Improving Reliability of Real-Time Remote Vehicle Control through Duplicating Control Packets

Woonghee Lee, Joon Yeop Lee, and Hwangnam Kim
School of Electrical Engineering, Korea University, Seoul, Rep. of Korea
Email: {tgorevenge, charon7, hnkim}@korea.ac.kr

***Abstract*—Over the past decade, techniques for remote control vehicles (RCVs) have progressed a lot, and RCVs are receiving huge attention in various fields. Since RCVs, such as drones, have high mobility, it is important for operators to control RCVs reliably. Thus, to keep monitoring and controlling RCVs, operators continuously exchange control packets with RCVs by wireless. However, control packets can be delayed or lost due to any communication or network problems, which may cause collisions or crashes. To overcome this limitation, in this paper, we propose *DupCon*, a new control packet transmission scheme. Considering characteristics of packets, *DupCon* duplicates only control packets, not data packets, and transmits the originals and their duplicates through different paths, respectively. We designed and implemented *DupCon* on real devices. Moreover, we conduct an analysis on *DupCon* to demonstrate the validity of *DupCon* theoretically. Furthermore, we conducted various experiments, and the results show that *DupCon* improves reliability and stability of real-time remote vehicle control.**

***Index Terms*—Remote Control Vehicle, Drone, Unmanned Aerial Vehicle, Control packet, Multipath TCP**

I. Introduction

Supported by advancements of various technologies, many researchers have become interested in remote control vehicles (RCVs), such as drones. In addition, many service providers have tried to utilize RCVs in various fields. RCVs are able to move and perform diverse operations to accomplish missions given by operators. During missions, RCVs continuously exchange control packets with operators, so that the operators keep monitoring RCVs' status and control their actions properly. Therefore, delayed transmissions or losses of control packets can be critical. For instance, while controlling RCVs in real time, delayed transmissions or losses of control packet may cause catastrophic accidents, such as collisions (with obstacle or other RCVs) or crashes.

These days, it is not hard to utilize multiple communication modules together. Also, including researches on Multipath Transmission Control Protocol (MPTCP), various researches on utilizing multiple interfaces simultaneously were conducted [1]. By using MPTCP, if the status of one link deteriorates, a user is able to continuously transmit packets through another link, which improves data communications' reliability and stability. However, this method is not effective in preventing delays or losses of packets already queued or transmitted because it is a reactive approach to cope with such problems. Such delayed transmissions or losses of control packets can be fatal to RCVs moving fast.

To overcome the aforementioned problem, we propose *DupCon*, a new control packet transmission scheme. *DupCon* considers characteristics of packets, and then selectively duplicates only control packets, not data packets. After that, *DupCon* transmits the originals and their duplicates through different networks, respectively. Through these operations, *DupCon* prevents delayed transmissions and losses of control packets, which improves the reliability of controlling RCVs. We designed and implemented *DupCon* on real devices, and we conducted various experiments to verify the effectiveness of *DupCon*. Moreover, we conduct a theoretical analysis on *DupCon* to demonstrate the validity of *DupCon* and analyze the number of subflows to guarantee the target delay.

The contributions of this paper are summarized as follows:

- We propose *DupCon* which reduces the delivery delay of control packets, so RCVs can be controlled more reliably using *DupCon*.
- We devise a new method which duplicates only control packets, not data packets, and transmits the original and its duplicate through different paths.
- We implemented *DupCon* on real devices and conducted many experiments using the devices.
- We conducted the theoretical analysis on *DupCon* to prove the validity of *DupCon*.

It should be noted that we focus on drones among RCVs since drones are the most representative and widely used RCVs these days. However, since *DupCon* is not designed to be specialized in specific RCVs, *DupCon* can be applied to any RCVs.

The remainder of this paper is organized as follows. Firstly, we introduce related work and describe *DupCon*'s novelties and advantages compared to the related work in Section II. We explain the problem statement, the design of *DupCon*, and the implementation details in Section III. We conduct the theoretical analysis on *DupCon* in Section IV. We explain experiments and evaluate the performance of *DupCon* in Section V. Finally, Section VI concludes this paper.

II. Related work

In this section, we introduce various researches relevant to our research. After that, we describe novelties and advantages of *DupCon* compared to the related work.

RCVs are movable dynamically, so the delay of packet transmission may cause difficulty in controlling them properly. Thus, many researches relevant to RCVs focused on reducing the delay. Among them, some researches tried to decrease delay in terms of physical or Media Access Control (MAC) layer. Chandhar *et al.* utilized beamforming of a massive Multiple Input Multiple Output (MIMO) to reduce the communication interference, which provides significant improvement in reduced delay [2]. Similarly, Cai *et al.* focused on MAC and proposed a scheme which utilizes full-duplex radios and

Multi-Packet Reception (MPR) to minimize packet delay in Unmanned Aerial Vehicles (UAV) ad-hoc networks [3]. In addition to the above researches, there are researches which focused on approaches in transport or application layer. Lasota *et al.* proposed a real-time safety system capable of allowing safe Human-Robot Interaction (HRI), which disables Nagle's Algorithm [4] to reduce delay [5]. Also, Van *et al.* proposed a framework that applies Forward Error Correction (FEC) at the application layer in drones to decrease the average number of retransmissions, which reduces communication delay [6].

When employing multiple RCVs, the efficient routing is necessary for low packet transmission delay [7]. Thus, some researchers conducted researches on devising routing algorithms suitable for networks composed of RCVs. Firstly, Alshabtat *et al.* proposed a new routing protocol for drones equipped with directional antennas [8]. Taking advantage of directional antennas, the proposed protocol reduces the number of multi-point relays, which results in the decreased delay of packet transmission. Secondly, Zhang *et al.* proposed an Air-Ground Routing Optimization protocol [9]. Using the protocol, drones select the optimum routing paths among all Air-Air, Air-Ground, or Ground-Ground links at each hop, so that more data is transmitted with short delay.

Supported by advancements of communication and network technologies, it is not hard for devices to utilize multiple interfaces simultaneously using MPTCP. By transmitting packets through multiple paths at the same time, the reliability and the throughput of communications can be improved. Thus, some researchers focused on applying multi-path communications to drones. Firstly, Chiba *et al.* proposed an application layer protocol which duplicates all packets and sends them through two different paths [10]. The authors carried out simulations which show that the proposed protocol reduces transmission delays and packet loss ratios. However, this protocol duplicates all packets without any consideration on characteristics of packets, which is so naïve. Thus, this protocol wastes communication bandwidth unnecessarily and cannot enhance the throughput in spite of utilizing multiple interfaces. Secondly, Shailendra *et al.* proposed a new path scheduler of MPTCP which segregates data and control packets [11]. After the segregation, the proposed scheduler assigns data and control packets to two different interfaces separately to transmit control packets more reliably. However, because the interface for control packet is only used to transmit control packets, it is impossible to improve the throughput of data packet transmissions even when the interface for control packets is idle. Moreover, when using the scheduler, the interface's communication bandwidth cannot be fully utilized. It is because that, compared to data packets, control packets are created less often and the size of control packet is relatively small in general.

Compared to the related work, *DupCon* has novelties and advantages in several perspectives. Firstly, unlike [10], *DupCon* utilizes a new method which considers characteristics of packets, and then selectively duplicates only control packets, not data packets. After that, *DupCon* transmits the originals and their duplicates through different paths to reduce the delay of control packet transmissions. Secondly, in contrast to [11], multiple interfaces are fully utilized to transmit both data and control packets in *DupCon*, so the overall throughput can be improved by using *DupCon*. Thirdly, many related work conducted performance evaluations through only simulations. On the contrary, we implemented *DupCon* in Linux kernel layer and conducted many experiments using real devices with *DupCon*. Finally, in addition to the evaluations through empirical experiments, we conducted theoretical analysis to prove the validity of *DupCon*. By contrast, most of the related work conducted only experiments or simulations without theoretical analysis.

III. System design and Implementation

In this section, we describe the problem statement first, and then we explain the concept of technique used in *DupCon*. After that, we give detailed explanations about *DupCon*'s design and three major components.

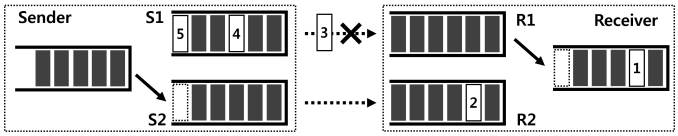
A. Problem statement

While operating drones, an operator consistently exchanges control and data packets with them through wireless communication. Compared to the wired communication, the wireless communication is more unstable, and its available bandwidth fluctuates more frequently. In addition to this, the drones' high mobility exacerbates the instability of communication. If the quality of communications for drones deteriorates, packet transmissions can be extremely delayed, or packets can be lost, which makes drones' flight unstable. Especially, control packets should be delivered timely without losses or delays, so that the operator keeps monitoring drones' status and controls their actions properly. For instance, when using the offboard mode [12] of Pixhawk 4 (PX4) autopilot [13] with MAVLink protocol [14], a drone should receive a control packet every 0.5 s at the latest. Thus, problems of transmitting control packets may cause serious accidents.

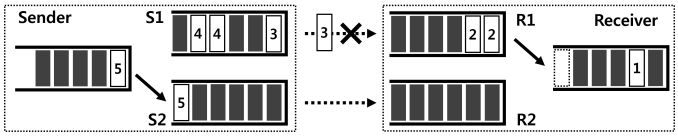
By utilizing MPTCP and multiple interfaces, even when one communication path's status deteriorates, the operator is able to communicate with a drone stably through another communication path. However, this method is a reactive approach to cope with such problems, so it is not effective in preventing losses or delays of packets already queued or transmitted. Even though control packets are not delivered to a drone in a very short time, such momentary problem of transmissions can be serious. For instance, a quadcopter at 30 m/s moves 3 m during just 100 ms. 30 m/s is the typical flying speed of drones for disaster management [15]. Thus, the high reliability with short delay is the important concern when operating drones.

B. Duplicated Control Packet Transmission

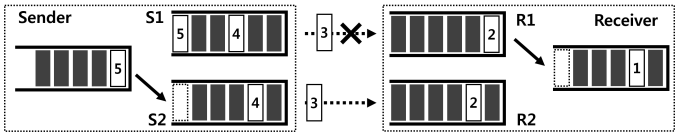
The existing system using MPTCP cannot prevent losses or delays of packets already queued or transmitted as shown in Fig. 1(a). In the figure, there are a sender and a receiver, and each has two wireless interfaces. The interfaces of the sender and receiver are identical, and each interface of the sender is connected to the corresponding interface of the receiver. The sender and receiver conduct communications through two paths, and these communication pairs utilize different channels to prevent interference. White blocks with a number indicate control packets, and black blocks are data packets. In this situation, if the communication link of upper path deteriorates, the third control packet may be lost. In addition, all the packets



(a) The case when using the existing system without duplication.



(b) The case when using the existing system with duplication.



(c) The proposed case when using *DupCon*.

Fig. 1: The packet transmissions of various cases.[1]

queued in the first subflow, including the fourth and fifth control packets, may be delayed or lost.

In the existing system using MPTCP, the default path scheduler chooses a subflow with the shortest Smoothed Round Trip Time (SRTT) when inserting a packet to one of subflows. Thus, the device cannot designate a subflow to transmit a certain packet. Therefore, if the device duplicates a packet and then transmits the original and its duplicate, the both packets may be inserted into the same subflow as shown in Fig. 1(b). The duplicated packet is the following packet of the original, so the time between insertions of them is too short for the communication situation to be changed. For this reason, by using the existing system with MPTCP, it is likely that the original and its duplicate are almost always inserted into the same subflow. If the original and its duplicate are transmitted through the same path, not only the delay or the loss of control packet is not prevented but also the communication bandwidth is wasted unnecessarily. Thus, the original and its duplicate should be transmitted through different paths.

In addition to the aforementioned scheduler using SRTT, by default, the MPTCP contains another scheduler which transmits packets in a round-robin fashion. Thus, if the device sends the original and its duplicate using such scheduler, they can be transmitted through different paths. This is the simplest method to insert the original and its duplicate into different subflows. However, the device using this method transmits packets without considering communication qualities of paths because packets are transmitted in a round-robin fashion simply. Thus, the overall throughput is degraded compared to the case when using the existing system with default scheduler which utilizes SRTT. We will show this degradation in Section V-B.

To overcome the aforementioned limitations, we devise *DupCon* which duplicates control packets and transmits the originals and their duplicates through different paths. The

[1]This figure is drawn to distinctly present the difference between cases, so the ratio of control packets to data packets is exaggerated.

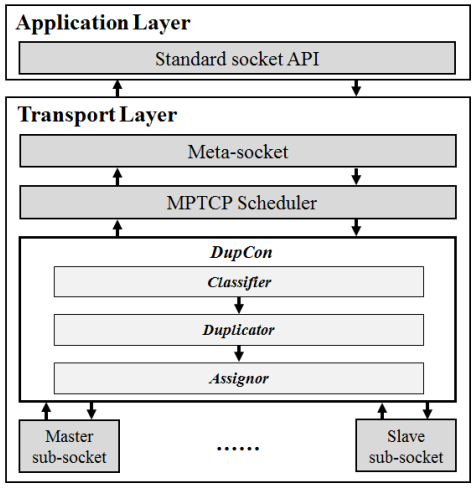


Fig. 2: The structure of *DupCon* in kernel.

Fig. 1(c) shows the concept of *DupCon* briefly. As shown in the figure, the third control packet is successfully transmitted even when one of links abruptly becomes unavailable.

C. Design of DupCon

When using MPTCP, there are two types of socket, meta-socket and sub-socket, as shown in Fig. 2. To conduct the data communication, the application creates the meta-socket first, and then repeatedly adds a sub-socket as the application needs. Thus, there are generally one meta-socket and multiple sub-sockets. Among sub-sockets, the firstly created sub-socket is called the master sub-socket, and the others are slave sub-sockets. When the application sends data using the standard socket Application Programming Interface (API), the meta-socket receives data and then makes segments using the data. After that, the meta-socket hands the segments to the MPTCP scheduler. The MPTCP scheduler takes a role of assigning segments to sub-sockets according to the rule, such as the shortest SRTT first or the round-robin. We implemented *DupCon* between the MPTCP scheduler and the sub-sockets as shown in Fig. 2. *DupCon* is composed of three major components, *Classifier*, *Duplicator*, and *Assignor*.

1) Classifier: *Classifier* takes a role of classifying segments by data type. In the kernel layer, it is not trivial to know which data contains control information. In Linux kernel, `struct sk_buff` is used to contain data and information about each socket's buffer, and Fig. 3 shows the structure of `sk_buff` according to [16]. As shown in the figure, `struct sk_buff` contains many pointers. Among them, *Classifier* utilizes the pointer indicating the starting location of memory for data to access the data directly. By analyzing the directly accessed data, *Classifier* decides whether this segment contains control information or not. If so, *Classifier* hands the segment to *Duplicator*, otherwise, *Classifier* does nothing.

2) Duplicator: If the segment contains control information, *Duplicator* duplicates the segment and then hands both the original and its duplicate to *Assignor*. *Duplicator* does not need to be located in the kernel layer, and it is possible for the application to contain *Duplicator* alternatively. In this case, the application knows whether a certain data contains control

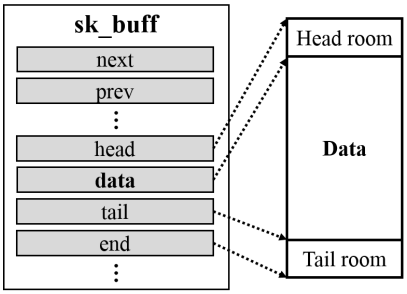


Fig. 3: The structure of `sk_buff` in kernel.

information or not, *Duplicator* is able to duplicate only control data without *Classifier*.

3) Assignor: Assignor takes a role of assigning the original and its duplicate into two sub-sockets belonging to different interfaces. First, *Assignor* forwards the original to the most preferred sub-socket based on the assessment of default scheduler. After that, *Assignor* selects one more sub-socket, which is most preferred among the remaining sub-sockets except the sub-sockets of the interface linked to the previously selected sub-socket. Then, *Assignor* forwards the duplicate to the sub-socket selected later.

After the sender with *DupCon* transmits the original and its duplicate to the receiver, the receiver receives one of them first. *DupCon* in the receiver forwards only this first packet to the application and discards the packet which arrived later. As described above, the receiver side's *DupCon* is relatively simple, so we do not explain the details.

D. Implementation

To implement *DupCon*, we modified source files of `MPTCP v0.91.3` [17], specifically `mptcp_sched.c`, `mptcp_ctrl.c`, `skbuff.c`, `mptcp.h`, and `skbuff.h`. After that, we implemented *DupCon* in Linux kernel of Ubuntu 16.04 LTS on computing board for drone. The computing board on drone takes a role of conducting various computing operations except processing undertook by the flight controller. For performance evaluations, we built a test-bed composed of two *DupCon*-implemented computing boards equipped with two interfaces.

IV. Theoretical analysis

In this section, we conduct a theoretical analysis on the delay of control packet transmission when using *DupCon*. After that, we analyze practical cases with *DupCon* to demonstrate the validity of *DupCon*. In addition, we analyze the number of subflows necessary to guarantee the target delay.

A. Analysis on the delay of control packet transmission

For the analysis, we assume a network situation where there are a sender and a receiver and they have multiple identical interfaces. Each interface of the sender is connected to the corresponding interface of the receiver. These communication pairs utilize different channels to prevent interference. For example, the number of pairs is two in the situation shown in Fig. 1. The sender transmits data packets to the receiver continuously, whereas the control packet is created and sent periodically. The order of control packets is distinguished from that of data packets. In this analysis, the delay means the time between one moment when a packet is forwarded to a sub-socket in the sender and the other moment when a sub-socket in the receiver forwards the packet to the upper layer. In the assumed situation, we focus on the delay of only control packet transmission. Compared to data packets, the control packets are created much less often, and the time between consecutive control packet transmissions is relatively long. Therefore, reordering control packets[2] very rarely occurs, so the delay caused by reordering control packets is not considered in this analysis.

When the communications are in a steady-state, all the communications through subflows are identical. In this situation, the delay of ith subflow, d_i^{total}, is composed of three component delays as follows:

$$d_i^{total} = d_i^{sq} + d_i^{net} + d_i^{rq}, \tag{1}$$

where d_i^{sq}, d_i^{net}, and d_i^{rq} indicate delays caused by buffer queuing in the sender, propagation delay, and buffer queuing in the receiver, respectively.

Firstly, d_i^{sq} in Eq. (1) is defined as follows:

$$d_i^{sq} = size_i^{sq}/rate_i^{send}. \tag{2}$$

Because the sender and receiver conduct the communication without any constraints, the sending queue is always filled with packets. Therefore, if the sender's queue size and sending rate are $size_i^{sq}$ and $rate_i^{send}$ respectively, the time between one moment when a control packet is inserted to the queue and the other moment when the packet is transmitted from the sender is as shown in Eq. (2).

If W and RTT indicate the maximum size of congestion window and the round trip time in the steady-state, the size of congestion window increases from W/2 to W and then drops to W/2 in $RTT \cdot W/2$, over and over [18]. Thus, the relation between W and $rate_i^{send}$ can be represented as follows:

$$\begin{aligned} rate_i^{send} &= \{(3W^2/8) \cdot MSS\}/(RTT \cdot W/2) \\ &= 3W \cdot MSS/4RTT, \end{aligned} \tag{3}$$

where MSS means the maximum segment size. Eq. (3) can be transformed in terms of W as follows:

$$W = 4rate_i^{send} \cdot RTT/3MSS. \tag{4}$$

Fundamentally, in the network layer in kernel, the initial size of buffer is set to the default buffer size, and then the operating system dynamically adjusts the size if necessary. In addition, with considering fluctuation of sending rate, the size of send buffer[3] is set to twice $W \cdot MSS$. Using Eq. (4), $size_i^{sq}$ can be represented as follows:

$$\begin{aligned} size_i^{sq} &= 2W \cdot MSS \\ &= 8rate_i^{send} \cdot RTT/3. \end{aligned} \tag{5}$$

Using Eq. (5), Eq. (2) can be transformed as follows:

$$d_i^{sq} = 8RTT/3. \tag{6}$$

[2]When using MPTCP, reordering packets frequently happens because packets are transmitted through multiple interfaces.

[3]The send buffer and receive buffer are terms used in Linux kernel.

Secondly, because each interface of the sender is directly connected to the corresponding interface of the receiver, d_i^{net} in Eq. (1) is defined as follows:

$$d_i^{net} = dist/c, \tag{7}$$

where c and $dist$ are the speed of light and the distance between the sender and receiver respectively.

The third component delay in Eq. (1) is defined as follows:

$$d_i^{rq} = size_i^{rq}/rate_i^{proc}, \tag{8}$$

where $size_i^{rq}$ and $rate_i^{proc}$ are the receiver's queue size and processing rate respectively. Similar to the send buffer, the size of receive buffer is set to twice the advertised window size multiplied by MSS. The communication between the sender and receiver in this situation is closed loop, so the amount of packets in the receiver's receive buffer should be less than that of the sender's send buffer. Thus, the upper bound of $size_i^{rq}$ is $size_i^{sq}$ in Eq. (5). As a result, according to Eq. (8), the upper bound of d_i^{rq} is $8rate_i^{send} \cdot RTT/3rate_i^{proc}$. The speed of processing packets in the receiver is relatively much faster than that of sending packets in the sender. Thus, d_i^{rq} is much less than d_i^{sq} in general.

Until now, we considered the delays without retransmission. However, retransmissions can occur due to various causes, so we should consider the delay caused by retransmissions additionally. Every time a retransmission occurs, a lost packet is inserted to the send buffer and transmitted again, so $d_i^{sq}+d_i^{net}$ is added to d_i^{total}. Therefore, this addition is repeated as many times as the number of retransmissions until the packet is delivered successfully. If p is the probability that a packet is transmitted successfully and q, (1-p), is the probability of transmission failure, the probability that the packet is transmitted successfully after the transmission failure occurs n times is $q^n \cdot p$. Thus, if $d_i^{sq}+d_i^{net}$ is briefly expressed as d_i^{rt}, the expected delay with considering retransmissions, $E[d_i^{total}]$, is as follows:

$$\begin{aligned} E[d_i^{total}] &= d_i^{rq} + d_i^{rt} \cdot p + 2d_i^{rt} \cdot qp + 3d_i^{rt} \cdot q^2 p + \cdots \\ &= d_i^{rq} + d_i^{rt} \cdot \sum_{k=1}^{\infty} kq^{k-1}p. \end{aligned} \tag{9}$$

In Eq. (9), $\sum_{k=1}^{\infty} kq^{k-1}p$ is equal to the expectation of geometric distribution, so Eq. (9) can be transformed as follows:

$$E[d_i^{total}] = d_i^{rq} + \frac{d_i^{rt}}{p}. \tag{10}$$

When using *DupCon*, the control packet is duplicated, and the original and its duplicate are transmitted through different interfaces as shown in Fig. 1(c). If the number of interfaces the sender or the receiver has is n, the original packet and $n-1$ duplicate packets are transmitted through n interfaces. Thus, the probability that at least one of them is delivered without retransmission, p^{system}, is as follows:

$$\begin{aligned} p^{system} &= 1 - (1-p_1)(1-p_2)\cdots(1-p_n) \\ &= 1 - \prod_{i=1}^{n}(1-p_i), \end{aligned} \tag{11}$$

where p_i is the probability that the transmission through the ith interface is successful. As we mentioned, all the subflows' communications are identical, so all p_i, i = 1, ..., n, are the same. Therefore, Eq. (11) can be transformed as follows:

$$p^{system} = 1 - (1-p)^n = 1 - q^n. \tag{12}$$

Thus, based on Eqs. (10) and (12), the delay when using *DupCon* with n interfaces is as follows:

$$E[d^{total}] = d^{rq} + \frac{d^{rt}}{p^{system}} = d^{rq} + \frac{d^{rt}}{1-q^n}. \tag{13}$$

According to Eq. (13), the delay increases slowly although q is increased, which means that *DupCon* enables users to control drones with short delay even when the quality of communication deteriorates. As explained before, $d^{rq} \ll d_i^{sq} < (d_i^{sq}+d_i^{net}) = d^{rt}$. Thus, Eq. (13) can be simplified as follows:

$$E[d^{total}] \approx \frac{d^{rt}}{1-q^n}. \tag{14}$$

Based on Eq. (14), Fig. 4 shows the ratio of delay when using *DupCon* to that when using the existing system with one interface, as the packet loss probability increases. For instance, if the packet loss probability is 0.25, *DupCon* with two interfaces is able to successfully transmit a control packet within 80% of the delay when using the existing system on average. It means that, if an operator uses *DupCon*, the stability of controlling drones can be significantly improved by adding just one more interface.

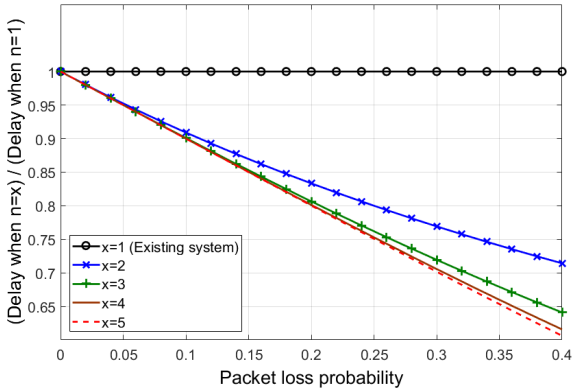
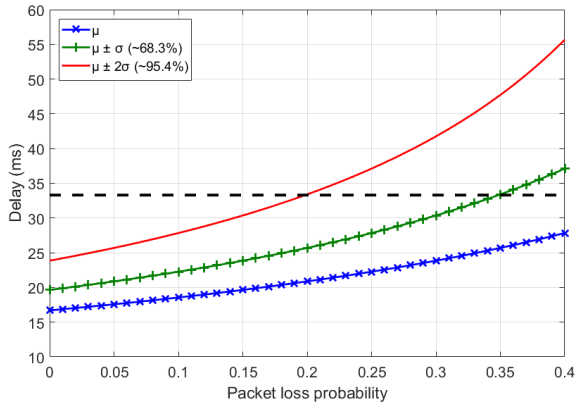


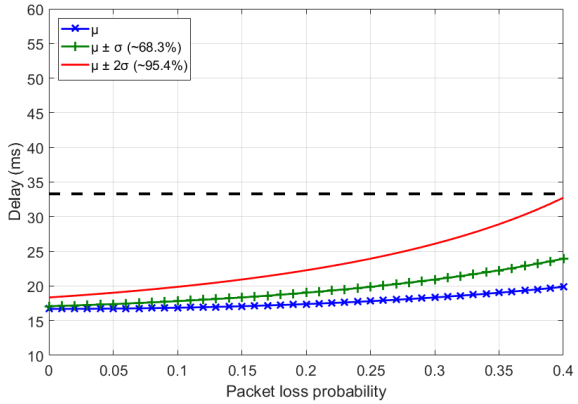
Fig. 4: The ratio of delay when using *DupCon* to that when using the existing system.

B. Analysis on practical cases with DupCon

To conduct practical analysis on *DupCon*, we assume a situation where a drone communicates with the Ground Control System (GCS) using Wi-Fi connectivity and the operator controls the drone while watching the video collected by the drone. In this situation, if a 720p30 stream is encoded at 1 Mbps and the available bandwidth is 2 Mbps, it takes 118.7 ms for the operator to see the collected video according to [19]. In other words, while the operator senses the need for a flight change and issues a directive, the drone has moved about 1.8 m if the drone is flying at 15 m/s. In addition to this, including analysis results in [20] and [21] which analyzed drone networks experimentally, we analyze the delay of control packet transmission when using *DupCon* with two



(a) When using the existing system.



(b) When using *DupCon*.

Fig. 5: The analytic delay results in the assumed scenario.

interfaces compared to the existing system. Fig. 5 shows the results. In the assumed situation, if the operator or GCS finds some problems based on the information from the ith packet transmitted by the drone, the operator or GCS sends a control packet immediately. To apply the countermeasure properly, the control packet should be delivered to the drone before the i+1th packet arrives at the GCS. In other words, the target delay in this situation is the time between consecutive packet arrivals at the GCS. The dotted line in Fig. 5 indicates the target delay.

According to Fig. 5(a) which shows the result when using the existing system, if the packet loss probability is under 0.2, about 95% of packets can be transmitted within the target delay. However, if the packet loss probability is increased up to 0.35, only about 68% of packets can be delivered within the target delay. On the contrary, when using *DupCon*, even when the packet loss probability is 0.4, which represents the severe communication situation, most of the packets can be delivered to the drone within the target delay.

C. Analysis on the number of subflows necessary to guarantee the target delay

As described above, by using *DupCon*, much more control packets can be delivered within the target delay. Based on Eq. (14), if Eq. (15) is valid, it is guaranteed that control packets are delivered to the drone within the target delay.

$$E[d^{total}] \approx \frac{d^{rt}}{1-q^n} = \frac{d^{sq}+d^{net}}{1-q^n} \leq d^{target}, \tag{15}$$

where d^{target} indicates the target delay. Eq. (15) can be transformed as follows:

$$q^n \leq 1 - \frac{d^{sq}+d^{net}}{d^{target}}, \tag{16}$$

and Eq. (16) also can be transformed as follows:

$$n \geq log_q(1 - \frac{d^{sq}+d^{net}}{d^{target}}). \tag{17}$$

Therefore, using Eq. (17), it is possible to obtain the number of required interfaces to guarantee that 68%, 95%, or 99% of control packets are transmitted to the drone within the target delay by putting $\mu+\sigma$, $\mu+2\sigma$, or $\mu+3\sigma$ in d^{target} respectively.

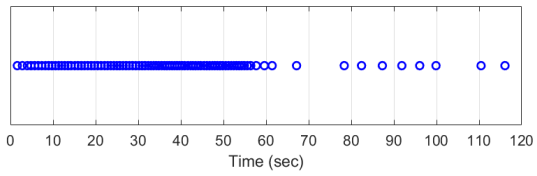
V. Performance Evaluation

In this section, we explain various experiments and show the results which verify the performance of *DupCon*. Firstly, we explain the experiment to show the limitation of existing system. After that, we conduct the performance evaluation by comparing *DupCon* and existing systems, and explain the results that *DupCon* outperforms the others.

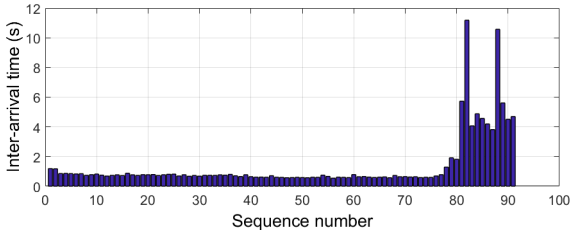
A. Performance of existing system in the problem situation

In the existing system for drone networks, each drone has just one interface and communicates with other drones and GCS via ad-hoc network. Thus, if the communication quality of only one interface deteriorates, drones have trouble in receiving control packets from GCS. The loss of connection implies the loss of control for drones, and delayed transmissions or losses of control packet may cause catastrophic accidents. We conducted an experiment to confirm this problem. There were two computing boards for drones, one sender and one receiver, in this experiment, and each board had one IEEE 802.11g interface. After the experiment began, the sender transmitted data and control packets to the receiver. The data packets were sent to the receiver continuously. However, we defined the control packets which were created and transmitted with a determined time interval, 0.5 s. After some time, we removed the sender's antenna to emulate the situation when the quality of communication link became poor. During the experiment, we recorded moments when the receiver received control packets and time intervals between consecutive moments.

Fig. 6(a) shows the arrival moments of control packets, and it is easy to see that the receiver received control packets periodically early in the experiment. However, after about 60 s, because of the poor communication condition, delayed transmissions or losses of control packets occurred frequently, so the receivers received the control packets occasionally. More specifically, Fig. 6(b) shows the inter-arrival times of control packets. As shown in the figure, there is not much difference between inter-arrival times early in the experiment. However, after the communication quality deteriorated, the values of inter-arrival time were increased significantly. As shown in these results, it is hard for the existing system to deal with such problem situation.



(a) Arrival moments of control packets.



(b) Inter-arrival times of control packets.

Fig. 6: The performance of existing system in the problem situation.

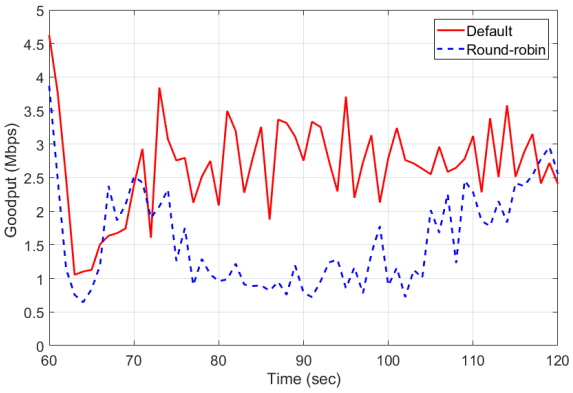
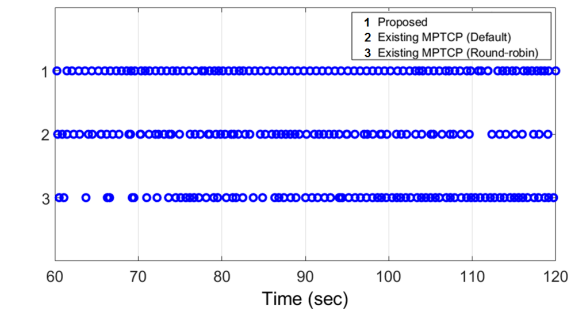


Fig. 7: The round-robin scheduler's problem in case of poor communication quality.

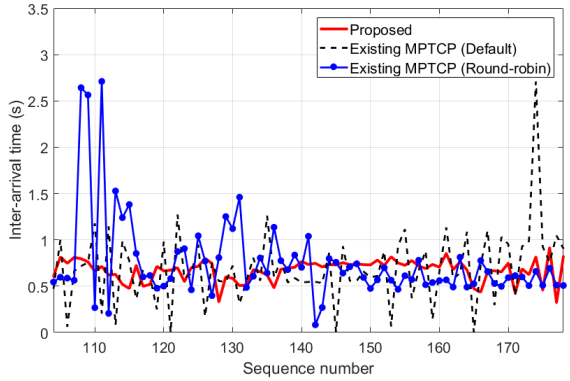
B. Evaluation on transmitting packets in a round-robin fashion

As explained in Section III-B, the straightforward method to transmit the original and its duplicate packet through different paths is to send them using multiple interfaces in a round-robin fashion. However, since the sender using this method transmits packets without considering the communication situation, the overall performance of data communication is degraded. We compared performances of two schedulers in MPTCP, the round-robin scheduler and the default scheduler which selects the subflow with the shortest SRTT. In this experiment, there were two computing boards for drones, one sender and one receiver, and each board had two interfaces. Each interface of the sender was connected to the corresponding interface of the receiver. Similar to the previous experiment, the sender started to transmit packets to the receiver, and after some time, we made the quality of communication poor.

Fig. 7 shows performances of two schedulers in case of poor communication quality. As shown in the figure, the round-robin scheduler could not deal with the change of communication situation, whereas the default scheduler



(a) Arrival moments of control packets in each case.



(b) Inter-arrival times of control packets.

Fig. 8: Performance comparison of different cases.

properly conducted data communication through the interface with better communication quality. Thus, the default scheduler outperformed the round-robin scheduler, which means that the straightforward method explained in the previous paragraph is unsuitable for *DupCon*. For this reason, we designed *DupCon* to be compatible with the default scheduler as explained in Section III-C.

The throughput of data packet transmission is scarcely affected by *DupCon* because *DupCon* duplicates only control packets which are created less often and the size of control packets is relatively small compared to data packets.

C. Evaluation on the performance of DupCon

We conducted an experiment to validate the effectiveness of *DupCon*. Similar to the experiment in Section V-A, the sender started to transmit data and control packets to the receiver through two communication paths. After some time, we made the quality of communication poor. The data packets were delivered to the receiver continuously, but the control packets were transmitted with a determined time interval, 0.5 s. However, unlike the experiment in Section V-A, the sender duplicated every control packet, and then transmitted the original and its duplicate to the receiver in this experiment.

We compared three cases in this experiment. The first case is when using *DupCon*. In the second and third cases, the sender simply duplicated control packets and transmitted them using the existing system with MPTCP, so it was not guaranteed that the original and its duplicate were transmitted through different paths. The MPTCP in the second case used

TABLE I: Mean and standard deviation of inter-arrival times of control packets.

	Normal situation		Problematic situation	
Case	***Mean***	***Std***	***Mean***	***Std***
1	0.5849	0.1155	0.6591	0.1665
2	0.6095	0.142	0.8929	0.6721
3	0.5879	0.1638	0.6993	0.4067

the default scheduler, whereas the round-robin scheduler was used in the third case. When the communication quality is good, there is no problem in data communication naturally. Thus, Fig. 8 shows only the results after the communication condition deteriorated.

Firstly, Fig. 8(a) shows the arrival moments of control packets when the quality of communication was poor. As shown in the figure, in the first case when using *DupCon*, the rings are placed almost uniformly, which means that delayed transmissions or losses of control packets rarely occurred. On the contrary, in the second and third cases, delayed transmissions or losses of control packets frequently occurred, so distances between consecutive points are considerably different. Secondly, Fig. 8(b) shows the results in terms of inter-arrival times of control packets. In the figure, compared to the result of the first case, there are more significant fluctuations in the results of the second and third cases, which results in unstable control of drones. In addition, Table I shows means and standard deviations of inter-arrival times in each case. When the communication was in a normal situation, there is not much difference between values of cases. However, when the quality of communication was poor, values of the second and third cases are much larger than those of the first case. In conclusion, through the results of experiments, we confirm that *DupCon* improves the reliability and stability of controlling drones.

VI. Conclusion

These days, RCVs are receiving huge attention in various fields, and among them, drones are the most representative and widely used RCVs. Since drones' mobility level is high and drones are able to fly fast in the air, it is so important to control drones reliably and stably. However, due to the instability of wireless communications, control packets can be delayed or lost, which may cause collisions or crashes. To overcome this problem, we propose *DupCon* which duplicates control packets and transmits the originals and their duplicates through different paths. We designed and implemented *DupCon* on real devices. Moreover, we conducted a theoretical analysis on *DupCon* and various experiments to verify the effectiveness of *DupCon*. Through the analysis and experiments, we confirmed that *DupCon* improves the reliability and stability of control packet transmission.

In future work, we will implement *DupCon* on various RCVs and evaluate *DupCon* in practical situations. Furthermore, we plan to conduct varied evaluations on *DupCon* using the multi-UAV simulator we devised [22].

Acknowledgment

This research was supported in part by Unmanned Vehicles Advanced Core Technology Research and Development Program through the Unmanned Vehicle Advanced Research Center(UVARC) funded by the Ministry of Science, ICT and Future Planning, the Republic of Korea (NRF-2016M1B3A1A01937599), and in part by the Brain Korea 21 Plus Project in 2018.

References

[1] M. Scharf and A. Ford, "Multipath tcp (mptcp) application interface considerations," Tech. Rep., 2013.

[2] P. Chandhar, D. Danev, and E. G. Larsson, "Massive mimo for communications with drone swarms," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1604–1629, 2018.

[3] Y. Cai, F. R. Yu, J. Li, Y. Zhou, and L. Lamont, "Medium access control for unmanned aerial vehicle (uav) ad-hoc networks with full-duplex radios and multipacket reception capability," *IEEE Transactions on Vehicular Technology*, vol. 62, no. 1, pp. 390–394, 2013.

[4] J. Nagle, "Congestion control in ip/tcp internetworks," 1984.

[5] P. A. Lasota, G. F. Rossano, and J. A. Shah, "Toward safe close-proximity human-robot interaction with standard industrial robots," in *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*. IEEE, 2014, pp. 339–344.

[6] B. Van den Bergh, A. Chiumento, and S. Pollin, "Ultra-reliable ieee 802.11 for uav video streaming: From network to application," in *Advances in Ubiquitous Networking 2*. Springer, 2017, pp. 637–647.

[7] J. Lee, K. Kim, S. Yoo, A. Y. Chung, J. Y. Lee, S. J. Park, and H. Kim, "Constructing a reliable and fast recoverable network for drones," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[8] A. I. Alshabtat, L. Dong, J. Li, and F. Yang, "Low latency routing algorithm for unmanned aerial vehicles ad-hoc networks," *International Journal of Electrical and Computer Engineering*, vol. 6, no. 1, pp. 48–54, 2010.

[9] J. Zhang, K. Li, A. Reinhardt, S. S. Kanhere *et al.*, "Agro-optimal routing with low latency and energy consumption in wireless sensor networks with aerial relay nodes," 2014.

[10] N. Chiba, M. Ogura, R. Nakamura, and H. Hadama, "Dual transmission protocol for video signal transfer for real-time remote vehicle control," in *Communications (APCC), 2014 Asia-Pacific Conference on*. IEEE, 2014, pp. 315–320.

[11] S. Shailendra, K. Aniruddh, B. Panigrahi, and A. Simha, "Multipath tcp path scheduler for drones: A segregation of control and user data," in *Proceedings of the 18th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2017, p. 40.

[12] P. D. Team. (2018) Offboard - px4 user guide. [Online]. Available: https://docs.px4.io/en/flight_modes/offboard.html

[13] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 6235–6240.

[14] L. Meier, J. Camacho, B. Godbolt, J. Goppert, L. Heng, M. Lizarraga *et al.*, "Mavlink: Micro air vehicle communication protocol," *Online]. Tillgänglig: http://qgroundcontrol.org/mavlink/start.[Hämtad 2014-05-22]*, 2013.

[15] P. Chandhar and E. G. Larsson, "Massive mimo for drone communications: Applications, case studies and future directions," *arXiv preprint arXiv:1711.07668*, 2017.

[16] A. Jaakkola, "Implementation of transmission control protocol in linux," in *Proceedings of Seminar on Network Protocols in Operating Systems*, p. 10.

[17] C. Paasch, S. Barré *et al.*, "Multipath tcp in the linux kernel," *Available from {www.multipath-tcp.org}*, 2013.

[18] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the tcp congestion avoidance algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[19] D. Barrett and P. Desai, "Low-latency design considerations for video-enabled drones," *Available from {http://www.ti.com/lit/wp/spry301/spry301.pdf}*, 2016.

[20] R. Muzaffar, "Routing and video streaming in drone networks," Ph.D. dissertation, Queen Mary University of London, 2017.

[21] M. Asadpour, D. Giustiniano, and K. A. Hummel, "From ground to aerial communication: dissecting wlan 802.11 n for the drones," in *Proceedings of the 8th ACM international workshop on Wireless network testbeds, experimental evaluation & characterization*. ACM, 2013, pp. 25–32.

[22] W. G. La, S. Park, and H. Kim, "D-muns: Distributed multiple uavs' network simulator," in *Ubiquitous and Future Networks (ICUFN), 2017 Ninth International Conference on*. IEEE, 2017, pp. 15–17.