Network Coding for Severe Packet Reordering in Multihop Wireless Networks

Chansook Lim

Department of Computer and Information Communications Engineering, Hongik University, Sejong 30016, Republic of Korea

Correspondence should be addressed to Chansook Lim; chansooklim@hongik.ac.kr

Received 30 January 2015; Revised 21 June 2015; Accepted 9 July 2015

Academic Editor: Antonino Staiano

Copyright © 2015 Chansook Lim. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Network coding is known to be effective in overcoming packet losses and packet reordering in multihop wireless networks. Despite the benefits, network coding is hard to deploy without being compatible with TCP. To address this problem, a seminal paper proposed a network coding scheme that adopts an ACK-based sliding-window network coding approach. In this paper we show that the previous scheme may not suffice to mitigate the effects of packets received out of order in multipath wireless networks where severe packet reordering persists. We propose a modified network coding layer where the receiver acknowledges every degree of freedom by using the sequence number of a newly seen packet instead of using that of the oldest unseen packet so that the network coding layer can be compatible with a TCP variant for severe packet reordering. To reduce the decoding matrix size and the coding buffer size, our scheme allows retransmission at the network coding layer. Simulation results show that our scheme outperforms the exiting scheme in multipath wireless networks particularly when severe packet reordering persists.

1. Introduction

The emergence of wireless multimedia sensor networks (WMSNs) [1] poses new research issues on protocols through several layers. In WMSNs, multimedia data such as images, video, and audio data is transferred in high data rates. Due to the characteristics of multimedia data, some issues including reliability, congestion control, and use of multiple paths become more critical than in classical wireless sensor networks. Applying standard transport protocols such as TCP will be one of the approaches to the issues. Particularly, IP-based wireless sensor networks [2] will likely facilitate TCP implementation. However, there are well-known TCP-related problems that most of multihop wireless networks including WMSNs and wireless ad hoc networks have in common. We consider a network coding approach to address these problems.

Network coding was introduced as an effective means to overcome high packet loss rates in wireless networks. Due to the broadcast nature of wireless networks, interflow network coding is effective in reducing the number of packet transmissions and packet losses. Also intraflow network coding is useful for schemes exploiting path diversity of multihop wireless networks such as opportunistic routing. Benefits of network coding in multihop wireless networks have been presented through many real systems including COPE and MORE [3, 4].

Network coding also has been noted for the potential to address the TCP throughput problem in wireless networks. How to obtain high TCP throughput in wireless networks is one of the research topics that have long been studied. TCP was originally designed based on the assumption that packet losses are indicative of congestion. In wireless networks, however, packet losses and reordering occur also due to time varying channels and so forth. Therefore the existing TCP implementations unnecessarily reduce congestion window reacting to random losses or packet reordering. Although a lot of studies have been focused on how to make TCP differentiate congestion from random losses and packet reordering in wireless networks [5, 6], the problem still remains difficult. Network coding can mask packet losses and reordering from TCP to some extent, because a random linear combination
of TCP segments rather than each TCP segment itself is sent with redundancy. Hence we can consider network coding as a means to address the TCP performance problem in wireless networks.

Benefits of network coding for TCP performance do not necessarily mean compatibility with TCP. A problem is that the encoding/decoding operation performed on a batch of packets does not lend itself to TCP, which is a sliding-window protocol. The seminal work to address this problem, [7], introduced a new network coding layer between a transport layer and a network layer. The approach of [7] called TCP/NC has two major features. First, whenever the source is allowed to transmit, it sends a random linear combination of all packets in the congestion window. Second, instead of acknowledging the original packets, the receiver acknowledges degrees of freedom by sending the source the sequence numbers of “seen” packets. That is, TCP/NC allows an ACK-based sliding-window network coding. If it is always possible to send a random linear combination of all packets in the congestion window, the ACK number would indicate the number of innovative coded packets received with high probability.

However, since the coding header cannot be indefinitely long in practice, the number of packets to be mixed is limited. Given this restriction, the TCP/NC network coding layer may not mask all packet reordering from TCP when severe packet reordering persists. This leads to TCP performance degradation. We argue that in order to handle severe packet reordering, it is more effective to use a TCP variant for persistent packet reordering such as TCP-PR [8].

In this paper, we propose a new network coding layer that is compatible with a TCP variant robust to persistent packet reordering. To this end, we make two major changes on the original TCP/NC network coding layer. First, the receiver acknowledges every degree of freedom by sending the source the sequence numbers of a newly seen packet instead of that of the oldest unseen packet. By doing this, the source can figure out which packet is newly seen, even when coded packets are received out-of-order and therefore the order in which packets are seen is extremely different from the order in which they were sent from the source. Second, the source can perform retransmission when it is deemed to help reduce the decoding buffer size. This can mitigate the problem of a large coding buffer required to distinguish packet loss from packet reordering when packet reordering is severe.

To evaluate the performance of our new scheme, we simulated 2-path multihop wireless networks using ns-2. We generated severe packet reordering by transmitting UDP background traffic with a high data rate through one of the two paths. Simulation results show that as packet reordering gets more severe due to higher background traffic, our new scheme increasingly outperforms the original TCP/NC scheme.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 presents the background and discusses a motive for proposing a new network coding layer. Section 4 describes our new network coding layer and Section 5 discusses the performance of our new scheme. In Section 6 we conclude.

2. Related Work

In wireless networks, TCP often suffers from performance degradation, because the existing TCP cannot differentiate congestion from random losses. When exploiting path diversity is pursued, a packet reordering problem is added on top of it. Therefore, how to enable TCP to differentiate congestion from random packet losses and packet reordering has long been studied [6, 9]. Nevertheless the problem has not been completely solved. Since network coding can mask random packet losses and packet reordering from TCP to some extent, it can help a TCP sender avoid reducing congestion window unnecessarily.

For real implementation of network coding, compatibility with TCP has been considered an important issue [4]. A network coding forwarding scheme may generate packet reordering or may not be able to sufficiently mask packet reordering from TCP when packet reordering is severe. The resulting packet reordering may degrade TCP performance. Besides, there are other issues such as effects of decoding delay.

In most approaches to network coding, the encoding/decoding operation is performed on a batch of packets. If the source generates a stream of random linear combinations of a batch of $k$ packets, the receiver can decode the batch once it receives any set of at least $k$ linear combinations. One of the important problems of a batch-based approach is that it does not lend itself to a sliding-window protocol such as TCP. Until the entire batch has been received and decoded, there is no guarantee that the receiver will be able to extract any of the original packets and pass it on to higher layers. As a consequence, packets are acknowledged only at the end of a batch. Until all the packets in a batch are decoded, the sending window cannot proceed and a long decoding delay can cause a TCP timeout. To address this problem, [7] introduced a new network coding layer called TCP/NC which sits between a transport layer and a network layer. TCP/NC that allows an ACK-based sliding-window network coding approach has two major features. First, whenever the source is allowed to transmit, it sends a random linear combination of all packets in the congestion window. Second, the receiver acknowledges degrees of freedom and not individual packets.

Thereafter many schemes have employed the approach of [7]. CoMP [10], a network coding multipath forwarding scheme, uses multipath online network coding of [7]. CoMP estimates loss probabilities in order to adapt the rate of redundant linear combinations and allows reencoding on a hop-by-hop basis. The other feature of CoMP is that a backpressure methodology is used for congestion control and a credit-based method is used to prevent intermediate nodes from sending too many noninnovative packets. For compatibility with network coding, [11] proposed a coded transport protocol, CTCP, which uses network coding directly. CTCP acknowledges degrees of freedom as in [7] but uses a block coding approach instead of a sliding-window approach. The reason for not using a sliding-window approach is that the receiver may not be able to decode even the first packet of a file until the entire file is received. To make congestion control work well with network coding based on acknowledging
degrees of freedom, CTCP uses a notion of “token” that allows the CTCP sender to transmit a packet. Sending a packet means using a token, whereas receiving an ACK means generating a token. Our aim is to design a new network coding layer which is based on TCP/NC, but modified, so that it can help avoid TCP performance degradation in the presence of severe packet reordering.

To improve performance of transport protocol for multi-homing devices, some schemes incorporate network coding with MPTCP (multipath TCP) [12]. The main feature of MPTCP is that the MPTCP sender opens multiple TCP subflows each of which is assigned a path. Each subflow maintains its own congestion window. An MPTCP sender stripes packets across these subflows. MPTCP has been extensively studied particularly for multihomed environments. NC-MPTCP in [13] is a mixed approach of MPTCP and network coding to address the head-of-line blocking problem that a multipath transfer scheme may suffer from. This scheme tries to avoid a receiver buffer becoming full by mixing network coded subflows and regular subflows. The weakness of this scheme is that despite the guideline, it will not be easy to decide which subflows should be network coded among all the subflows particularly when the conditions of the paths change dynamically. MPTCP/NC of [14] incorporates network coding with MPTCP to utilize multiple network technologies in heterogeneous networks for mobile devices. In MPTCP/NC, all subflows of MPTCP use network coding to aid in the subflow scheduling problem by eliminating the need to track specific packets sent over the network.

Besides these, there are other approaches to address the TCP performance problem in coded networks. Reference [15] proposes a network coded multipath scheme called CodeMP to improve TCP performance in disruptive MANETs. CodeMP uses ACK Piggy coding to mitigate self-interference due to contention for a shared wireless channel between TCP data and its ACKs. Reference [16] examines the fairness problem between regular TCP flows and coded TCP flows in a bottleneck topology.

Although most of the above schemes consider multipath scenarios, they either do not consider severe packet reordering or employ MPTCP. Note that schemes using MPTCP can perform best in multihoming scenarios.

3. Background

3.1. TCP/NC. As described in Section 2, the main feature of TCP/NC [7] is that the source sends a random linear combination of all packets in the congestion window and the receiver acknowledges degrees of freedom and not individual packets. “Acknowledging degrees of freedom” means that the receiver sends an ACK for a received packet if the packet is “seen.” Loosely speaking, the definition of “seeing a packet p” is that the packet p can be decoded if the receiver can decode all the packets following p in terms of the order in which the packets are sent at the source. More formally, the authors of [7] define “seeing a packet” as follows. Let \( p_k \) be a packet and \( q \) a linear combination involving packets with indices larger than \( k \). “A node has seen a packet \( p_k \)” means that the node has enough information to compute a linear combination of the form \((p_k + q)\). The number of seen packets is always equal to the dimension of the knowledge space or the number of degrees of freedom that has been received so far. The receiver generates a new TCP ACK with the sequence number equal to that of the oldest unseen packet. Given that these sequence numbers are always obtained in an increasing order, they indicate the numbers of innovative coded packets received.

In TCP/NC, the source can transmit redundant linear combinations in order to compensate for packet losses. The redundancy factor is chosen based on the effective loss probability on the links so that the receiver can collect linear combinations at the same rate as the rate at which packets are mixed by the encoder.

3.2. Effect of Severe Packet Reordering on TCP/NC. In theory, TCP/NC allows all the packets in the coding buffer to be encoded for transmission. However the coding header size scales with the number of the packets involved in the linear combination. Since the coding header cannot be indefinitely long in practice, the number of packets to be mixed is limited. In [7] only a constant-sized subset of the packets chosen from within the coding buffer is mixed. This subset is called the coding window. Given the maximum coding window size \( W \), the source can mix at most \( W \) packets from the coding window.

The authors of [7] observed in their simulation that low TCP throughput persisted for a long time period. A possible reason for this problem they guessed is that a small field might cause received linear combinations to be noninnovative or might cause packets to be seen out of order, resulting in duplicated ACKs. They mention that the probability that such problems persist for a long time falls rapidly with the field size. However it is not clear in [7] how large the field size should be.

In [11], the problem of the decoding delay and complexity associated with TCP/NC is pointed out. Although the sliding-window approach of TCP/NC allows for better throughput, the receiver may not be able to decode even the first packet of the file until the entire file is received. This may lead to high decoding delay. To address this problem, our scheme allows the network coding layer to retransmit a packet which is deemed to help decoding and reduce the size of the decoding matrix.

These weaknesses can be aggravated in a multipath environment where the main side effect is packet reordering. Network coding essentially masks packet loss and reordering to some extent. However in the presence of severe packet reordering, the TCP/NC layer cannot mask all the reordering from the TCP layer. Figure 1(a) was presented in [7] to show the notion of a seen packet in terms of basis matrix. The basis matrix is in reduced row echelon form (RREF) obtained by performing Gauss-Jordan elimination on the coefficient matrix. Figure 1(a) implies that packets are “seen” in an increasing order in terms of the sequence number. However if severe packet reordering persists, the sequence numbers of seen packets are not necessarily obtained in an increasing order. Thus the basis matrix may often look like Figure 1(b). In
this situation, the TCP/NC receiver will send duplicate ACKs because it generates a TCP ACK with the sequence number equal to that of the oldest unseen packet. Note that the TCP/NC source delivers TCP ACKs to the TCP layer. Upon receiving three duplicate ACKs, the TCP source wrongly determines the fact that the packet was lost. This leads to unnecessary congestion window reduction. This implies that, with TCP/NC, the existing standard TCP is not sufficient to handle severe packet reordering.

3.3. TCP for Persistent Packet Reordering and Our Approach. Since it is well known that the existing TCP performs poorly when severe packet reordering persists, many schemes to address this problem have been proposed. Among them, TCP-PR, a TCP variant for persistent packet reordering, does not use DUPACKs and relies only on timers to detect packet losses [8]. To this end, TCP-PR records the timestamp for each sent packet. If a packet remains for a longer time period than a given threshold, the packet is assumed lost. The threshold is set to the timestamp of the packet plus an estimated maximum possible round-trip time, mxrtt. Whenever an ACK is received, mxrtt is updated as follows: mxrtt := β × srtt, where β denotes a constant greater than 1 and srtt an exponentially weighted average of past RTTs. As a new ACK arrives, srtt is continuously updated as follows: srtt := max{α⁻¹/cwnd} × srtt, samplertt, where α denotes a positive constant smaller than 1, [cwnd] the floor of the current congestion window size, and samplertt the RTT for the packet whose ACK just arrived. The parameter α can be interpreted as a smoothing factor in units of RTTs.

Once a packet loss is detected, congestion window in TCP-PR can be updated in a similar way to standard TCP variants. In slow-start mode, cwnd increases exponentially each RTT. When the first loss occurs, cwnd is halved and the sender transitions to congestion-avoidance mode in which cwnd increases linearly until a packet loss occurs. Subsequent packet losses cause further halving of cwnd, but the sender remains in congestion-avoidance mode. To prevent bursts of packet losses from causing excessive decreases of cwnd, the sender, upon detecting a packet drop, memorizes the snapshot of the packets that have not been acknowledged. In this way this memorized packet list contains only those packets that were sent before cwnd was halved but have not been acknowledged. If a packet in the memorized list is declared dropped, cwnd is not halved.

In real-world implementation, the receiver can use the SACK option which is supported by most commercial TCP implementations. With the SACK option, the TCP-PR sender can figure out the sequence numbers of received packets and open the cwnd accordingly. Note that, in TCP-PR, the receiver sends an ACK with the sequence number of a newly received packet instead of the sequence number of the oldest packet to be received. Thus, TCP-PR is not compatible with the network coding layer that sends an ACK with the sequence number of the oldest unseen packet. This is the case for all other TCP variants relying on timers instead of duplicate ACKs to determine packet losses.

4. Proposed Network Coding

We propose a new network coding layer that is compatible with TCP-PR described in Section 3. To this end, we make two major changes on the original TCP/NC layer proposed in [7]. First, the receiver acknowledges a degree of freedom by sending the sequence number of newly seen packet instead of that of the oldest unseen packet. In this way the sender can figure out which packet is newly seen even when coded packets are received out-of-order and the order of seen packets is extremely different from the order in which they were sent from the source. In case of TCP-PR, receiving an ACK with the sequence number of a newly seen packet,
the sender can increase the congestion window accordingly. Therefore the effect of severe packet reordering can be kept to a minimum.

Second, the source network coding layer can perform retransmission when it is deemed to help reduce the decoding matrix size and the coding buffer size. The original TCP/NC does not require a large decoding matrix at the receiver or a large coding buffer at the source. Since the receiver sends a TCP ACK with the sequence number of the oldest unseen packet and the TCP source, upon receiving three duplicate ACKs, retransmits, unseen packets are quickly recovered. However, if packet reordering persists severely, the original TCP/NC frequently performs spurious retransmissions and unnecessary reduction of congestion window. As mentioned before, for compatibility with TCP-PR, our scheme sends an ACK with the sequence number of a newly seen packet. At the core of TCP-PR and several other TCP variants robust to packet reordering is the idea that the TCP source needs to wait long enough to distinguish packet reordering from packet losses considering RTT variance. This implies two things: (1) the TCP-PR source needs a larger buffer than the existing standard TCP variants and (2) it takes longer time lost packets to be recovered. Since the TCP-PR receiver needs to wait longer until recovery of lost packets, the receiver network coding layer needs a large decoding matrix enough to accommodate all undecoded linear combinations. Consequently, the coding buffer at the source network coding layer should be large enough as well. One possible approach to reduce the size of the coding buffer and decoding matrix is to perform retransmission at the network coding layer.

In the original TCP/NC, the sender can transmit redundant linear combinations to compensate for packet losses. However, when packet reordering is severe, redundant transmission may not be enough to help reduce the space for coding and decoding. Our approach allows the network coding layer to retransmit linear combinations deemed necessary instead of blindly relying only on transmission of redundant linear combinations. Retransmission must be dealt with carefully. While it is desirable to reduce the coding buffer by retransmitting a packet to help decoding, the TCP source must be informed of a packet loss due to congestion as soon as possible so that the sending rate can be lowered. Masking too many losses from the TCP layer will result in excessive traffic from the TCP source. This is a serious problem particularly in multihop wireless networks where bandwidth is scarce resource.

To help the source network coding layer determine which packet to retransmit, the sequence number of the newest one among the consecutively decoded packets (i.e., the one just before the oldest undecoded packets) is contained in the coding header of the ACK packet. Retransmission is considered if the distance between the highest ACK number and the oldest undecoded sequence number becomes too far, because this means the decoding matrix also is too large. To reflect the fact that the degree of packet reordering affects the size of the decoding matrix, we measure “distance” between the highest ACK and the current ACK. Each time a new distance is measured, s_dist, which is an exponentially weighted average of past distances, and max_distance are calculated as follows:

\[
s_{\text{dist}} := \max\{\text{distance}, 0.9^{10} \times s_{\text{dist}}\}
\]

\[
\text{max\_distance} := s_{\text{dist}} + 10.
\]

The rationale of this calculation is similar to that of srtt calculation for TCP-PR. If distance soars intermittently, s_dist will be reduced to 90% of the previous (long) s_dist value every time when 10 new distance values are measured. Clearly, the severer the reordering, the longer the s_dist and max_distance. If the difference of the highest ACK and the oldest undecoded sequence number is greater than \(4 \times \text{max\_distance}\), retransmission is performed when one of the following conditions is met. If the ACKs with the same oldest undecoded sequence number are received max_distance times, the oldest undecoded packet is retransmitted. Otherwise, the sender retransmits one from the packets in the coding buffer whose sequence numbers are smaller than the difference between the highest acknowledgement and max_distance, and have not been acknowledged nor retransmitted. In order to prevent too frequent retransmission, we empirically set max_distance to s_dist + 10 and the threshold triggering retransmission to \(4 \times \text{max\_distance}\), though it should be dependent on network conditions. This is an interesting issue for further study.

The algorithm for the source side and the receiver side is summarized as follows. What is different from the original TCP/NC is italicized.

**Source side:**

1. Packet arrives from TCP sender:

   a. If packet is not already in the coding window, add it to the coding window.

   b. Repeat the following:

      i. Generate a random linear combination of the packets in the coding window.

      ii. Add the network coding header specifying the set of packets in the coding window and the coefficients used for the random linear combination.

      iii. Deliver the packet to the IP layer.

2. ACK arrives from receiver:

   Remove the ACKed packet from the coding buffer and hand over the ACK to the TCP sender.

   a. If the difference of the highest ACK and the oldest undecoded sequence number is greater than 4 times max_distance, retransmission is performed if one of the following cases holds.

      i. If ACKs with the same oldest undecoded sequence number are received max_distance times, the oldest undecoded packet is retransmitted.
(ii) Otherwise, retransmit a packet from the coding buffer whose sequence number is smaller than the difference between the highest acknowledgment and max_distance and has not been acknowledged nor retransmitted.

Receiver side:

(1) ACK arrives from TCP sink:
   If the ACK is a control packet for connection management, deliver it to the IP layer and return to the wait state; else, ignore the ACK.

(2) Packet arrives from source side:
   (a) Remove the network coding header and retrieve the coding vector.
   (b) Add the coding vector as a new row to the existing coding coefficient matrix, and perform Gaussian elimination to update the set of seen packets.
   (c) Add the payload to the decoding buffer. Perform the operations corresponding to the Gaussian elimination, on the buffer contents. If any packet gets decoded in the process, deliver it to the TCP sink and remove it from the buffer.
   (d) Generate a new TCP ACK with sequence number equal to that of a newly seen packet.

5. Performance Evaluation

5.1. Simulation Setup. We simulated the original TCP/NC and our new network coding scheme using ns-2. We used TCP-Reno and TCP-PR on the original TCP/NC and our new network coding layer, respectively. The coefficients for linear combinations were chosen from GF(2^8). To simulate persistent packet reordering in a multihop wireless network, we transmitted packets across two paths between the source node and the sink node. The physical and MAC layer protocols are standard IEEE802.11 with RTS/CTS disabled. The radio propagation model is two-ray ground reflection model. Each link has a bandwidth of 11 Mbps and a queue of 50 packets. For routing, we used AOMDV which was modified so that the sender transmits packets over multiple paths. TCP packets were distributed equally over the two paths for all the simulation runs except the ones to test the effect of traffic distribution ratio. The source alternates between the two paths every time it sends a fixed number of packets according to the distribution ratio.

The topology is shown in Figure 2. Between the TCP source node and the TCP sink node there are a short path and a long path in terms of hop counts. TCP packets with length of 1000 bytes were emanated from the source using FTP for about 100 seconds. When we put more intermediate nodes in the long path for the purpose of making RTT difference between two paths, the resulting packet reordering was not significant. This implies that adding more hops on one of multiple paths in a multihop wireless network does not necessarily increase RTT enough to cause significant packet reordering at the TCP receiver. However, when adding a background UDP flow passing through some links along the long path, significant packet reordering was generated. The background traffic was varied from 0.5 Mbps to 1.0 Mbps. The more the background traffic, the more severe the RTT imbalance between the two paths. To illustrate the effect of background traffic, Figure 3 shows the time series of the sequence numbers of received TCP segments and the congestion window when a background flow of 1.0 Mbps passes the long path and network coding is not applied. We can see that out-of-order packets caused the congestion window to be cut down several times. Out of 8 congestion window reductions shown in the plot, only two reductions were caused by packet losses and the other reductions by out-of-order packets.

The receiver window size of TCP was set to 40 packets. We set the $\beta$ value for TCP-PR to 1.2, 1.5, 1.7, and 2.0. As described in [8], the $\beta$ for wired networks is set to a larger value, like 3. The larger the $\beta$, the longer the TCP source waits before determining a packet loss. However, in wireless networks with low bandwidth, even slightly excessive traffic can have a negative impact on TCP performance. This is why we set $\beta$ to a conservative value.

For the network coding layer, the redundancy parameter $R$ was set to 1.0, 1.1, and 1.2. The coding window size is varied from 2 to 5. In our simulation we set the loss rates of links to 0 to focus on packet reordering.

5.2. Simulation Results

5.2.1. Effect of Background Traffic. Figure 4 shows the TCP goodput versus background traffic for the redundancy factor of 1.0. Each point in the plot is averaged over 10 iterations with the session start time randomized. We varied the background traffic from 0.5 Mbps to 1.0 Mbps. As expected, the goodput for TCP-Reno without network coding is significantly reduced as background traffic increases. For TCP-Reno with the original TCP/NC, the goodput is improved, but not significantly. Although the goodput with the maximum coding window $W$ 5 is better than that with $W$ 2, the difference is not great. In contrast, for TCP-PR, with our network coding, much higher goodput is achieved. The reason why TCP-Reno with the original TCP/NC does not significantly outperform TCP-Reno without network coding is that the network coding layer of the original TCP/NC cannot mask out-of-order packets from TCP in the presence of severe packet reordering. Figure 5 shows the change of the congestion window for TCP-Reno with the original network coding. For about 5 seconds, the congestion window was cut down 9 times. In contrast, TCP-PR does not suffer congestion window reductions during the same time period. Note that even though the congestion window keeps increasing, the effective window size is bounded by the receiver's advertised window, which is set to 40 packets in our simulation.
5.2.2. Effect of Maximum Coding Window and Redundancy Factor. Figure 6 shows the TCP goodput versus the coding window size \( W \) for 1.0 Mbps background traffic. The minimum value of \( W \) for nontrivial linear combination is 2. While a large value of \( W \) makes the network coding layer more robust to packet reordering, it affects the coding header size and encoding/decoding complexity. Therefore the maximum coding window size \( W \) was set to 5. Although TCP-Reno with the original TCP/NC gives higher goodput for a higher value of \( W \), the improvement was not as great as expected. In case of TCP-PR, with our NC, the goodput is not affected by \( W \) because TCP-PR is robust to packet reordering and our NC can selectively retransmit. Also this plot shows that the effect of redundancy parameter \( R \) is not significant for both schemes. From this we can infer that redundant random linear combinations may not help in wireless networks with limited bandwidth as much as in wired networks with high bandwidth. With \( W \) set to 2, the performance gap between the two schemes widened. This is because the smaller \( W \), the harder for the original TCP/NC to mask out-of-order packets from the TCP layer. Once packet reordering is exposed to the TCP layer, TCP-PR is to outperform TCP-Reno. That is why we saw the performance gap widened.

5.2.3. Effect of Traffic Distribution Ratio. We performed simulations varying the traffic portion for the short path of the two paths. The portion varies from 10% to 90%. The background UDP traffic passes through the long path. For this simulation, the background traffic is set to 1.0 Mbps and \( R \) and \( W \) are set to 1.0 and 2, respectively. Figure 7 shows the TCP goodput versus the traffic portion. For all range of portion, TCP-PR with our network coding outperforms TCP-Reno with TCP/NC and TCP-Reno without network coding. TCP-Reno with TCP/NC and TCP-Reno without network coding shows best goodput for the traffic portion of 90%. We can see that TCP-Reno with
TCP/NC can mask packet reordering from TCP for a small number of out-of-order linear combinations. In contrast, TCP-PR with our network coding scheme is not greatly affected by the number of out-of-order linear combinations.

6. Conclusions

In this paper, we proposed a new network coding scheme that is more robust to packet reordering in multihop wireless networks. In a packet distribution system over multiple paths, severe packet ordering may persist if a fair amount of background traffic passes through one of multiple paths. In such a situation, the original TCP/NC approach combined with the existing standard TCP may not suffice to overcome severe packet reordering. Unlike the original TCP/NC scheme, our network coding layer acknowledges a degree of freedom with the sequence number of a newly seen packet. Also for the purpose of reducing the size of the decoding matrix and the coding buffer, retransmission at the coding layer is allowed. Simulation results show that our network coding scheme outperforms the existing TCP/NC scheme under the conditions of severe packet reordering.

As future work, we will examine other factors including loss rates and diverse topologies.

Conflict of Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (Grant no. 2009-0069162).

References

[1] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, “A survey on wireless multimedia sensor networks,” Computer Networks, vol. 51, no. 4, pp. 921–960, 2007.

[2] S. Tozlu, M. Senel, W. Mao, and A. Keshavarzian, “Wi-Fi enabled sensors for internet of things: a practical approach,” IEEE Communications Magazine, vol. 50, no. 6, pp. 134–143, 2012.

[3] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, “XOR in the air: practical wireless network coding,” in Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM ’06), pp. 243–254, ACM, Pisa, Italy, September 2006.

[4] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, “Trading structure for randomness in wireless opportunistic routing,” in Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 169–180, August 2007.

[5] S. Biaz and N. H. Vaidya, “De-randomizing congestion losses to improve TCP performance over wired-wireless networks,” IEEE/ACM Transactions on Networking, vol. 13, no. 3, pp. 596–608, 2005.

[6] C. Lai, K.-C. Leung, and V. O. K. Li, “Enhancing wireless TCP: a serialized-timer approach,” in Proceedings of the IEEE INFOCOM, pp. 1–5, IEEE, San Diego, Calif, USA, March 2010.

[7] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, “Network coding meets TCP: theory and implementation,” Proceedings of the IEEE, vol. 99, no. 3, pp. 490–512, 2011.
[8] S. Bohacek, J. Hespanha, J. Lee, C. Lim, and K. Obraczka, “A new TCP for persistent packet reordering,” IEEE/ACM Transactions on Networking, vol. 14, no. 2, pp. 369–382, 2006.

[9] C. Lochert, B. Scheuermann, and M. Mauve, “A survey on congestion control for mobile ad hoc networks,” Wireless Communications and Mobile Computing, vol. 7, no. 5, pp. 655–676, 2007.

[10] S. Gheorghiu, A. L. Toledo, and P. Rodriguez, “Multipath TCP with network coding for wireless mesh networks,” in Proceedings of the IEEE International Conference on Communications (ICC’10), IEEE, May 2010.

[11] M. Kim, J. Cloud, A. Parandehgheibi et al., “Congestion control for coded transport layers,” in Proceedings of the 1st IEEE International Conference on Communications (ICC ’14), pp. 1228–1234, June 2014.

[12] MPTCP, https://datatracker.ietf.org/wg/mptcp/documents/.

[13] M. Li, A. Lukyanenko, and Y. Cui, “Network coding based multipath TCP” in Proceedings of the IEEE Conference on Computer Communications Workshops, pp. 25–30, March 2012.

[14] J. Cloud, F. du Pin Calmon, W. Zeng, G. Pau, L. M. Zeger, and M. Medard, “Multi-path TCP with network coding for mobile devices in heterogeneous network,” in Proceedings of the IEEE Vehicular Technology Conference (VTC Fall ’13), September 2013.

[15] C.-C. Chen, G. Tahasildar, Y.-T. Yu, J.-S. Park, M. Gerla, and M. Y. Sanadidi, “CodeMP: network coded multipath to support TCP in disruptive MANETs,” in Proceedings of the 9th IEEE International Conference on Mobile Ad-Hoc and Sensor Systems (MASS ’12), pp. 209–217, October 2012.

[16] P.-L. Ageneau, N. Boukhatem, and M. Gerla, “Fairness evaluation of pipeline coded and non coded TCP flows,” in Proceedings of the 1st IEEE International Conference on Communications (ICC ’14), pp. 2754–2760, June 2014.
Hindawi
Submit your manuscripts at
http://www.hindawi.com