Performance Improvement of TCP over an Ad Hoc Network by Combining of Data and ACK packets

Taichi Yuki† Takayuki Yamamoto† Masashi Sugano†† Masayuki Murata†
Hideo Miyahara† Takaaki Hatauchi†††
†Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan
{t-yuki,tak-ymmt,murata,miyahara}@nal.ics.es.osaka-u.ac.jp
††Faculty of Comprehensive Rehabilitation, Osaka Prefecture College of Nursing
3-7-30, Habikino, Habikino, Osaka 583-8555, Japan
sugano@osaka-hsu.ac.jp
†††Energy and Electrical Systems Company, Fuji Electric Co., Ltd.
1, Fuji, Hino, Tokyo 191-8502, Japan
hatauchi-takaaki@fujielectric.co.jp

Abstract

Since a radio channel is shared among terminals in an ad hoc network, packet collisions are frequent. When transmitting packets using TCP, data and ACK packets are transmitted in opposite directions on the same radio channel. Therefore, frequent collisions are unavoidable, and this seriously degrades throughput. To reduce the likelihood of packet collisions when an intermediate node transmits both data and ACK packets, these two types of packet can be combined and transmitted at the same time to increase the efficiency of radio channel utilization.

In this paper, we propose a new technique to improve TCP performance by combining data and ACK packets. Our proposed technique is applicable to an ad hoc network that uses a table-driven routing method. By means of a simulation using networks with various topologies, we have found that throughput can be improved by up to 60% by applying our proposed technique.

1 Introduction

In recent years, many ways of applying wireless ad hoc networks have been developed. When an ad hoc network provides the same services as a wired network, TCP [1] should be used as the transport layer protocol because it is the de facto standard for wired networks. An ad hoc network is multi-hop network composed of wireless channels, though, and the transmission quality of a wireless channel is less stable than that of a wired channel. Therefore, packet loss occurs frequently in an ad hoc network, and the consequent connection failure can severely degrade TCP performance. Several groups have studied TCP performance over ad hoc networks [2, 3, 4, 5]. Much of this research has focused on TCP performance degradation caused by terminal movement. For example, [2] developed the explicit link failure notification (ELFN) technique. ELFN reduces the effect of the decrease of the TCP window size when a link failure occurs in the middle of a route. In this technique, a node freezes the TCP mechanism when a link failure occurs, thus preventing the TCP from making the window size excessively small, and so the TCP performance is improved. Although [2] focused on node mobility, we have examined the performance degradation caused by short-duration link failure in an ad hoc network where the terminal is stationary. We have applied ELFN and developed a technique that improves the performance of such a network [6].

However, we found that simply avoiding link failure was not sufficient to improve TCP performance. Packet collisions occur because TCP is based on bidirectional communication. When TCP is used as the transport layer protocol, a TCP sender sends data packets and a TCP receiver receiving packets sends ACK packets to a sender for acknowledgement. In ad hoc networks, since nodes cannot distinguish between data packets and ACK packets, collisions often occur over wireless connections. As packets travel over multi-hop links, they often collide. The IEEE 802.11 standard provides for channel reservation based on an RTS/CTS control message, but this causes another problem. Neighboring nodes that can receive radio-wave signals must be silent until the channel is released, especially in a high-density network topology. Many nodes cannot send packets they want to send, and eventually packet losses occur and performance deteriorates [7].

In this paper, we propose a technique to improve TCP performance in an ad hoc network that focuses on the bidirectional characteristic of TCP. In this technique, if a data packet and an ACK packet meet in an intermediate node, they will be collectively transmitted in an opposite direction simultaneously. In this way, packet collisions can be avoided and effective use of a wireless channel achieved.

To evaluate this technique, we applied it to an ad hoc

network which used table-driven routing with fixed terminals. In a fixed ad hoc network, packet losses are caused mainly by packet collisions rather than by node mobility. In this way, we could clearly see the effect of this technique. Flexible Radio Network (FRN) is a commercially available product based on an ad hoc network system, and it is driven by a routing table with fixed nodes [8]. We therefore used FRN to evaluate our proposed technique. Through a simulation using networks with various topologies, we found that throughput could be improved by up to 60%, or at least 10% in a very high load situation, by applying our proposed technique.

This paper is organized as follows. Section 2 describes the FRN system and Section 3 describes our proposed technique. We evaluate the technique in Section 4 and conclude the paper in Section 5.

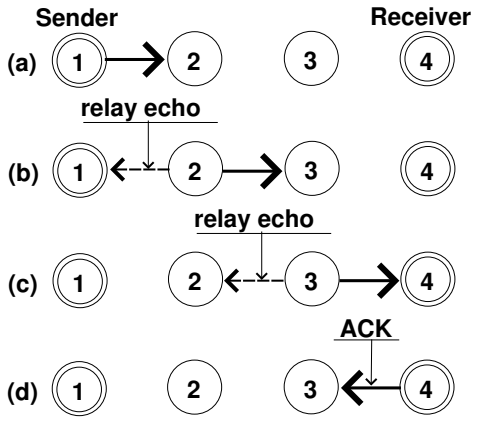


Figure 1: Relay echo mechanism

2 System Description of FRN

FRN is a commercial product based on ad hoc network technology. A network can be built only by suitably placing terminal nodes, and the network can be extended only by adding nodes as needed. FRN has been used, for example, for electric energy control in factories and ticket management systems in skiing areas. Services usually supplied over a wired network are now being requested through an FRN. In such a system, all nodes are controlled by a protocol which can efficiently adapt to node failures and changes to the network configuration.

An FRN's routing system is table-driven, like a DSDV (destination-sequenced distance vector) type [9] with periodic communication. Routing protocols of an on-demand type, such as AODV (ad-hoc on-demand distance vector) [9], are suitable for networks whose nodes move rapidly. However, FRN nodes are basically stationary, they can know their neighboring nodes, and manage through a routing table. Later on, we outline the FRN routing method and the FRN data-link protocol. The details of these are given in [10].

2.1 Data-link Protocol

In an FRN, the radio channel is divided into fixed-length time slots. When a packet is to be sent, the node wanting to send the packet does a carrier sense at the start of the time slot and this carrier sense prevents the packet from colliding with another. In addition, acknowledgement between neighboring nodes is done using a packet which is also used for forwarding from one node to the next. Every neighboring node in a wireless network can receive packets from a node even when it is not the packet source/destination. We call such a packet a *relay echo* in an FRN. The final destination node of a packet does not forward the packet, instead sending a relay echo, and so it sends an FRN ACK packet to the previous node. Forwarding of a packet and sending of an FRN ACK are done in the time slot immediately after the slot in which the node receives the packet. Figure 1 shows the relay echo mechanism.

When the transmission of a packet fails because of a link failure or packet collision, the node resends the packet after waiting for a random number of time slots to prevent another packet collision. Although this random number of time slots is generally within a range of three to five slots, the most desirable number of slots is undetermined. We therefore examined the interval of random time slots, setting it as 3-5 slots (the conventional number) as the shortest interval, 3-9 slots, 3-13 slots, $\cdots$ and 3-25 slots as the longest interval.

The maximum lifetime - the maximum time that a packet is allowed to exist within a network - is defined by slot for all packets and set at the source node. This lifetime is decreased by one for every time slot even if the packet remains in a buffer. When the value reaches zero, the packet is discarded. In the original FRN system, the value of this parameter was defined to be long enough for a network scale. If the value is too small, packets cannot reach their destination; if the value is too big, unneeded packets remain in the network for a long time. Therefore, this value is very important. In this paper, we tentatively set it as 32 slots.

2.2 Routing Protocol

In the FRN, each node manages network information in a *network configuration table*. The network configuration table contains the route information from the node to each destination node. The route information consists of a list of the neighboring nodes' addresses on the routes to a destination node and the hop count of each route. This network configuration table is created through periodic exchanges of control packets that contain information regarding the shortest route.

Every node maintains multiple sets of route information for each destination node, and selects one when sending packets to that node. This selection method is as follows. For each destination, routes are classified into three groups according to their hop count (Figure 2).

- Forward route: The route(s) having the lowest hop

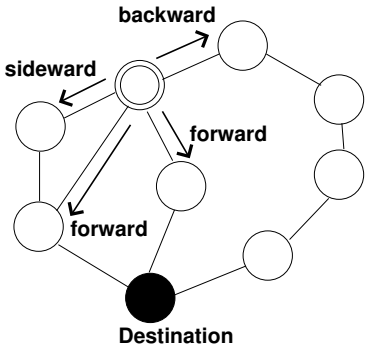


Figure 2: Classification of adjacent nodes

count to the destination.

- Sideward route: The route(s) whose hop count to the destination is equal to the shortest hop count plus one.
- Backward route: The route(s) whose hop count to the destination is equal to the shortest hop count plus two or more.

The transmitting-priority order with respect to which nodes to send to is forward route, sideward route, and backward route. If every transmission to a node on a forward route fails, transmission to a sideward node is attempted. If transmission along all possible sideward routes also fails, the node transmits to a backward node.

3 The Proposed Technique

We explained the problems that arise in ad hoc networks in Section 1. In this paper, we focus on the problem of packet collisions caused by bidirectional TCP communication.

Here we explain our proposed technique to alleviate the problem of packet collisions. ACK packets are very small, containing only TCP header information. Transmitting such an ACK packet using a time slot as big as that used for a data packet wastes radio channel capacity. Therefore, we have considered ways to transmit a combined data and ACK packet by exploiting a characteristic of a wireless channel: that all nodes within the range of an electric wave used to transmit a packet can know the packet contents.

To put it more concretely, every node needs to have two queues. Data packets and ACK packets are saved in their respective queues. When a packet is in both of queue, each destination is determined, and the combined packet is transmitted in a form where an ACK packet is added to a data packet. If each node has only one queue, the combined data and ACK packet can be saved in the queue. However, the combined packet will be erased when one relay echo or ACK (not of TCP but of FRN's datalink layer) arrived in this case, and the function of the relay echo will be destroyed.

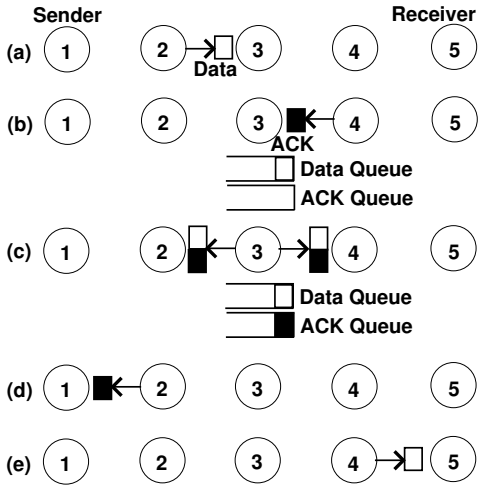


Figure 3: A process where the proposed technique is used

Figure 3 shows a process where our proposed technique is used. If a node does not have packets in each queue, the node behaves as before. If a node has packets in each queue, the node combines a data packet and an ACK packet from the top of each queue and sends the combined packet to two destinations (Figure 3(c)). If a node receives a combined packet, it then determines the two destinations of the combined packet, and when that node is one of the destinations, it receives the portion of the packet addressed to itself. If the node is not one of the destinations, it discards the packet. Here, we must be careful regarding the time slot that the next-hop nodes use to forward the packets. If the nodes forward the packet in the next time slot, as before, packet collision invariably occurs and the node that sent the combined packet cannot receive each relay echo. To avoid this, we set up the time slot so that a packet is less likely to collide with another when nodes receive the combined packet. Figures 3(d) and 3(e) show this. Node 4 received the data-packet portion of the combined packet, so it postpones retransmitting the ACK-packet portion for one time slot to prevent a collision. Node 3 can thus receive each relay echo from nodes 2 and 4.

This technique enables more efficient use of the radio channel and reduces the chance of packets colliding when a node has both data and ACK packets. When TCP is used as the transport layer protocol, there are bidirectional streams in the network, so intermediate nodes often possess both data and ACK packets. Thus, our technique should significantly improve TCP performance.

We evaluated the effect of the delayed ACK option of TCP [1]. The delayed ACK option is aimed at effective use of a wireless channel by sending collected ACK segments. When a destination node receives a data packet, the node delays the return of an ACK packet for a fixed time. All of a node's accumulated ACK packets can be transmitted as one ACK packet if another data packet is receivable within this time. If the number of ACK packets can be lowered by

using this option, improved TCP throughput in an FRN can be realized. Therefore, we next evaluated whether our proposed technique is also effective when used simultaneously with this option.

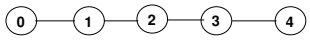
4 Evaluation

4.1 Simulation Environment

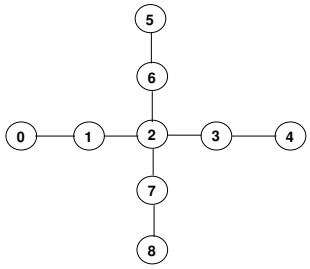
We evaluated our proposed technique through simulations using ns-2 [11] with its radio propagation model extended by the CMU Monarch Project [12]. We used the IEEE 802.11 multicast transmission mode for all packet transmissions with a slight modification to simulate the FRN time slots. In all simulations, the time slots were synchronized at all nodes. This mode is a single-hop multicast that does not produce the channel reservation mechanism that is produced by RTS/CTS of the IEEE 802.11 unicast mode. The radio transmission range was 250 m and each node's buffer capacity was large enough to inhibit buffer overflow in our simulations. Each node exchanged its network configuration table at intervals sufficiently long to not affect the system performance. We set the maximum lifetime as 32 slots in all simulations.

We used the network topologies shown in Figure 4. A circle and a number in the circle mean a node and its address. A line connecting two nodes means that they can communicate directly. Although these topologies are very simple, we can use them to identify basic tendencies of our proposed technique and apply the results to the general FRN network. In all simulations, we used TCP Reno as the transport layer protocol. We also evaluated the case where the delayed ACK option was used. We used the Figure 4(a) topology to evaluate the technique in a pure bidirectional connection. In Figure 4(b), there is a crossing of connections and we evaluated the technique in such a case. Figure 4(c) shows a mesh topology, a more complicated topology with more random connections.

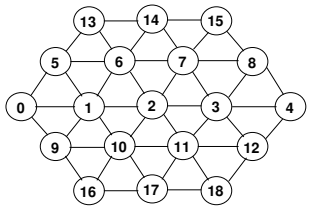
We used throughput as a measure of performance. The throughput was defined as the average number of acknowledged data packets sent from every node per time slot. That is, we measured the total network performance.



(a) Chain Topology



(b) Cross-Chain Topology



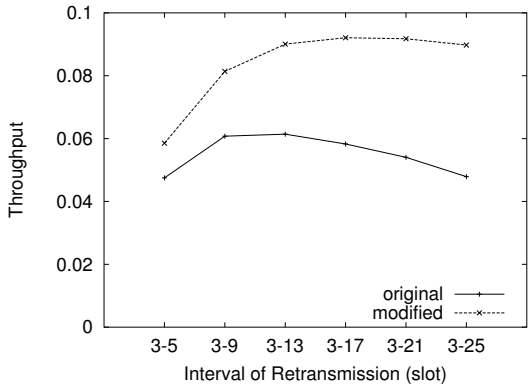
(c) Mesh Topology

Figure 4: Network topologies

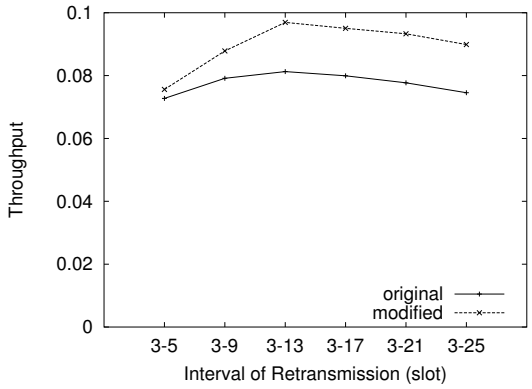
4.2 Results and Discussions

First, we evaluated throughput with the simplest topology (Figure 4(a)). In this network, node 0 was a TCP sender and node 4 was a receiver, so this connection looked like a chain of 4 hops. The results are shown in Figure 5. As mentioned (Section 2), we changed the retransmission interval from 3-5 time slots to 3-25 time slots and measured the throughput at each interval. Figure 5(a) and Figure 5(b) show, respectively, the results without and with the delayed ACK option. A similarity in the two figures is that throughput is low with a short retransmission interval, improves as the interval becomes longer, and eventually deteriorates again at the longest intervals. This is because many packets collide when the retransmission interval is short, lowering throughput, but as the interval becomes longer, packet collisions become less common. However, if the interval is too long, the retransmission timing becomes late, the connection response becomes poor, and the TCP performance deteriorates. Therefore, we must set the interval carefully. In this topology, an interval of 3-17 time slots allowed the best throughput without the delayed ACK option and our proposed technique improved throughput by 60%. With the delayed ACK option, the best throughput was obtained when the interval was 3-13 time slots; in this case, throughput was improved by 20%. The throughput with the delayed ACK option was better than without the option entirely, because the option reduced packet collision in the network. The rate of improvement with the option was lower, though, because packet collision had already been suppressed by the option.

Second, we used the cross-chain topology (Figure 4(b)). In this topology, we observed the effect of collisions when there were two different connections. (The connection between nodes 5 and 8 was added to the connection shown in Figure 4(a).) The throughput shows the added value of throughput of two connections. The results for this topology are shown in Figure 6. The general pattern of the results was similar to that for the chain topology, but the rate of improvement with our proposed technique was only 20% without the delayed ACK option and 15% with the option - slightly lower than for the chain topology. Our proposed technique focuses on packet collisions caused by one

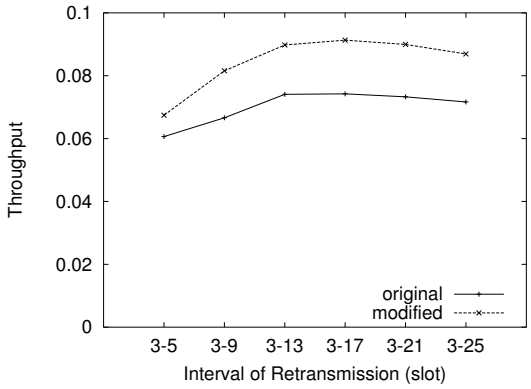


(a) Without the delayed ACK option

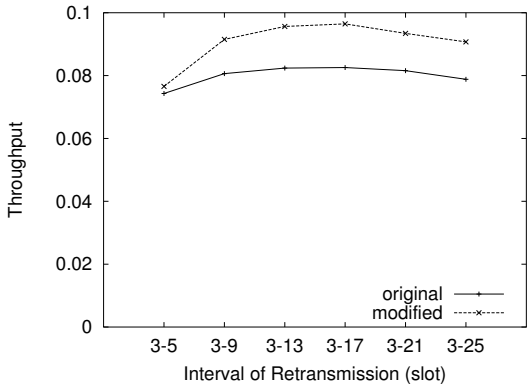


(b) With the delayed ACK option

Figure 5: Throughput of one connection on chain topology



(a) Without the delayed ACK option



(b) With the delayed ACK option

Figure 6: Throughput of crossing two connections on cross-chain topology

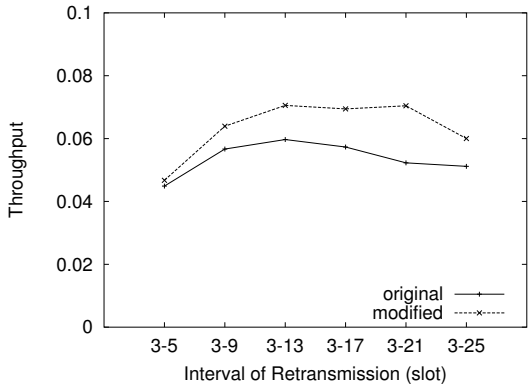
TCP connection, and if there are more than one connection in the network, cross connections occur. Compared with a single-connection case, the degree of improvement thus becomes smaller.

Next, we simulated three or more connections in the mesh network (Figure 4(c)). The three connections were nodes 0 to 4, nodes 13 to 18, and nodes 15 to 16. The results are shown in Figure 7. The pattern again resembled the previous cases with an 18% improvement without the delayed ACK option and a 10% improvement with the option. Last, we simulated random connections with the mesh topology. We set the number of TCP connections (i.e., the network load) to 3, 6, or 9. Random connection meant that two nodes were randomly selected and one became a sender while the other was a receiver. We generated 20 connection patterns and averaged the rate of improvement. Since we had obtained the best throughput using 3-17 time slots in the earlier simulations, we used 3-17 time slots here. The results are shown in Table 1. When there were 9 connections, the network load was very high, but the rate of improvement was at least close to 10%. Thus, even when the load is very high and the connections are random, our proposed technique is effective in an FRN.

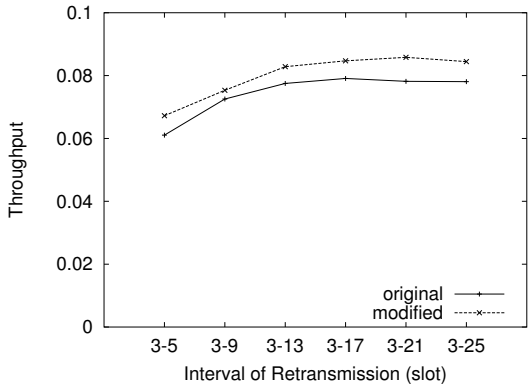
5 Conclusion and Future Work

In this paper, we analyzed the problem of packet collision that arises when TCP is used in an ad hoc networks. We have proposed a technique that combines data and ACK packets, and have shown through simulation that this technique can make radio channel utilization more efficient. In the simulation, the technique improved the TCP performance by up to 60%, and by about 10% even when the network load was very high.

In the future, we will analyze a routing method that prevents packets from colliding. This method will enable data packets and ACK packets to pass along separate paths, thus eliminating the possibility of collision between data packets and ACK packets.



(a) Without the delayed ACK option



(b) With the delayed ACK option

Figure 7: Throughput of three connections intermingled on mesh topology

load	delayed ACK option	
	OFF	ON
3	23.47	21.53
6	15.86	12.96
9	12.38	9.93

Table 1: Average % improvement at the time of setting up a connection at random on mesh topology

Acknowledgements

This work was partly supported by the Special Coordination Funds for promoting Science and Technology of the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

[1] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[2] D. Kim, C. K. Toh, and Y. Choi, "TCP-BuS: Improving TCP Performance in Wireless Ad Hoc Networks," in *Proc. of the ICC 2000*, June 2000.

[3] G. Holland and N. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks Part I: Problem Discussion and Analysis of Results," in *Proc. of MOBICOM '99*, Aug. 1999.

[4] S. Bansal, R. Shorey, S. Chugh, A. Goel, K. Kumar, and A. Misra, "The Capacity of Multi-Hop Wireless Networks with TCP Regulated Traffic," in *Proc. of GLOBECOM '02*, Nov. 2002.

[5] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The Impact of Multihop Wireless Channel on TCP Throughput and Loss," in *Proc. of INFOCOM '03*, Mar. 2003.

[6] T. Yuki, T. Yamamoto, M. Sugano, M. Murata, H. Miyahara, and T. Hatauchi, "A Study on Performance Improvement of TCP over an Ad Hoc Network," *Transactions of IEICE* (in Japanese), vol. J85-B, pp. 2045–2053, Dec. 2002.

[7] S. Ray, J. Carruthers, and D. Staorobinski, "RTS/CTS-Induced Congestion in Ad Hoc Wireless LANs," in *Proc. of WCNC '03*, Mar. 2003.

[8] "Flexible Radio Network, Fuji Electric Co. Ltd." available at `http://www.fujielectric.co.jp/denki/p26/ecop_contents2.html`.

[9] C. E. Perkins, *Ad Hoc Networking*. Addison-Wesley, 2001.

[10] M. Sugano, T. Araki, M. Murata, T. Hatauchi, and Y. Hosooka, "Performance Evaluation of a Wireless Ad Hoc Network: Flexible Radio Network (FRN)," in *Proc. of the ICPWC '00*, Dec. 2000.

[11] "The Network Simulator - ns-2." available at `http://www.isi.edu/nsnam/ns/`.

[12] "The CMU monarch project." available at `http://www.monarch.cs.cmu.edu/`.