Implementation of Socket Priority Module for Unmanned Aerial Vehicle Network using FlyNetSimulator

Jauzak Hussaini Windiatmaja, Johannes Calvin Tjahaja, Kgs. Al Amin, Riri Fitri Sari*
Department of Electrical Engineering, Faculty of Engineering, University of Indonesia

E-mail: *riri@ui.ac.id

Abstract. Unmanned aerial vehicle (UAV) is an aircraft without a human pilot onboard and a sort of unmanned vehicle. UAV Network research activities in the real world can pose several challenges, especially in urban environments or in densely populated environments. This is due to the nature of the UAV which requires a large area of operation. Therefore, digital simulation is vital in conducting research related to UAV and its networks. One of the UAV simulators recently developed is FlyNetSim. FlyNetSim is an open-source UAV Network Simulator based on NS-3 and Ardupilot. In the application, UAV can have some network-related problems. One of them is when a UAV is in an environment that has a lot of interfering networks because it is contending a network channel. This can cause network delay. These delays will raise with the increase in the number of nodes that interfere with the UAV. To overcome the problem of interfering nodes, we propose to add a priority set module for UAV and Ground Control Station (GCS). The proposed module will set priority for the UAV and GCS socket. This module is intended to give priority to packets sent by UAVs or GCS. Later the packet that has priority will be sent first to reduce the delay caused by the appearance of interfering nodes. Setting priority in the socket may reduce the average delay caused by contending nodes. Test result shows that there is a 16.2% reduction of average delay. Test result also shows that there is a 4% raise in packet delivery ratio.

Keywords: Unmanned Aerial Vehicles, NS-3, FlyNetSimulator, Socket Priority

1. Introduction
Unmanned Aerial Vehicle (UAV) research is emerging due to the rise of UAV usage in daily life. Research conducted by Skylogic Research shows that almost 68% of UAVs are purchased for professional purposes [1]. UAV usage for professional purposes is diverse, starting from agricultural surveillance activities, disaster management, to environmental monitoring [2][3]. To the extension of wireless coverage, UAV can be integrated into communication infrastructure [4]. Research related to UAV and Network is presented in [5][6]. UAV Network research activities in real life can pose some challenges, especially in the urban area or in a very densely populated area. This is due to the nature of the UAV which requires a large area of operation. Therefore, digital simulation is vital in conducting research related to UAVs and its networks.

One of the existing UAV network simulators is FlyNetSim. FlyNetSim is built on NS-3 and Ardupilot. NS-3 is an open-source network simulator with wide community support [7]. Ardupilot is a simulator that is used to model UAV with its characteristics such as control, navigation, and route plan.
ArduPilot also supports a broad type of vehicles such as helicopters, multi-copters, fixed-wing plane, ships, submarines, and rovers [8]. FlyNetSim can capture interdependencies between the operation of UAV and the environment of the network, such as sensing, navigating, and condition of battery [9].

In the application, UAV can have some network-related problems. One of them is when a UAV is in an environment that has a lot of interfering networks because it is contending a network channel. This can cause network delays. These delays will increase with the increase in the number of nodes that interfere with the UAV. Therefore, we propose a socket priority module. The proposed module set priority for the UAV and GCS socket. This priority is intended to give priority to packets sent by UAVs and GCS. Later the packet that has priority will be sent first to reduce the delay caused by the appearance of interfering nodes.

The rest of this paper is structured as follows. Section II reviews the simulation tools related to UAV, network, and a combination of the two. Section III discusses the details of the module we proposed and their implementation on FlyNetSim. Section IV discusses the results and analysis of the implementation carried out. Finally, Section V concludes the results of the research conducted.

2. UAV and Network Simulator
This section lays out the overview of network simulators and UAV simulators. We present the epitome of FlyNetSimulator, a joint UAV-Network simulator that is used to implement the proposed socket priority algorithm.

![ArduPilot Basic Structure](image)

**Figure 1.** ArduPilot Basic Structure [14]

2.1. UAV Simulator
The main function of UAV simulators is to archetype functional aspects and aerodynamics aspects of UAV systems. Some simulators like H-SIM Flight Simulator only simulate human-controlled vehicles and autonomous vehicles [10]. PX4 and Ardupilot are the most used autopilot simulators. Ardupilot is a pioneer in open-source system of autopilot simulator that supports helicopters, multi-copters, fixed-wing aircraft, rovers, antenna trackers, and even submarines. Ardupilot has a wider range of supported platforms than PX4, such as Pixhawk [11], NAVIO [12], and Erle-Brain [13]. The fundamental structure of Ardupilot consists of five main components, i.e. vehicle code, hardware abstraction layer (AP_HAL), open-source libraries, tools directories, external support code (i.e. dronkekit and mavlink) [14]. The fundamental structure of Ardupilot is described in Figure 1.

Some UAV simulators are built based on Ardupilot Simulator to broaden its functionalities. For example, Gazebo [15] broaden the scope of simulated vehicles to include planes, small robots, and land
rovers using Robot Operating System [16]. Gazebo is a very heavy simulator, so that it might be hard to run on one environment when doing simulation on many vehicles. Other simulators based on Ardupilot Simulator are XPlane-10 and RealFlight. However, most of them only focus on navigation and motion. To create FlyNetSim, Software In The Loop (SITL) is used [14]. SITL is a simple simulator build on Ardupilot that supports autopilot navigation. SITL allows ArduPilot to run on end-user computer instantaneously, without requiring any special hardware. It exploits the way that ArduPilot is a compact autopilot that can run on many platforms.

When running in SITL the sensor data originated from a flight dynamics model in a flight simulator. Ardupilot has a wide scope of vehicle simulators built-in. It can be interfaced with several external simulators. This allows ArduPilot to be used on a broad variety of vehicle types, i.e. ground vehicles, fixed-wing aircraft, multi-rotor aircraft, underwater vehicles, antenna trackers, camera gimbals, and some kind of sensors such as optical flow and lidar.

2.2. Network Simulator
There exist many simulators that offer full or a part of network simulation. Most simulators are discrete event-driven. Discrete event-driven means that it forms a timeline of events that are sequentially processed. One pioneer of network simulator is TOSSIM [17]. TOSSIM is created to model the transmissions of wireless sensors.

OPNET [18] is a commercial network simulator. OPNET provides wireless network simulation and supports a broad variety of technologies and protocols. NS-3 [7] and OMNET++ [19] are the most used network simulators by the researcher, among other open-source network simulators. NS-3 and OMNET++ simulators are also a discrete-event driven. NS-3 can simulate many technologies. NS-3 is capable to embed diverse statistical models for traffic generation, mobility, and channel gain. It provides comprehensive modeling of physical layer function. NS-3 supports 4G, 5G, and WiFi communication protocols. NS-3 can also support extended protocols and options at different layers of the network. Furthermore, NS-3 can be effectively used to interface with libraries, external systems, and external applications.

2.3. FlyNetSimulator
FlyNetSim is a non-proprietary UAV and Network Simulator built on the base of NS-3 and Ardupilot. FlyNetSim is capable to complement the experiment with a simulated edge server, UAV, and wireless networks. Unlike other integrated UAV-network simulators, FlyNetSim is capable to support various scenarios in terms of UAV controls, e.g., data stream transport, telemetry controls, and multi-UAV usage. FlyNetSim is also capable of emulating a wide range of network schemes, i.e. ad-hoc communications of device-to-device, heterogeneous networks system, multi-hop and multi-path communications, and also different mobility models. FlyNetSim simulator is more efficient compared to other simulators in terms of computational resource usage and scalability. Hence, FlyNetSim simulator can support the complex tasks of information autonomy framework. Using this simulator, we may simulate the real-world network conditions with varying load and mobility [5].

FlyNetSim architecture is described in Figure 2. The FlyNetSim architecture inheres four main components: Fly simulator to commence UAV, GCS, and its visualization tools; Network simulator to simulate real-life network infrastructure and its environment (i.e. contending nodes); Middleware to supports two-way communications based on ZMQ through the network; Autogen to produce a map for the UAV components (i.e. sensors) and the corresponding network nodes.

2.4. Socket Priority
Socket is an endpoint of a bidirectional transmission link connecting two applications run on a network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. Commonly, a server runs on a dedicated computer and has a socket with specific port number. Server task is listening to the socket for a client to make a connection request. To establish a connection, client should know the hostname of the machine on which the server is running and the port
number on which the server is listening. The client is also obligated to identify itself to the server so it binds to a local port number that will be used during a connection session. In the case of everything works out positively, the server acknowledges the connection [20].

Socket allows the network to set some options. One of them is Socket Priority. Socket priority allows the network to set the input message priority for this socket. Valid values for this parameter are usually in hierarchical integer (i.e 0-6). If we set the priority to 0, the application's priority defined in the network services database is used. For sockets with a priority value of 1, traffic is read in and processed even during input list shutdown and if traffic limiting is enabled for this server, traffic limiting checks are bypassed. When there are resource constraints, packets with the largest priority values are discarded first. The priority for an individual socket overrides the application's priority defined in the network services database. This option is valid only for TCP and UDP sockets [21].

3. Method
The packets are prioritized from trusted nodes by giving the socket from trusted node a priority. The priority for an individual socket overrides the application's default priority defined in the network. By giving higher priority to trusted node, we aim to out-weight the trusted node from the interfering node, so that the access point will process the trusted packet first. The proposed algorithms are implemented in FlyNetSimulator that is based on Ardupilot and NS3. The implementation is based on C++.

3.1. Proposed Algorithm for UAV Socket Priority Module
Figure 3 shows the flowchart of the proposed algorithm. The flowchart started with checking if the socket used to send the packets are trusted. The trusted sockets consist of UAV nodes and Ground Control Station node. If the socket came from trusted node the simulator then set a priority tag in the socket. The Socket then sent the desired packet, either in the form of a UAV command, GCS command, or just contending packets.

At the receiving side, when receiving packets, it first checks the incoming packets, whether it has a priority socket or not. If the packet came from the priority socket, then the receiver will prioritize the packet to be received. If the packet does not come from the priority socket, then the receiver will ignore the packet.
3.2. Implementation in FlyNetSim

The implementation of our proposed method is described in two algorithms: Algorithm 1 for packet send and Algorithm 2 for packet receive.

**Algorithm 1. Send Packet**

```plaintext
1    procedure SendMsg(socket, message)
2    while send request
3    socket <- socket priority
4    packet <- create packet(message)
5    socket -> send packet
6    end while
```

Algorithm 1 shows the implementation of sending trusted packets. SendMsg is a procedure to send packet from UAV or Ground Control Station node. SendMsg expects an input of socket source and the message to send. The algorithm starts with checking if there is a packet send request. If send request exists, then SendMsg creates a socket priority for the socket source input. Subsequently, the message is encapsulated in the packet and the packets are sent via the socket created.

**Algorithm 2. Receive Packet**

```plaintext
1    procedure RcvPacket(packet, address)
2    while packet receive request
3    if priority exist in socket
4    receive the packet
5    message <- parse packet message
6    end if
7    end while
```

Algorithm 2 shows the implementation in receiving packets. RcvPacket is a procedure to receive packet, whether from a trusted node or a contending node. RcvPacket expects an input of packet received

![Flowchart of the algorithm for UAV Socket Priority](image-url)
and sender address. The algorithm starts with checking if there is a packet receive request. If packet receive exist, then RcvPacket checks whether the packet came from the priority socket or not. If the packet came from a priority socket, RcvPacket then receives the packet and parse the message inside the packet.

3.3. Test Scenario

We measure the delay and packet delivery ratio (PDR) for evaluating the proposed method. Delay and PDR are measured in two scenarios which have various parameters of interfering nodes, traffic rates of interfering nodes (in Mbps), and packet size each of each interfering node (in bytes). Each scenario uses LTE as the network type. Each scenario runs with and without the implementation of socket priority module to find out if the module created successfully reduces delay and affects the packet delivery ratio.

Detailed scenarios are described in Table 1. The first scenario (S01) uses 1 interfering node, 4 Mbps traffic rates, and 100 bytes packet size on each contending node. The second scenario (S02) uses 2 interfering nodes, 6 Mbps traffic rates, and 120 bytes packet size on each contending node. The number of interfering nodes on two scenarios are defined to represent single interference (1 node) and multi interference (more than 1 node) situation. Traffic rates, packet size, and the number of interfering nodes are defined increasing (from 4 to 5, from 100 to 120, and from 1 to 2) to discover whether these three variables affect the performance of the UAV network.

| Scenario ID | Interfering Nodes | Traffic Rates | Packet Size Each Node |
|-------------|-------------------|---------------|-----------------------|
| S01         | 1                 | 4             | 100                   |
| S02         | 2                 | 6             | 120                   |

4. Results and Analysis

The results and analysis of the implemented method of UAV Socket Priority Module in FlyNetSim are described as follows.

4.1. Results

Delay test scenario results are described in Figure 4. The result shows a reduction in the average delay during packet transmission from Unmanned Aerial Vehicle to Ground Control Station. This reduction in delay is caused by using priority socket on the trusted node, thus the GCS only accepts incoming packets from the UAV and not by other interfering nodes. In scenario 1, the network average delay reaches a peak at 20 ms when not using the set priority socket on the trusted nodes, whereas the average delay reached a peak at 17 ms when using the set priority socket on the trusted nodes which means there is a reduction by 3 ms. There is an 18.6% reduction in the network average delay in scenario 1. In scenario 2, the network average delay reaches a peak at 50 ms when not using the set priority socket on the trusted nodes, whereas the average delay reached a peak at 39 ms when using the set priority socket on the trusted nodes which means there is a reduction by 11 ms. There is a 13.9% reduction in the network average delay in scenario 2.

Packet Delivery Ratio test scenario results are described in Figure 5. The result shows a slight rise in packet delay during packet transmission from Unmanned Aerial Vehicle to Ground Control Station. This slight raise in packet delivery ratio is caused by using priority socket on the trusted node, thus the GCS only accepts incoming packets from the UAV and not by other interfering nodes. In scenario 1, the packet delivery ratio is 91% when not using the set priority socket on the trusted nodes, whereas the packet delivery ratio is 94% when using the set priority socket on the trusted nodes. There is a 3% increase in the ratio. In scenario 2, the packet delivery ratio is 89% when not using the set priority socket on the trusted nodes, whereas the packet delivery ratio is 94% when using the set priority socket on the trusted nodes. There is a 5% increase in the ratio. The packet delivery ratio is similar to scenario 1 and scenario 2 when using a set priority socket on the trusted nodes.
4.2. Analysis
The average network delay can be reduced by blocking the incoming packet from interfering nodes, which means the Ground Control Station only receives packets that come from the socket that is prioritized. Hence the time needed to receive all incoming packets is reduced because the total accepted packets are also reduced. The packet delivery ratio is also increased when the priority method is applied. This might happen because the optimum rate is reached when using the priority method. When a socket is given a priority means that it will be prioritized so other incoming packets from the socket other than the priority socket will be blocked. There are 6 socket priority types in NS-3, i.e. best effort, filler, bulk,
interactive bulk, interactive, and control. In these experiments, we use control to solve the problem of the network delay.

5. Conclusion
We have presented a Socket Priority Module for Unmanned Aerial Vehicle Network using FlyNetSimulator. The goal is to reduce packet delay caused by contending nodes that interfere with the UAV and GCS nodes. Scenario 1 averaging 3.3 ms delay reduction and 3% packet delivery ratio raise; and scenario 2 averaging 5.9ms delay reduction and 5% packet delivery ratio raise. The test shows that there is a 16.2% average delay reduction and a 4% average packet delivery ratio raise of both scenarios. From the experiment, we can conclude that using socket priority module in UAV and GCS node reduced the delay and raise the packet delivery ratio that is caused by contending nodes.

Acknowledgment
This work is supported by the PTUPT Grant of Ministry of Research, Technology, and Higher Education (Kemenristekdikti) of Republic of Indonesia number: NKB-356/UN2.RST/HKP.05.00/2020.

References
[1] Bezmenov V, Safin K and Stepanov S 2020 Application of unmanned aerial vehicles for solving engineering tasks IOP Conf. Series: Materials Science and Engineering 890(1) DOI: 10.1088/1757-899X/890/1/012135
[2] Erdelj M, Natalizio E, Chowdhury K R and Akyildiz I F 2017 Help from the sky: leveraging UAVs for disaster management IEEE Pervasive Computing 16 (1) pp 24-32 DOI: 10.1109/MPRV.2017.11
[3] Sona G, Passoni D, Pinto L, Pagliari D, Masseroni D, Ortuani B and Facchi A 2016 UAV multispectral survey to map soil and crop for precision farming applications Int. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLI-B1 pp 1023–9
[4] Shaikh Z, Baidya S and Levorato M 2018 Robust multi-path communications for UAVs in the urban IoT 2018 IEEE Int. Conf. on Sensing, Communication and Networking (SECON Workshops) pp 1–5
[5] Callegaro D, Baidya S, Ramachandran G S, Krishnamachari B and Levorato M 2019 Information autonomy: self-adaptive information management for edge-assisted autonomous UAV systems MILCOM 2019 - 2019 IEEE Military Communications Conf. pp 40–5
[6] Callegaro D, Baidya S and Levorato M 2020 Dynamic distributed computing for infrastructure-assisted autonomous UAVs IEEE Int. Conf. on Communications (IEEE ICC 2020)
[7] Henderson T R, Lacage M, Riley G F, Dowell C and Kopena J 2008 Network simulations with the ns-3 simulator SIGCOMM demonstration 14 14 p 527
[8] Ardupilot 2020 Ardupilot Autopilot Suite Documentation Retrieved on 10 June 2020 from http://ardupilot.com/
[9] Baidya S, Shaikh Z and Levorato M 2018 FlyNetSim: an open source synchronized UAV network simulator based on ns-3 and ardupilot ACM Int. Conf. on Modeling, analysis simulation of wireless and mobile systems (ACM MSWiM 2018)
[10] H-SIM 2020 H-SIM Flight Simulator Retrieved on 10 June 2020 from http://www.h-sim.com
[11] Meier L, Tanskanen P, Heng L, Lee G H, Fraudorfer F and Pollefeys M 2012 PIXHAWK: a micro aerial vehicle design for autonomous flight using onboard computer vision Autonomous Robots 33(1-2) pp 21–39
[12] Navio 2020 NAVIO2 Autopilot Retrieved on 10 June 2020 from https://emlid.com/navio/
[13] ErleRobotic 2020 Erle Robotics Retrieved on 10 June 2020 from https://www.erlerobotics.com
[14] Ardupilot 2020 Ardupilot SITL Retrieved on 10 June 2020 from http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html
[15] Koenig N and Howard A 2004 Design and use paradigms for Gazebo, an open-source multi-robot simulator Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2004) 3 pp
2149–54

[16] Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R and Ng A Y 2009 ROS: an open-source robot operating system ICRA workshop on open source software

[17] Levis P, Lee N, Welsh M and Culler D 2003 TOSSIM: accurate and scalable simulation of entire TinyOS applications Proc. of the 1st Int. Conf. on Embedded networked sensor systems pp 126–37

[18] Chang X 1999 Network simulations with OPNET Proc. of the 31st Conf. on Winter simulation: Simulation—a bridge to the future 1 pp 307–14

[19] Varga A and Hornig R 2008 An overview of the OMNeT++ simulation environment Proc. of the 1st Int. Conf. on Simulation tools and techniques for communications, networks and systems & workshops (ICST 2008)

[20] Parekh C and Patel R M 2017 Encryption and decryption implementation method using network socket programming Int. J. of Advanced Research in Computer Sci. 8(5) pp 2597–604

[21] IBM 2020 IBM Knowledge Center Retrieved on 10 June 2020 from http://ibm.com/support/knowledgecenter