# Network Coding with Wait Time Insertion and Configuration for TCP Communication in Wireless Multi-hop Networks

Eiji Takimoto, Shuhei Aketa, Shoichi Saito, and Koichi Mouri

***Abstract*—In TCP communication in wireless multi-hop networks, traffic and inference increases relative to the number of acknowledgment (ACK) packets. PiggyCode, a previously proposed approach, focuses on the bidirectionality of DATA and ACK packets, and reduces the number of transmissions by encoding DATA and ACK packets. However, the effect of PiggyCode depends on the transmission rate. We propose a method to enhance the effect of network coding techniques regardless of transmission rate by inserting wait times into packet relay processes. Simulation results demonstrate the effectiveness of the proposed method. In addition, we propose a method to dynamically adjust wait times relative to hop count and traffic.**

***Index Terms*—TCP, network coding, wireless multihop networks**

## I. Introduction

WITH the evolution and popularity of wireless devices, wireless mesh networks comprising multiple wireless nodes with router functions have attracted increasing attention. Internet access is a typical wireless mesh network service. Most Internet communication uses the transmission control protocol (TCP) as the transport layer protocol because it is highly reliably. TCP uses an acknowledgment (ACK) packet to achieve reliable communication. However, mechanisms based on automatic repeat requests generate DATA-ACK interference, i.e., inter-flow interference.

PiggyCode[1] is the first approach to propose countermeasures against the inter-flow interference problem. PiggyCode focuses on the bidirectionality of DATA packets and ACK packets, and uses network coding (NC)[2] to reduce the number of intermediate node transmissions. An NC module located in the MAC layer, operates on packets in the wait state in a MAC layer interface queue. Therefore, PiggyCode is effective only when the network is close to congestion. Over an entire network, the number of packet transmissions should be reduced regardless of the amount of traffic, i.e., the effect of NC should be constant.

Thus, we propose two methods to improve the efficiency of PiggyCode. The first method is delay insertion in the relay and coding process. The second method is delay time adjustment that considers the communication environment.

Manuscript received January 8th, 2017; revised February 8th, 2017.

E. Takimoto, S. Aketa, K. Mouri are with College of Information Science and Engineering, Ritsumeikan University, 1-1-1 Noji-higashi Kusatsu Shiga 525-8577 Japan e-mail: takimoto,mouri@cs.ritsumei.ac.jp, saketa@asl.cs.ritsumei.ac.jp.

S. Saito is with Nagoya Institute of Technology, Gokiso Showa Nagoya 466-8555 Japan.

## II. PiggyCode

### A. Overview

PiggyCode is an NC-based scheme designed to enhance TCP communication. NC improves transmission efficiency by encoding packets in a transmission node and relay nodes, and by decoding in a receiving node. As stated previously, PiggyCode focuses on the bidirectionality of DATA packets and ACK packets, and reduces the number of intermediate node transmissions by NC. Furthermore, improvement of throughput and round trip time (RTT) are improved due to the deduced number of transmissions.

PiggyCode adds an NC layer, which provides encoding and decoding functionalities, between the network layer and the MAC layer. The NC layer consists of two modules and a buffer:

- encoding module
- decoding module
- decoding buffer.

The NC layer performs different operations for each network node role. The NC layer of a TCP sender node copies a DATA packet and stores the copied packet in the decoding buffer. The copied packet will be used to decode. For intermediate nodes, the NC layer in the intermediate node encodes a DATA packet and an ACK packet and then transmits the encoded packet. The encoding process runs when a packet is added to the interface queue. If a packet that can be encoded by the enqueuing packet exists, the coding module encodes it by XORing and attaches a PiggyCode header, which indicates that the packet is encoded, to the encoded packet. Then, the encoding module adds the encoded packet to the head of the interface queue. When there are multiple candidates to encode in the interface queue, the packet closest to head of the queue is selected. Thus, RTT is decreased. The receiver side NC layer uses the TCP packet stored in the decoding buffer to decode the receiving encoded packet.

### B. PiggyCode Limitations

PiggyCode considers TCP packets in an interface queue as encoding objects. Therefore, PiggyCode has two limitations.

When the transmission rate is low, the probability of TCP packets existing in an intermediate node interface queue is low. Therefore, an intermediate node has no opportunity to encode.

When the transmission rate is high, nodes cannot transmit packets immediately due to network congestion. Moreover, sender and receiver nodes cannot always transmit DATA and ACK packets, respectively. Thus, packets waiting in an interface queue tend to incline DATA or ACK packets. Therefore, efficient encoding is impossible.

Consequently, PiggyCode cannot maximize the effect of NC consistently. To improvement PiggyCode, DATA and ACK packet encoding should not be influenced by the transmission rate. Thus, we propose a method that inserts wait time for packet forwarding. Furthermore, we also propose a method to adjust the wait time based on hop count and the length of the interface queue to suppress increased RTT caused by wait time insertion.

## III. Improvement of Coding Ratio by Wait Time Insertion

To address the problems described in the previous section, the proposed method inserts a wait time to increase the coding ratio. When the NC layer in an intermediate node intends to enqueue a TCP packet to the interface queue, the NC layer confirms whether a packet that is a candidate to encode with the TCP packet exists. If not, the NC layer inserts wait time to the TCP packet to increase the likelihood of encoding. Thus, PiggyCode can encode more packets regardless of the transmission ratio. In the following, we describe the proposed methods and simulations.

### A. *Proposed Method*

The proposed method uses an encoding queue and an interface queue. When the NC layer receives a DATA packet from the IP layer, the NC layer searches the interface queue for an ACK packet. If an ACK packet is found, the NC layer encodes the DATA packet and the ACK packet and then inserts the encoded packet at the top of the interface queue. If an ACK packet is not found, the NC layer enqueues the DATA packet in the encoding queue rather than the interface queue. The DATA packet waits for the predefined time in the encoding queue. If the NC layer receives an ACK packet from the IP layer before the predefined time expires, the DATA packet is dequeued from the encoding queue to be encoded with the ACK packet. After encoding, the encoded packet is inserted into the head of the interface queue. If the predefined time expires, the DATA packet is dequeued from the encoding queue and is enqueued into the interface queue. When the NC layer receives an ACK packet, the NC layer first searches the encoding queue for a DATA packet to encode. If a DATA packet is found, the NC layer encodes both packets and inserts the encoded packets in the head of the interface queue. Otherwise, the NC layer searches for a DATA packet in the interface queue. After the search, the NC layer process is the same as above.

The proposed method targets only DATA packets because if both DATA and ACK packets are targeted, the delay caused by the wait time increases and ACK clocking is hindered.

### B. *Implementation using ns-2*

We implemented the proposed method using ns-2[3], to verify the effectiveness of the proposed method. Fig. 1 shows the module structure of the implementation. The NC module consists of an encoding block, a decoding block and a decoding buffer that temporarily stores packets to decode an encoded packet. The encoding block has a queue in which DATA packets wait the predefined time. Likewise, the

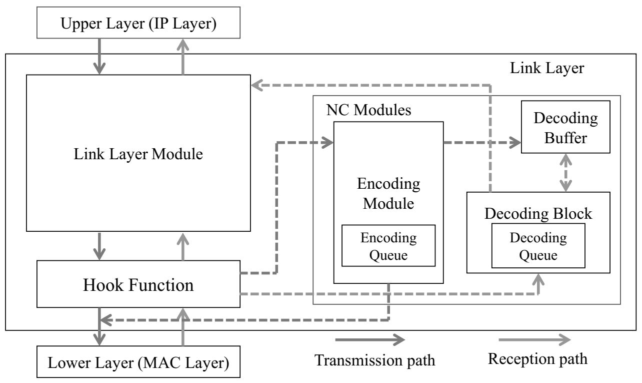

Fig. 1. Node Architecture

decoding block has a queue in which encoded packets wait to be decoded.

Each node performs in promiscuous mode because the encoded packets must be received by both an up-link side node and a down-link side node. When a node intends to send a TCP packet, the hook function always hooks the TCP packet and passes it to the encoding block. The encoding block processes each packet type differently.

If the TCP packet is a control packet, such as TCP-SYN, the encoding module does nothing. The module returns the packet to the original transmission processing flow. Otherwise, the encoding module copies the TCP packet and stores the copied packet in the decoding buffer for subsequent decoding. After buffering the copied packet, the encoding module of the TCP sender or the destination nodes does nothing.

On the other hand, the encoding module of the intermediate node responsible for NC processing has many things to do. The encoding block confirms whether the TCP packet is a DATA packet. The encoding block adds DATA packets to the encoding queue and sets the wait timer. When an ACK packet is transmitted before the waiting timer expires, the encoding block dequeues a DATA packet and cancels the wait timer of the dequeued packet. The dequeued packet is encoded using the ACK packet. The encoding block adds the coding header used for decoding to the encoded packet. Then, the block passes the encoded packet to the lower layer. When the wait time expires, the encoding block dequeues the DATA packet related to the wait time from the encoding queue and passes the packet to the lower layer. If the encoding queue is full when it receives a DATA packet from the upper layer, the encoding block forcibly dequeues a packet from the head of the encoding queue and adds the new DATA packet.

If an ACK packet is sent, the encoding block dequeues a DATA packet from the head of the encoding queue. The ACK packet is encoded using the DATA packet and a coding header is added to the packet. Then, the encoding block passes the encoded packet to the lower layer. If there is no DATA packet in the encoding queue, the encoding block passes the ACK packet to the lower layer as is.

The hook function passes a receiving packet from the lower layer to the decoding block. The decoding block confirms whether the packet is encoded. If the packet is encoded, the decoding block acquires the packet specified in the coding header from the decoding buffer. Then, the decoding block decodes the encoded packet using the specified

TABLE I
SIMULATION PARAMETERS

| Parameter | Value |
|---|---|
| Simulator | ns-2(version 2.34) |
| Propagation Model | two-ray model |
| Simulation Time | 400s |
| Distance among nodes | 200m |
| Propagation range | 250m |
| Data Rate | 2Mbps |
| Basic Rate | 1Mbps |
| Application | CBR (200 – 1000Kbps) |
| Packet Size | 1024bytes |
| TCP variant | TCP/NewReno |
| RTS/CTS | off |

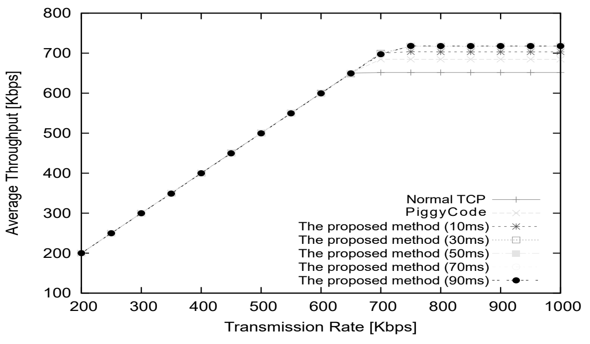

Fig. 2. Throughput in two-hop topology.

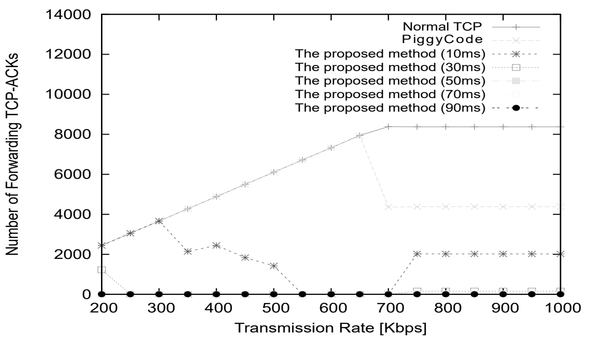

Fig. 3. Number of ACKs transmitted by intermediate nodes in two-hop topology.

packet. The original decoded packet is copied and passed to the upper layer. The copied packet is stored in the decoding buffer because this is the first time that the NC layer has handled the packet. If the specified packet is not in the decoding buffer, the encoded packet is added in the decoding queue to wait for the specified packet. The decoding block periodically checks for the arrival of the specified packet.

## C. Simulation Evaluations

*1) Simulation Environment:* We evaluated the effectiveness of the proposed method by simulation. We used two, three, and four-hop chain network topologies. In the simulations, we measured throughput, RTT, and the number of ACK packet transmissions. The targets for comparison were normal TCP (NewReno) and PiggyCode. The number of ACK packet transmissions indicates how many ACK packets were not encoded out of all ACK packets transmitted by the destination node. We evaluated the network topologies with different hop counts and wait times. Other parameters are shown in TABLE I.

*2) Simulation Results:*

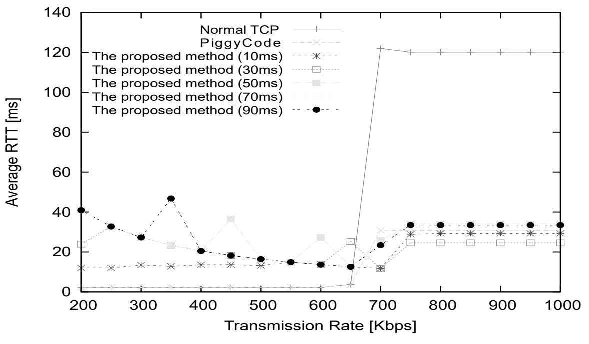

Fig. 4. RTT in two-hop topology.

*a) Two-hop topology:* Fig. 2, Fig. 3, and Fig. 4 show the average receiving throughput, the number of ACK packets transmitted by intermediate nodes, and RTT, respectively. The throughput of the proposed method is greater than normal TCP and PiggyCode. PiggyCode improved throughput by 5% compared to the normal TCP. The proposed method improved throughput by 10% compared to the normal TCP. Relative to wait time, longer wait times improve throughput because the number of packet transmissions is suppressed As a result, an encoding block with a long wait time can encode more packets.

The number of PiggyCode ACK packet transmissions decreased by approximately 50% when the transmission rate was greater than 700Kbps. However, the number of PiggyCode ACK packet transmissions was the same as the normal TCP when the transmission rate was less than 700Kbps. Thus, the effectiveness of PiggyCode only emerges under congestion conditions. In contrast, the proposed method further decreased the number of ACK packet transmissions.

The limitation of PiggyCode is the timing of DATA and ACK packets. The TCP sender node sends multiple DATA packets in succession according to its congestion window size. Since the TCP receiver node successively receives DATA packets, the TCP receiver node also creates successive transmission of multiple ACK packets. Therefore, the intermediate node cannot encode frequently due to the biased reception. On the other hand, the wait time of the proposed method solves this deviation. Thereby, the proposed method achieved to increase the number of the encoded packets, particularly for long wait times.

The RTTs of the normal TCP and PiggyCode are approximately 4ms at a low transmission rate. On the other hand, the RTT of the proposed method is longer according to the length of the wait time. When the transmission rate is greater than 700kbps, the RTT of the normal TCP increases significantly due to congestion. PiggyCode and the proposed method suppressed the increase of RTT to less than 40ms. The suppression effect comes from NC. Thus, the time DATA packets spend waiting in the encoding queue is short when the transmission rate is high. Therefore, RTT is not influenced by wait time. In particular, the simulation results showed the best when the wait time was 30ms.

*b) Three-hop topology:* Fig. 5, Fig. 6, and Fig. 7 shows the average receiving throughput, the number of ACK packet transmissions of intermediation nodes, and RTT, respectively. All results decreased relative to increased hop count. In particular, the throughput of PiggyCode was reduced by

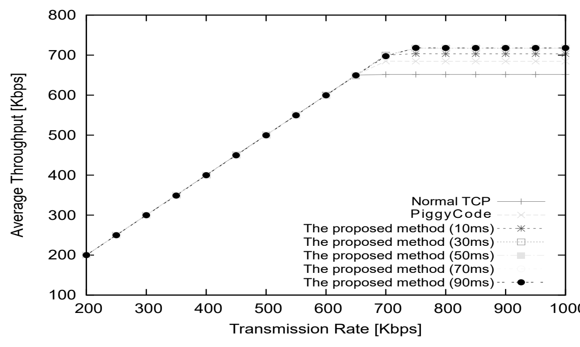

Fig. 5. Throughput in three-hop topology.

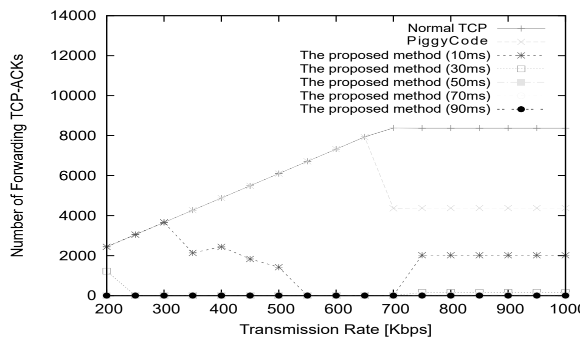

Fig. 6. Number of ACKs transmitted by intermediate nodes in three-hop topology.

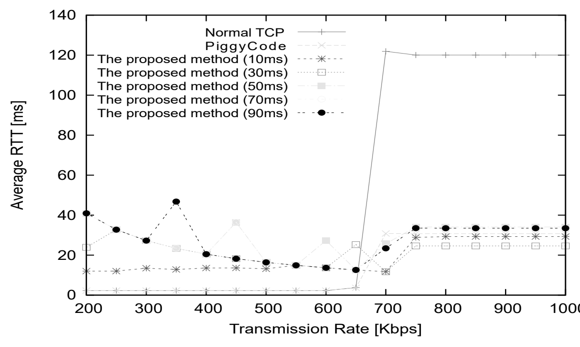

Fig. 7. RTT in three-hop topology.

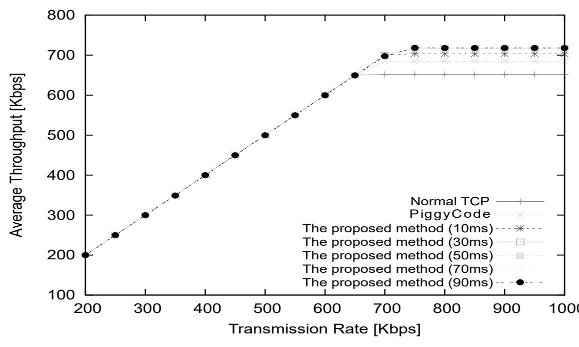

Fig. 8. Throughput in four-hop topology.

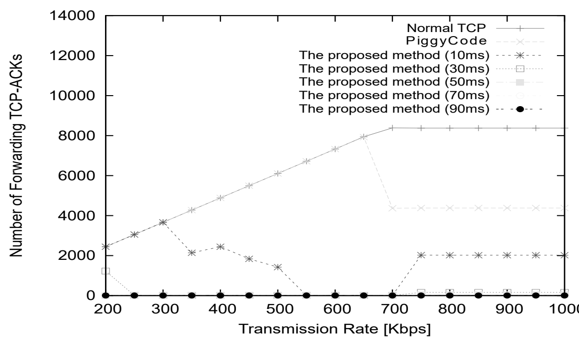

Fig. 9. Number of ACKs transmitted by intermediate nodes in four-hop topology.

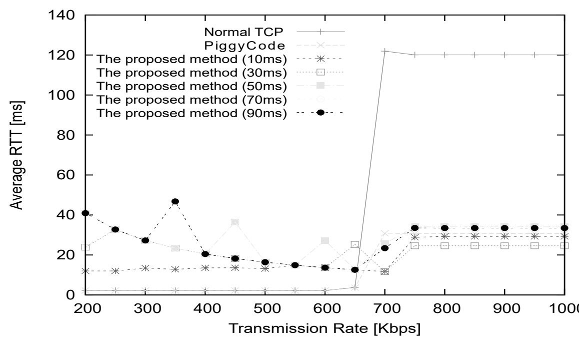

Fig. 10. RTT in four-hop topology.

7% compared to the normal TCP. In contrast, the proposed method improved throughput by 5% compared to the normal TCP. These results can be attributed to packets losses caused by the hidden terminal problem and increased inter-flow interference. The impact of packet losses are significant when a lost packet is an encoded packet. The dissipation of an encoded packet equals two packet losses, i.e., a DATA packet loss and an ACK packet loss. Therefore, the effect of packet loss for PiggyCode and the proposed method is more significant. Furthermore, when an intermediate node sends an encoded packet, the transmission mode is unicast and the destination MAC address in the MAC header of the encoded packet is the MAC address of the next hop node of the DATA packet included in the encoded packet. The node nearest to the TCP receiver node receives the encoded packet in a unicast manner. Thus, when the encoded packet dissipates under the MAC layer, the MAC layer retransmits the packet according to the IEEE802.11MAC standard[4]. Since the opposite side node is not designated in the MAC header, the node receives the encoded packet using promiscuous mode. When the opposite side node can not receive the encoded packet, the packet is never retransmitted. Therefore, packet losses increase with PiggyCode and the proposed method, both of which are based on NC techniques. PiggyCode decreased throughput significantly. Even though the proposed method is based on PiggyCode, it was not influenced by packet losses. The proposed method reduces the influence of packet losses by decreasing of the number of packet transmissions by inserting wait times. This can also be confirmed by the performance improvements obtained with longer wait times.

Thus, the RTTs of PiggyCode at a high transmission rate are the same as those of the normal TCP. In contrast, the RTTs of the proposed method were low. However, the RTTs of the proposed method were worse at a low transmission rate because of the wait time insertion. When the wait time was 90 ms, the RTTs of the proposed method were the shortest. The number of ACK transmissions was the same as the results obtained in the two-hop simulation.

*c) Four-hop topology:* Fig. 8, Fig. 9, and Fig. 10 show the average receiving throughput, the number of ACK packet transmissions of intermediation nodes, and RTT, respectively. All results worsened relative to increased hop count. Among the three methods, PiggyCode demonstrated the highest throughput. The improvement achieved by the proposed method was minimal, due to the frequent occurrence of packet losses caused by inter-flow interference and the hidden terminal problem. As hop count increases, the phenomenon becomes more remarkable.

### *D. Discussion*

The proposed method improved throughput up to a maximum of 10% with three hops. However, with four hops, the PiggyCode throughput was the best. The proposed method efficiently encoded a DATA packet and an ACK packet with the wait time insertion. Thus, the proposed method increased usable bandwidth and improved throughput. When the hop count is less than three, the proposed method with the long wait time was more efficient.

The RTTs of the proposed method increased with a low transmission rate due to the insertion of wait times. However, the RTTs of the proposed method with a high transmission rate decreased significantly because the proposed method encodes many TCP packets and the encoded packets are transmitted immediately. Thus, wait times should be short to suppress the impact on RTTs for low transmission rates and should be long to reduce RTTs for high transmission rates. The number of ACK transmissions is improved significantly in all scenarios. The results of the proposed method with the four-hop topology demonstrated a different tendency than the other scenarios, i.e., longer wait times decreased throughput.

The simulation results demonstrate that it is important to configure the length of the wait time relative to hop count and transmission rate in order to improve the efficiency of the proposed method. The proposed method is most efficient with a short-hop topology. Conversely, in the case a long-hop topology, the wait time should be zero, i.e., PiggyCode is suitable. Moreover, a short wait time will be more efficient with a low transmission rate and a long wait time will be more efficient with a high transmission rate.

## IV. Wait Time Configuration Method Based on Hop Count and Queue Length

As mentioned in the previous section, the effect of the proposed method is influenced by hop count and transmission rate. Therefore, appropriate configuration of the wait time will achieve the efficiency of the proposed method. In this section, we described the wait time configuration method and simulation evaluations.

### *A. Configuration Method Details*

The simulation results (Section 3) demonstrate that PiggyCode improved throughput more than the proposed method in the four-hop topology. Based on the results, the wait time was set to zero to achieve the same results as PiggyCode when the hop count of the topology was greater than four. Once a path from a TCP sender node to a TCP receiver node is established, an intermediate node can know the hop count to the TCP sender node and to the TCP receiver node by referring to its forwarding table. Thus, the intermediate node acquires the hop count of the path by adding the hop count to the TCP sender node and the hop count to the TCP receiver

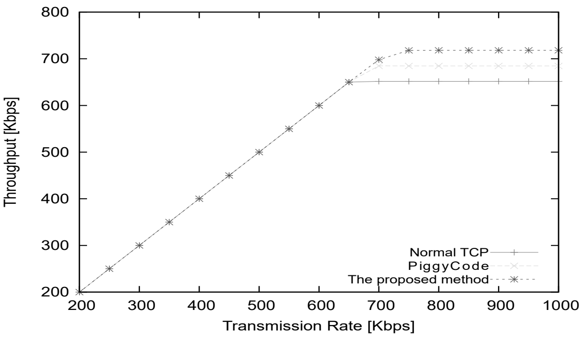

Fig. 11. Throughput in two-hop topology.

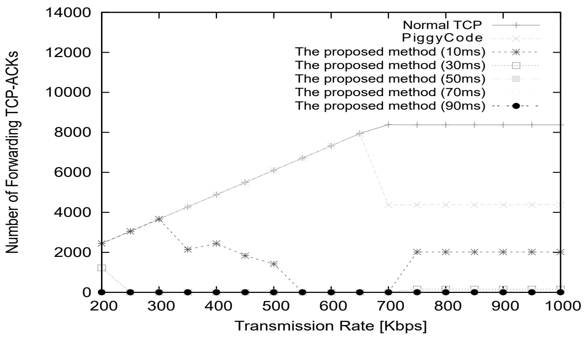

Fig. 12. Number of ACKs transmitted by intermediate nodes in two-hop topology.

node. Thus, the proposed method configures wait time based on the hop count among TCP end nodes.

Under the three-hop topology in which the wait time of the proposed method performs efficiently, the proposed method configures the wait time relative to the transmission rate.

- At a low transmission rate, the proposed method sets a shorter wait time.
- At a high transmission rate, the proposed method sets a longer wait time.

The proposed method determines, whether the transmission rate is high or low based on the length of the interface queue. When the transmission rate is low, the interface queue is nearly empty. In contrast, the interface queue is filled with packets at a high transmission rate. Thus, when the interface queue is almost empty, the proposed method decides as the transmission rate is low. Likewise, when the interface queue is filled with packets, the proposed method decides as the transmission rate is high.

From the simulation results, the appropriate wait time for two or three-hop topologies and the low transmission rate is 10 ms. At high transmission rate, 30 ms is a suitable wait time for the two-hop topology, and 90 ms is suitable wait time for the three-hop topology.

### *B. Simulation Evaluations*

We also evaluated the proposed method, which combines wait time insertion and wait time configuration. This simulation configuration was the same as the previous simulation. Fig 2, Fig 3, and Fig 4 show the two-hop results of the average receiving throughput, the number of ACK packet transmissions of intermediate nodes, and RTTs, respectively. In addition, Fig 2, Fig 3, and Fig 4 show the three-hop average receiving throughput, the number of ACK packet

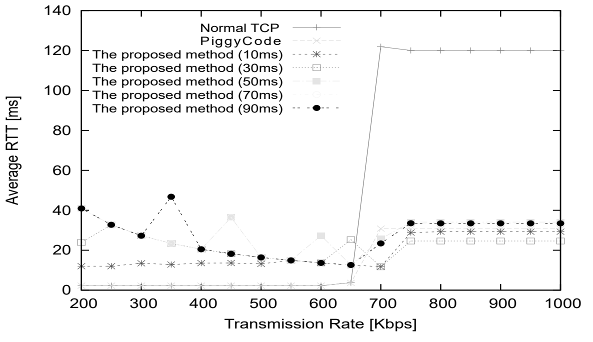

Fig. 13. RTT in three-hop topology.

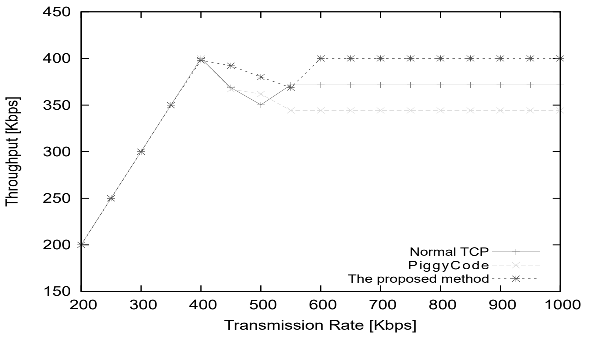

Fig. 14. Throughput in three-hop topology.

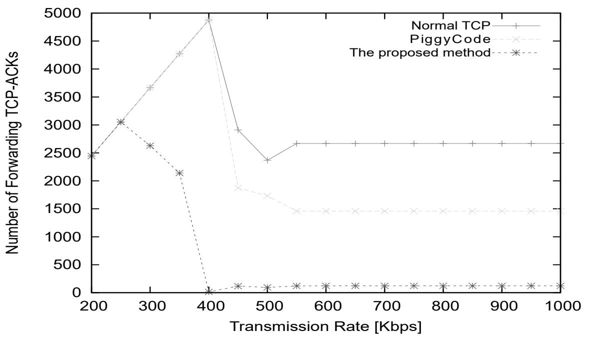

Fig. 15. Number of ACKs transmitted by intermediate nodes in three-hop topology.

transmissions of intermediate nodes, and RTTs, respectively. All results worsened relative to increased hop count.

The maximum throughput improved by the wait time configuration based on the hop count with the three-hop topology. The threshold that the PiggyCode scheme became effective was 700Kbps when the hop count was two. In contrast, the threshold of the proposed method was 300 Kbps. Moreover, the threshold of PiggyCode, when the hop count was three was 400 Kbps. On the other hand, the proposed method started to encode packets when the transmission rate is greater than 250 Kbps. Therefore, the proposed method gained NC efficiency at a lower transmission rate than PiggyCode. In addition, the proposed method encoded more packets at a high transmission rate.

The RTTs with a low transmission rate were suppressed by reducing the wait time. The RTTs with a high transmission rate were less than that of PiggyCode and the normal TCP because the wait time was long.

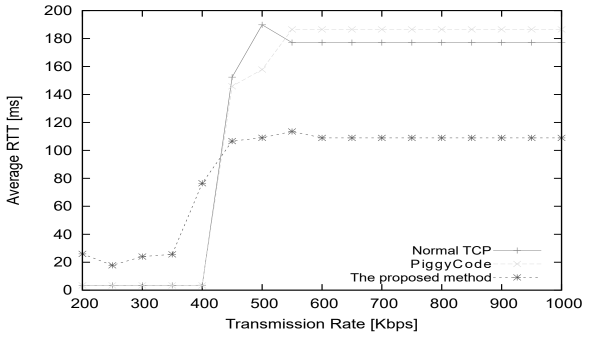

Fig. 16. RTT in three-hop topology.

## C. Discussion

The effectiveness of the proposed method has been confirmed by simulation result. As mentioned previously, PiggyCode has two drawbacks, i.e., it can only perform under congested conditions and the deviation of packet type reduces the probability for encoding. The proposed method addresses these issues by wait time insertion and wait time configuration. At a low transmission rate, the proposed method suppresses increasing RTT by shortening the wait time. At a high transmission rate, the proposed method prolongs the wait time to increase the probability of encoding. As a result, the number of ACK transmissions is reduced and RTT is also suppressed compared to PiggyCode. However, the proposed method reduces throughput when the hop count is greater than three. The wait time configuration based on hop count is a countermeasure against this limitation.

The proposed method does not support the "Delayed ACK" mechanism[5]. Delayed ACK is widely used in TCP implementations. The implementation of Delayed ACK reduces the number of ACK transmissions and the symmetry of DATA and ACK packets, on which PiggyCode and the proposed method rely, collapses. We intend to address the Delayed ACK mechanism in future work.

## V. Conclusion

In this paper, we revealed the problems of PiggyCode and proposed the method to solve them. We also confirmed the efficiency of our proposed method by simulation evaluations. The proposed method inserts the wait time to increase the chance for encoding, moreover, the wait time is configured based on the hop count and transmission rate to pick the best of the both the proposed method and PiggyCode. Simulation evaluations shows that the proposed method achieves the throughput improvement by 10%.

## References

[1] S. Scalia, L. and M. F., Gerla, "Piggycode: A mac layer network coding scheme to improve tcp performance over wireless networks," in *GLOBECOM'07*, 2007, pp. 3672–3677.
[2] R. Ahlswede, N. Cai, S. Y. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theor.*, vol. 46, no. 4, pp. 1204–1216, Sep. 2006.
[3] Network Simulator (NS-2). [Online]. Available: http://www.isi.edu/nsnam/ns/
[4] IEEE Std. 802.11 Wireless LAN Medium Access Control(MAC) and Physical Layer(PHY) specification. [Online]. Available: http://standards.ieee.org/getieee802/download/802
[5] R. Braden, request for Comments 1122: Requirements for Internet Hosts – Communication Layers, IETF. [Online]. Available: https://tools.ietf.org/html/rfc1122