coding can be large since the destination has to receive enough independent coded packets before decoding a generation.

The concept of *online network coding* was proposed by Sundararajan *et al.* [7] and discussed further in [8], [9]. For online network coding, packets can be decoded consecutively instead of generation by generation. In [8], a new encoding rule was proposed to guarantee small decoding delay by recovering packet losses within reasonable time. In [9], the performance of online random linear network coding approach for time division duplexing channels under Poisson arrivals was studied. SlideOR [10] is an online network coding scheme, which uses a moving sliding window at the source to determine the set of native packets to be coded.

Recently, the joint problem of network coding and transport layer design has received attention. Hassayoun *et al.* [11] analyzed the performance of TCP in coded wireless mesh networks with random packet loss. Chen *et al.* [12] proposed a distributed rate control algorithm for network coding based on the utility maximization model. An automatic repeat request (ARQ) network coding based TCP protocol (referred to as TCP ARQNC in this paper) was proposed in [13] and extended in [14]. This approach gives a new solution for the sliding window based TCP algorithm. However, packets of one generation are decoded when the receiver receives enough independent coded packets. This can potentially lead to a large decoding delay.

In this paper, our goal is to design an online network coding based TCP for real time applications with a small decoding delay. Our proposed algorithm includes a congestion control part and an online network coding control part. For congestion control, we use TCP Vegas [15]. We can distinguish random packet losses from the congestion losses using the difference between the time that a packet is transmitted and the time that the sender is notified the packet is lost. The online network coding is used to address the effect of random packet loss for wireless links. The sender chooses to transmit either a packet with new information or a redundant packet according to the feedback information. Meanwhile, packets are decoded consecutively instead of generation by generation so that the average decoding delay is small. Another advantage of our protocol is that there is no need to change the protocol stack at the intermediate nodes. We only need to modify TCP in the transport layer at the sender and receiver. In summary, the main contributions of our work are as follows:

- We propose TCP Vegas with online network coding (TCP VON), which is a combination of TCP Vegas and online network coding with low decoding delay.
- We establish an analytical framework to model the TCP VON and derive the average decoding delay.
- We conduct ns-2 simulations to validate the analytical model and compare the performance of TCP VON with TCP ARQNC [13]. The results show that TCP VON outperforms TCP ARQNC in terms of network throughput and average decoding delay.

This paper is organized as follows: Section II describes the background. Our proposed TCP VON algorithm is presented in Section III. The analytical model of our TCP VON is presented in Section IV. The performance comparison is presented in Section V. Conclusions are given in Section VI.

II. Preliminaries and Basic Definitions

In this section, we first define different types of packets and then present the basic idea of how packets are coded in an online manner. According to the previous works in [4], [5], [13], there are two types of data packets in the network, namely *native packets* and *coded packets*. Native packets are the original packets generated by the sender, which are treated as vectors over a finite field $\mathbb{F}_q$ of size q. Each native packet has a unique index number k corresponding to the order it is generated. Let $\mathbf{p_k}$ denote the kth native packet. A coded packet is a linear combination of several native packets. For example, if a coded packet $\mathbf{q}$ is a linear combination of packets $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_m}$, then $\mathbf{q} = \sum_{i=1}^{m} \alpha_i \mathbf{p_i}$, where α_i is a non-zero multiplicative coefficient randomly selected from the field $\mathbb{F}_q$.

A native packet $\mathbf{p_k}$ is called a *sent* packet when the sender has transmitted a coded packet of the form $\mathbf{q} = \alpha_k \mathbf{p_k} + \sum_{l \neq k} \alpha_l \mathbf{p_l}$. In this paper, we use the term *transmit* for coded packets and the term *send* for native packets. A native packet $\mathbf{p_k}$ is *sensed* when the receiver has received a coded packet $\mathbf{q} = \alpha_k \mathbf{p_k} + \sum_{l \neq k} \alpha_l \mathbf{p_l}$. That is, $\mathbf{p_k}$ is combined in a coded packet $\mathbf{q}$, and $\mathbf{q}$ is received at the receiver. Otherwise, $\mathbf{p_k}$ is *unsensed*. As discussed in [13], a native packet $\mathbf{p_k}$ is defined as *seen*, when the receiver has enough information to compute a linear combination of the form $(\mathbf{p_k} + \mathbf{r_k})$, where $\mathbf{r_k} = \sum_{i>k} \alpha_i \mathbf{p_i}$ and $\alpha_i \in \mathbb{F}_q$. Otherwise, $\mathbf{p_k}$ is *unseen*. A native packet $\mathbf{p_k}$ is called *decoded* when the receiver has enough information to decode $\mathbf{p_k}$. Otherwise, $\mathbf{p_k}$ is *undecoded*.

The decoding mechanism at the destination is based on Gaussian elimination [10] [13] [14]. If we consider the native packets as unknown variables, then the set of received packets composes a system of linear equations. The decoding process is equivalent to solving a set of linear equations. The receiver can form a *decoding matrix* using the coefficients of coded packets. Gaussian elimination can be applied to convert this matrix to a reduced row echelon form and solve the linear equations. Linear operations corresponding to the Gaussian elimination are performed on the received packets.

At the sender side, let i_p, i_q and i_r denote the smallest index among the packets that have not been decoded, seen, and sensed, respectively. Let i_s denote the smallest index of the packet that has not been sent. Note that $i_p \leq i_q \leq i_r \leq i_s$. To transmit a new coded packet, the sender combines native packets with consecutive indices within a *coding window*. Let $i_{\min}$ and $i_{\max}$ denote the minimum and maximum index of the native packets combined in the new coded packet, respectively. That is, the coding window begins at $i_{\min}$ and ends at $i_{\max}$.

To create a new coded packet, it is required that $i_{\min} = i_q$. However, $i_{\max}$ can be equal to either $i_s - 1$ or i_s for two reasons: First, it is not necessary to have $i_{\min} < i_q$. Even if packet $\mathbf{p_l}$ (for $l < i_q$) is combined in the coded packet, since $\mathbf{p_l}$ has already been seen at the receiver, the Gaussian elimination will eliminate the term of $\mathbf{p_l}$. Because $\mathbf{p_{i_q}}$ may not be seen at the receiver, the sender must combine $\mathbf{p_{i_q}}$ in the new coded packet. Therefore, $i_{\min} = i_q$. We call the packet with index i_q the *code base*. Second, if $i_{\max} > i_s$, the transmitted packet contains at least two new variables for the receiver. However, it adds only one equation to the system of linear equations at the receiver. Therefore, to provide decoding opportunity at the receiver, $i_{\max} \leq i_s$. Since packet $\mathbf{p_{i_s-1}}$ has already been sent, we have $i_{\max} \geq i_s - 1$.

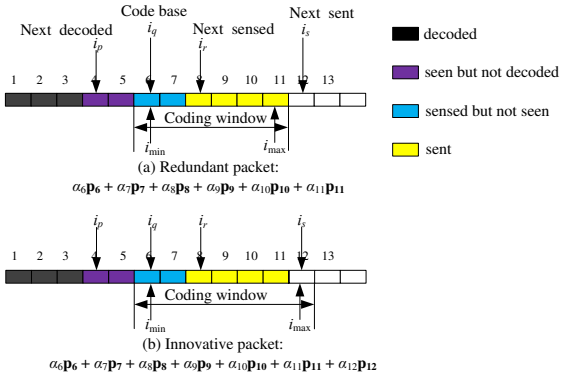


Fig. 1. Example of coded packets. $i_p = 4$, $i_q = 6$ and $i_r = 8$. The sender can transmit either (a) a redundant packet or (b) an innovative packet.

Consider the example in Fig. 1. If $i_{\max}$ is equal to i_s, then the transmitted packet is a linear combination of packets $\mathbf{p_{i_q}}, \mathbf{p_{i_q+1}}, \ldots, \mathbf{p_{i_s}}$. This packet contains information of a new native packet for the receiver and increases the number of variables at the receiver by one. We call such a packet an *innovative packet*. If $i_{\max}$ is equal to $i_s - 1$, then the coded packet is a linear combination of native packets $\mathbf{p_{i_q}}, \ldots, \mathbf{p_{i_s-1}}$, which is called a *redundant packet*.

For the case that there is no packet loss, the receiver can decode one native packet each time it receives an innovative packet. On the other hand, for the case that there are packet losses in the network, the sender should transmit redundant packets to compensate the effect of packet losses.

Let I_p, I_q and I_r denote the index of the next packet to be decoded, seen and sensed at the receiver, respectively. Note that i_p, i_q and i_r are variables at the sender while I_p, I_q and I_r are variables at the receiver and in the ACK packets. When an ACK arrives at the sender, i_p, i_q and i_r are updated by the values of I_p, I_q and I_r, respectively.

III. TCP VON Algorithm

In this section, we present the TCP VON algorithm for the sender and receiver, separately.

A. Sender Side Operation

1) Congestion and Flow Control: For the congestion and flow control, we use the TCP Vegas algorithm [15], which has two distinct advantages. First, TCP Vegas detects and reacts to congestion before it happens by monitoring the round trip time instead of detecting packet loss. Second, packet loss occurred on wireless links are not considered as a sign of congestion. The sender maintains several variables including congestion window size $cwnd$, measured round trip time RTT, the minimum of all measured round trip times $BaseRTT$, standard deviation of the measured round trip

time Δ, and the receive window size $rwnd$. The number of packets on the flight is approximated by the number of sent but unseen packets (i.e., $i_s - i_q$), which is limited by the congestion window size $cwnd$ and receive window size $rwnd$

$$i_s - i_q \leq \min\{cwnd, rwnd\}. \quad (1)$$

In order to focus on congestion control, we assume that the buffer size at the receiver is large enough so that the effect of the receive window $rwnd$ can be ignored [15], [16]. The congestion window size $cwnd$ is adjusted according to the algorithm from TCP Vegas [15]. We set two predetermined threshold values a and b, $a < b$. If $\left(\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}\right) \times BaseRTT < a$, then $cwnd$ is linearly increased. If $\left(\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}\right) \times BaseRTT > b$, then $cwnd$ is linearly decreased. In our algorithm, we select the default values of $a = 1$ packet and $b = 3$ packets.

2) Online Network Coding Control: The core of our online network coding control is to decide whether to transmit an innovative packet or a redundant packet such that the receiver can decode the packets with small decoding delay. We start by introducing the variables being used. The real gap $RealGap$ denotes the number of native packets sensed but unseen at the sender. That is, $RealGap = i_r - i_q$. The value of $RealGap$ is updated whenever the sender receives an ACK packet. $RealGap$ shows the number of lost innovative packets. If the $RealGap$ is zero, it indicates that all packets have been decoded at the receiver. If the $RealGap$ is greater than zero, it implies that there are packet losses in the system.

The old real gap $OldRealGap$ is the variable to store the previous value of $RealGap$. Whenever an ACK is received, $OldRealGap$ is set to be the value of $RealGap$ just before $RealGap$ is being updated. $RealGapDiff$ is the difference between $RealGap$ and $OldRealGap$ (i.e., $RealGapDiff = RealGap - OldRealGap$), which shows the decrease in real gap upon receiving an ACK packet. $RealGapDiff$ is calculated right after $RealGap$ has been updated.

The $ExpGap$ is the maximum number of lost packets which can be tolerated at the sender. It is updated whenever a redundant packet is transmitted or $RealGap$ is decreased. $State$ is a variable taking either $REDUNDANT$ or $INNOVATIVE$ to instruct whether the next packet to transmit is a redundant packet or an innovative one. If $RealGap > ExpGap$, then the number of lost packets exceeds the maximum number that the sender can tolerate and the $State$ will change into $REDUNDANT$. The sender transmits a redundant packet. If $RealGap \leq ExpGap$, $State$ is $INNOVATIVE$ and the sender transmits an innovative packet.

We now explain the use of $ExpGap$. First, both $RealGap$ and $ExpGap$ are set to zero. During operation, the $RealGap$ is equal to one, which indicates a packet loss. The sender transmits a redundant packet to compensate the packet loss and $ExpGap$ is increased by one. It causes $RealGap$ to be equal to $ExpGap$, and the sender does not transmit more redundant packets during the period when $RealGap$ is equal to $ExpGap$.

If the transmitted redundant packet gets lost in the network, the sender should be able to detect the loss of the redundant packet. Therefore, as soon as the sender transmits a redundant packet, it starts a timer with value $RTT + 4\Delta$, where 4Δ is used to provide some time margin. When the sender receives an ACK indicating that the redundant packet is successfully received by the receiver, the $RealGap$ is decreased by one before the timer expires. If $RealGap$ has not been decreased when the timer expires, it indicates that the redundant packet has got lost. In this case, the sender decreases $ExpGap$ by one, which causes $RealGap$ to be greater than $ExpGap$, so that the sender transmits another redundant packet. Note that one timer starts whenever a redundant packet is transmitted, and each redundant packet has its own timer.

If the transmitted redundant packet is received successfully at the receiver, then the sender is informed by an ACK packet indicating the decrease of $RealGap$ by one (i.e., $RealGapDiff = 1$). The sender decreases $ExpGap$ accordingly. The timer corresponding to the transmission of the redundant packet is stopped. $RealGapDiff$ can be larger than one, which indicates that more than one redundant packets have been received. In this case, $ExpGap$ is decreased by $RealGapDiff$. $RealGapDiff$ oldest timers are stopped.

3) Algorithm of Sender Side Operation: Algorithm 1 shows TCP VON algorithm at the sender. Lines 1 to 3 are used for initialization. Let i_m denote the total number of native packets created. As shown in Lines 5 to 8, if new data with X bytes arrives from upper layer, $n_s = \lceil \frac{X}{L} \rceil$ native packets are created, with length of L bytes (the last packet is padded with 0 if necessary). The total number of native packets is increased by n_s. Lines 9 to 22 show the online network coding algorithm. When $RealGap$ is greater than $ExpGap$, a redundant packet is transmitted, $ExpGap$ is increased by one, and a timer is started. Otherwise, an innovative packet is transmitted. When an ACK packet received (Line 23), the sender updates $BaseRTT$, Δ, i_p, i_q and i_r (Lines 24 to 25). Then, it calculates the value of $OldRealGap$, $RealGap$ and $RealGapDiff$ (Lines 26 to 28). If the $RealGapDiff$ is larger than zero, the sender will decrease $ExpGap$ accordingly and $RealGapDiff$ timers will be stopped. Then, the congestion control is performed in Lines 33 to 37. Lines 39 to 41 show the timeout event. The $ExpGap$ will be decreased by one when a timeout event occurs.

B. Receiver Side Operation

Algorithm 2 shows TCP VON algorithm at the receiver. I_p, I_q and I_r are initialized with value 1 and decoding matrix $\mathbf{D}$ is initialized with an empty matrix (Line 1). When the receiver receives a correct coded packet, it first checks the packet header and retrieves $i_{\min}$, $i_{\max}$ and the coding coefficients of native packets combined (Line 5). The coefficients are added as a new row to the decoding matrix and Gaussian elimination is executed (Lines 6 to 7). Operations corresponding to the Gaussian elimination are then performed on the received packets so far (Line 11). The receiver finds the next packet to be decoded in Lines 12 to 15. The sender increases the value of I_p by one until it finds that the native packet with index I_p is not decoded. Thus, I_p is equal to the index of

Algorithm 1 Algorithm of TCP VON at the Sender.

1: Set i_p, i_q, i_r, i_s, $i_{\min}$, and $i_{\max}$ to one and i_m to zero
2: Set $RealGap$ and $ExpGap$ to zero
3: Set $State$ to $INNOVATIVE$ and $cwnd$ to one
4: **while** the TCP connection is established
5: **if** data with length X bytes is received from upper layer
6: Segment data in $n_s := \lceil \frac{X}{L} \rceil$ packets $\mathbf{p_{i_m+1}}, \ldots, \mathbf{p_{i_m+n_s}}$
7: Set $i_m := i_m + n_s$
8: **end if**
9: **while** $i_s - i_q \leq cwnd$ **and** $i_{\max} \leq i_m$
10: **if** $RealGap > ExpGap$
11: Set $State := REDUNDANT$
12: Set $i_{\min} := i_q$, $i_{\max} := i_s - 1$
13: Set $ExpGap := ExpGap + 1$
14: Start a countdown timer with value $RTT + 4\Delta$
15: **else**
16: Set $State := INNOVATIVE$
17: Set $i_{\min} := i_q$, $i_{\max} := i_s$, $i_s := i_{\max} + 1$
18: **end if**
19: Create coded packet $\mathbf{q} = \sum_{j=i_{\min}}^{i_{\max}} \alpha_j \mathbf{p_j}$
20: Include $i_{\min}$, $i_{\max}$ and $\alpha_{i_{\min}}, \ldots, \alpha_{i_{\max}}$ in header
21: Transmit the coded packet
22: **end while**
23: **if** an ACK is received
24: Set $i_p := I_p$, $i_q := I_q$, $i_r := I_r$
25: Compute $BaseRTT$ and Δ
26: Set $OldRealGap := RealGap$
27: Set $RealGap := i_r - i_q$
28: Set $RealGapDiff := RealGap - OldRealGap$
29: **if** $RealGapDiff > 0$
30: Set $ExpGap := ExpGap - RealGapDiff$
31: Stop $RealGapDiff$ oldest timers.
32: **end if**
33: **if** $\left(\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}\right) \times BaseRTT > 3$
34: Set $cwnd := cwnd - 1$ in the next RTT
35: **elseif** $\left(\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}\right) \times BaseRTT < 1$
36: Set $cwnd := cwnd + 1$ in the next RTT
37: **end if**
38: **end if**
39: **if** a timer timeouts
40: $ExpGap := ExpGap - 1$
41: **end if**
42: **end while**

Algorithm 2 Algorithm of TCP VON at the Receiver.

1: Set $I_p := 1$, $I_q := 1$, $I_r := 1$, $\mathbf{D} := [\,]$
2: **while** TCP connection is established
3: **if** a packet is received
4: **if** packet is not corrupted
5: Retrieve $i_{\min}$, $i_{\max}$, and the coding coefficients
6: Add coding coefficients as a new row to matrix $\mathbf{D}$
7: Perform Gaussian elimination on $\mathbf{D}$
8: **if** an all-zero row appears
9: Discard the packet and eliminate the row in $\mathbf{D}$
10: **else**
11: Perform linear operations to packets corresponding to the Gaussian elimination
12: **while** native packet with index I_p is decoded
13: Deliver data in $\mathbf{p_{I_p}}$ to upper layer
14: $I_p := I_p + 1$
15: **end while**
16: **while** native packet with index I_q is seen
17: $I_q := I_q + 1$
18: **end while**
19: **while** native packet with index I_r is sensed
20: $I_r := I_r + 1$
21: **end while**
22: Send an ACK packet indicating I_p, I_q and I_r
23: **end if**
24: **else**
25: Discard the corrupted packet
26: **end if**
27: **end if**
28: **end while**

the first undecoded packet when the *while loop* is terminated. Similarly, the receiver will find the first packet unseen in Lines 16 to 18, and the first packet unsensed in Lines 19 to 21. Finally, an ACK packet indicating the indices I_p, I_q and I_r is generated and sent at the receiver (Line 22).

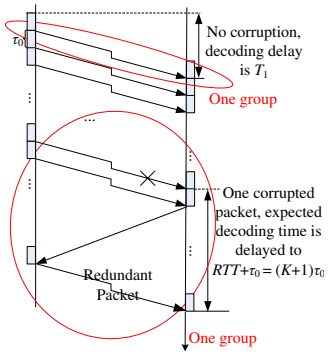
IV. Decoding Delay Analysis of TCP VON

In this section, we establish analytical model to determine the decoding delay of TCP VON. The *decoding delay* is the difference between the time that a packet is transmitted by the sender and the time it is decoded at the receiver. We consider a topology where the last hop is a wireless bottleneck link with capacity C. The sender has a large file to send. The packet size is L bits. The propagation delay from the sender to the receiver is T_0. Due to the congestion control mechanism of TCP Vegas, the transmission rate at the sender is C, and there is no congestion loss. The time to transmit a packet is $\tau_0 = L/C$. The end-to-end delay from the sender to the receiver and from the receiver to the sender are approximately $T_1 = T_0 + 3L/C$ and $T_2 = T_0$, respectively. The round trip time is $RTT = T_1 + T_2$. We use K to denote the ratio of RTT over τ_0. We approximate K with $\lceil RTT/\tau_0 \rceil$. Since RTT is much larger than τ_0, the error of approximation is negligible.

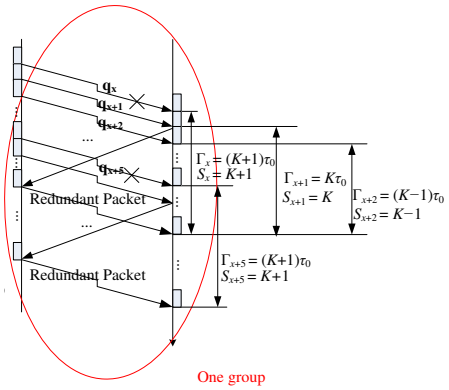
Each packet experiences random loss at the wireless link independently with probability p. Equivalently, we assume that packets are corrupted independently with probability p. Let $\mathbf{q_k}$ denote the kth coded packet received.

A *group* of packets is a set of packets which are decoded together. The *expected decoding time* of a packet $\mathbf{q_k}$, denoted as Γ_k, is the difference between its arrival time and the time it is expected to be decoded if no further packet corruptions happen. If there are no corrupted packet arrivals before $\mathbf{q_k}$ and $\mathbf{q_k}$ is correctly received, Γ_k is equal to 0. However, if $\mathbf{q_k}$ is corrupted, Γ_k will be delayed up to $RTT + \tau_0$. It is because the receiver has to wait for one redundant packet which is expected to arrive after $RTT + \tau_0$ seconds. There will be several packets in one group in this case.

In Fig. 2(a), if there is no corrupted packet arrival at the receiver, each packet can be decoded immediately with decoding delay T_1. If a corrupted packet arrives, the consecutive innovative packets cannot be decoded until a redundant packet arrives. In Fig. 2(b), when a corrupted packet $\mathbf{q_x}$ arrives, the expected decoding time is $RTT + \tau_0 = (K+1)\tau_0$. If it receives a correct packet (e.g., packet $\mathbf{q_{x+1}}$ or $\mathbf{q_{x+2}}$), the expected decoding time decreases by τ_0. However, if there is another corrupted packet $\mathbf{q_{x+5}}$, the expected decoding time is



(a) Zero or one packet corruption in one group.



(b) Multiple packet corruptions in one group.

Fig. 2. Groups of packets with different number of corrupted packets.

$(K+1)\tau_0$. The expected decoding time Γ_{k+1} is determined based on Γ_k and whether $\mathbf{q_{k+1}}$ is corrupted or correct. Thus, Γ_{k+1} is independent of $\Gamma_{k-1},\ldots,\Gamma_1$. We use S_k to denote the ratio of Γ_k to τ_0 (i.e., $S_k = \frac{\Gamma_k}{\tau_0}$). Therefore, S_{k+1} is independent of $S_{k-1},\ldots,S_1$.

The sequence $S_1, S_2, \ldots$ constructs a discrete Markov chain. If a corrupted packet arrives, the expected decoding time becomes $(K+1)\tau_0$ and the state becomes $K+1$. If a packet is correctly received, the expected decoding time is decreased by τ_0 and the state decreases by 1. Fig. 3 shows the state transition of the Markov chain. Let $\mathbf{M} = \{m_{jk}\}_{(K+2)\times(K+2)}, j = 0,1,\ldots K+1, k = 0,1,\ldots K+1$ denote the transition probability matrix of the Markov chain. $m_{j(K+1)} = p$ $(j = 0,1,\ldots,K+1)$, $m_{(j+1)j} = 1-p$ $(j = 0,1,\ldots,K)$, $m_{00} = 1-p$ and all the other entries of $\mathbf{M}$ are 0.

When the state is 0, all the packets can be decoded. If there are i packets in one group, there will be i transitions between two consecutive visits of state 0 accordingly. Let P_i denote the probability that there are i packets in one group. Matrix $\mathbf{M}$ can be divided into four submatrices as

$$\mathbf{M} = \left(\begin{array}{c|c} \mathbf{M}_{00} & \mathbf{M}_{01} \\ \hline \mathbf{M}_{10} & \mathbf{M}_{11} \end{array}\right) \quad (2)$$

$$= \left(\begin{array}{c|ccc} m_{00} & m_{01} & \cdots & m_{0(K+1)} \\ \hline m_{10} & m_{11} & \cdots & m_{1(K+1)} \\ \vdots & \vdots & \ddots & \vdots \\ m_{(K+1)0} & m_{(K+1)1} & \cdots & m_{(K+1)(K+1)} \end{array}\right).$$

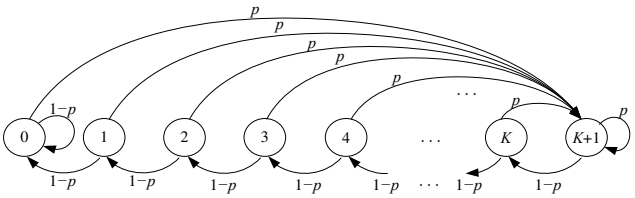


Fig. 3. State transition of Markov chain.

We have $P_1 = m_{00}$ and $P_2 = \sum_{j=1}^{K+1} m_{0j}m_{j0} = \mathbf{M}_{01}\mathbf{M}_{10}$. In general, we have

$$\begin{aligned} P_i &= \sum_{j_1=1}^{K+1}\sum_{j_2=1}^{K+1}\cdots\sum_{j_{i-1}=1}^{K+1} m_{0j_1}m_{j_1j_2}\ldots m_{j_{i-2}j_{i-1}}m_{j_{i-1}0} \\ &= \mathbf{M}_{01}\mathbf{M}_{11}^{i-2}\mathbf{M}_{10}, \qquad i \geq 3. \end{aligned} \quad (3)$$

The decoding delay of the ith (last) packet in the group with i packets is equal to the propagation delay T_1. The decoding delay of the $(i-1)$th packet in the group is $T_1+\tau_0$. The decoding delay of the $(i-j)$th $(j = 0,1,\ldots,i-1)$ the packet in the group is $T_1 + j\tau_0$. The average decoding delay of a group with i packets is

$$\tau_i = \textstyle\sum_{j=0}^{i-1}(T_1+j\tau_0)/i = T_1 + (i-1)\tau_0/2, \qquad i \geq 1. \quad (4)$$

Let t denote the total transmission time, $\tau_T(t)$ denote the total decoding delay of all the packets during time t and $N(t)$ denote the total number of packets transmitted during time t. We use $n_i(t)$ to denote the number of groups with i packets in total during time t. We have $N(t) = \sum_{i=1}^{\infty} i n_i(t)$ and $\tau_T(t) = \sum_{i=1}^{\infty} i n_i(t)\tau_i$. For large values of t, by the law of large numbers, the limit of $n_i(t)/\sum_{j=1}^{\infty} n_j(t)$ approaches P_i, the probability that there are i packets in one group. Thus, the average decoding delay can be computed as

$$\begin{aligned} \overline{\tau} &= \lim_{t\to\infty}\frac{\tau_T(t)}{N(t)} = \lim_{t\to\infty}\frac{\sum_{i=1}^{\infty} i n_i(t)\tau_i}{\sum_{i=1}^{\infty} i n_i(t)} \\ &= \lim_{t\to\infty}\frac{\sum_{i=1}^{\infty} i\left(n_i(t)/\left(\sum_{j=1}^{\infty} n_j(t)\right)\right)\tau_i}{\sum_{i=1}^{\infty} i\left(n_i(t)/\left(\sum_{j=1}^{\infty} n_j(t)\right)\right)} = \frac{\sum_{i=1}^{\infty} iP_i\tau_i}{\sum_{i=1}^{\infty} iP_i}. \end{aligned}$$

V. Performance Analysis

In this section, we analyze the performance of TCP VON via ns-2 simulations. We first validate our analytical model by comparing the analytical and simulation results. Then, we compare the throughput and delay performance of TCP VON and TCP ARQNC [13], [14]. Unless stated otherwise, the packet size L is set to 1250 bytes. The results are average on 50 simulation runs and each simulation lasts 100 seconds.

We consider a single-hop wireless network with one sender and one receiver. C is the capacity of the wireless link. Fig. 4 shows the comparison between the analytical and simulation results under different loss probabilities p. The propagation delay from the sender to the receiver T_0 is 5 ms. The random packet loss probability varies from 0 to 0.1 with step of 0.01. The results show that the simulation results match the analytical results, validating the correctness of our analytical model.

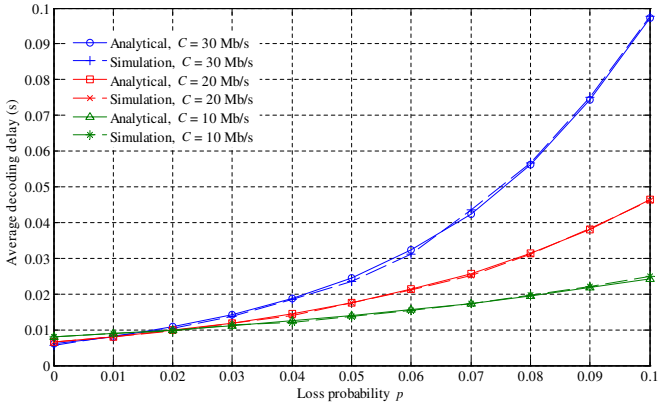


Fig. 4. Analytical and simulation results of average decoding delay for single hop topology under different values of p ($T_0 = 5$ ms, and $L = 1250$ bytes).

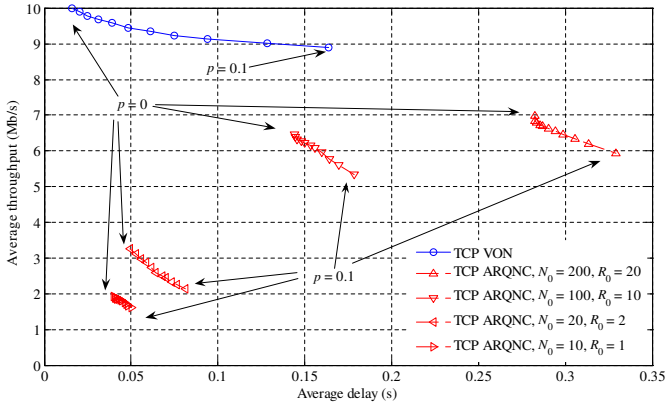


Fig. 5. Performance comparison for three-hop tandem topology under different values of packet loss probability p.

Next, we compare the performance of TCP ARQNC [13], [14] and TCP VON. For TCP ARQNC, the coded packets are transmitted in a generation by generation fashion. There are N_0 native packets in one generation and each coded packet is a linear combination of the N_0 native packets. For each generation, the sender first transmits $N_G = N_0 + R_0$ coded packets, where R_0 is the redundant factor. The receiver can decode all the native packets if N_0 or more coded packets are successfully received at the receiver. If more than R_0 packets are lost, the sender has to transmit more linear combinations until the receiver receives enough independent coded packets.

We consider a three-hop tandem topology for simulations. The first two hops are wired links with capacity of 50 Mb/s and propagation delay of 5 ms. The last hop is wireless link with capacity of 10 Mb/s, propagation delay of 2 ms, and random packet loss probability p. There is a TCP session between the source and destination.

We investigate the joint throughput and delay performance of the different TCP schemes using ns-2 simulation. Fig. 5 shows the throughput and delay performance of TCP VON and TCP ARQNC when packet loss probability p changes from 0 to 0.1 with step of 0.01. The throughput performance of TCP ARQNC is much worse than that of TCP VON when N_0 and R_0 are small. When we increase N_0 and R_0, although the throughput of TCP ARQNC is increased, the decoding delay performance degrades a lot. Both of the throughput and delay performance of TCP VON are better than that of TCP ARQNC when p is small (e.g., $p < 0.05$).

VI. Conclusions

In this paper, we proposed TCP VON, which incorporates online network coding into TCP. By executing online network coding control, packets can be decoded consecutively and the decoding delay is small. Then, we established a Markov chain model to compute the analytical delay performance of TCP VON. We also conducted ns-2 simulations to validate the correctness of our analytical models. Finally, we compared the delay and throughput performance of TCP VON and TCP ARQNC. Results showed that TCP VON outperforms TCP ARQNC. For future work, we plan to carry out empirical study of TCP VON and apply it in real networks.

References

[1] R. Ahlswede, N. Cai, S. Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. on Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[2] S. Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. on Information Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[3] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. on Networking*, vol. 11, no. 5, pp. 371–381, Oct. 2003.

[4] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. of 41st Allerton Annual Conference on Communication, Control, and Computing*, Urbana-Champaign, IL, Oct. 2003.

[5] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "XORs in the air: Practical wireless network coding," *IEEE/ACM Trans. on Networking*, vol. 16, no. 3, pp. 497–510, June 2008.

[6] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in *Proc. of ACM SIGCOMM*, Kyoto, Japan, Aug. 2007.

[7] J. K. Sundararajan, D. Shah, and M. Médard, "ARQ for network coding," in *Proc. of IEEE International Symposium on Information Theory*, Toronto, Canada, Jul. 2008.

[8] J. Barros, R. A. Costa, D. Munaretto, and J. Widmer, "Effective delay control in online network coding," in *Proc. of IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009.

[9] D. E. Lucani, M. Médard, and M. Stojanovic, "Online network coding for time-division duplexing," in *Proc. of IEEE GLOBECOM*, Miami, FL, Dec. 2010.

[10] Y. Lin, B. Liang, and B. Li, "SlideOR: Online opportunistic network coding in wireless mesh networks," in *Proc. of IEEE INFOCOM*, San Diego, CA, Mar. 2010.

[11] S. Hassayoun, P. Maille, and D. Ros, "On the impact of random losses on TCP performance in coded wireless mesh networks," in *Proc. of IEEE INFOCOM*, San Diego, CA, Mar. 2010.

[12] L. Chen, T. Ho, S. H. Low, M. Chiang, and J. C. Doyle, "Optimization based rate control for multicast with network coding," in *Proc. of IEEE INFOCOM*, Anchorage, AK, May 2007.

[13] J. K. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. Barros, "Network coding meets TCP," in *Proc. of IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009.

[14] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets TCP: Theory and implementation," *Proc. of the IEEE*, vol. 99, no. 3, pp. 490–512, Mar. 2011.

[15] L. Brakmo and L. Peterson, "TCP Vegas: End-to-end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communication*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[16] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. of ACM SIGCOMM*, Vancouver, Canada, Sept. 1998.