

無線メッシュネットワークのための固定バックオフ時間切替方式のLinux 実装設計

スリツールスタスカリドト[†] 船曳 信生[†] 中西 透[†] 渡邊 寛[†] 田島 滋人^{††}

[†] 岡山大学大学院自然科学研究科 〒700-8530 岡山市北区津島中 3-1-1

^{††} 大阪大学大学院基礎工学研究科 〒560-8531 大阪府豊中市待兼山町 1-3

E-mail: [†]dhoto@sec.cne.okayama-u.ac.jp, ^{††}{funabiki,nakanisi,can}@cne.okayama-u.ac.jp,
^{†††}tajima@ics.es.osaka-u.ac.jp

あらまし 本グループでは、柔軟で安価な大規模インターネットアクセスネットワークとして、無線メッシュネットワーク *WIMNET* の研究を進めている。WIMNET は、互いに無線通信を行う複数のアクセスポイント (AP) で構成され、ホストは通常、AP 間のマルチホップ通信によりインターネットとの通信を行う。WIMNET では、特定リンクが無線帯域を独占し、それ以外のリンクの帯域が不足することで通信性能の劣化が発生するが、その抑制には、通信量に応じて各無線リンクの動作を制御することが有効である。そこで、本グループでは、固定バックオフ時間切替 (*FBS*) 方式の提案を行い、*QualNet* シミュレータを用いたシミュレーションによる評価を行ってきた。本論文では、実機評価を目的として、*FBS* 方式の Linux 実装のための設計を示す。本設計は、Linux カーネルの 5 つのプログラムの修正および実装で構成される。

キーワード 無線メッシュネットワーク、固定バックオフ時間切替 (*FBS*) 方式、リナックス、実装

A Linux Implementation Design of Fixed Backoff-time Switching Method for Wireless Mesh Networks

Sritrusta SUKARIDHOTO[†], Nobuo FUNABIKI[†], Toru NAKANISHI[†], Kan WATANABE[†], and Shigeto

TAJIMA^{††}

[†] Graduate School of Natural Science and Technology, Okayama University

^{††} Graduate School of Engineering Science, Osaka University

E-mail: [†]dhoto@sec.cne.okayama-u.ac.jp, ^{††}{funabiki,nakanisi,can}@cne.okayama-u.ac.jp,
^{†††}tajima@ics.es.osaka-u.ac.jp

Abstract As a flexible and cost-efficient scalable Internet access network, we have studied architectures, protocols, and design optimizations of the *Wireless Internet-access Mesh NETWORK (WIMNET)*. WIMNET is composed of wirelessly connected access points (APs), where any host can basically access to the Internet through multihop communications between APs. In WIMNET, undesirable situations can often happen such that some links dominate the bandwidth while others become insufficient due to the limited shared bandwidth. To improve the performance by avoiding them, controls of activation chances for wireless links depending on their traffics are effective. Thus, previously, we have proposed the *Fixed Backoff-time Switching (FBS) method* for the CSMA/CA protocol, and verified the effectiveness using the *QualNet* simulator. In this paper, we present an implementation design of the *FBS* method in Linux kernel to show its practicality and investigate the performance in a real network. Our design consists of implementations or modifications of the five programs: *Kernel configuration*, *Debugfs*, *Minstrel*, *iw*, and *FBSdaemon*.

Key words Wireless mesh network, fixed backoff-time switching (*FBS*) method, Linux, implementation

1. Introduction

Recently, a *wireless mesh network* has been extensively studied as a promising network technology for a flexible and cost-efficient solution to expand the communication service area by distributing wireless mesh routers on a network field [1] ~ [3]. The mesh routers are connected with each other through multihop wireless communication links using *IEEE 802.11 standards*, in addition to wireless links between client hosts and routers. Then, as a scalable Internet access network based on this technology, we have studied architectures, protocols, and design optimizations of the *Wireless Internet access Mesh Network (WIMNET)* [3]. For a simple architecture, WIMNET is composed of only *access points (APs)* as mesh routers as shown in Figure 1. At least one AP acts as a *GateWay (GW)* to the Internet. Any host in WIMNET can be connected to the Internet through multihop communications between APs and the GW after associated with one neighbor AP.

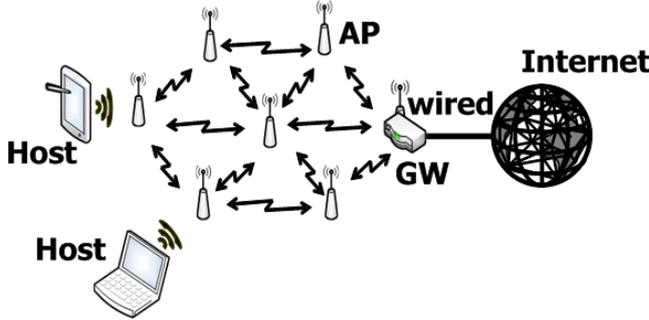


Fig 1: Outline of WIMNET.

WIMNET adopts the commonly used *CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) protocol* of the *IEEE802.11 MAC (Media Access Control)* for the shared communication media access by resolving contentions among interfered wireless links [4]. As illustrated in Figure 2, in this protocol, any node holding a transmission packet is on standby for a constant *DIFS* period and a random time called the *backoff-time* before starting the data frame transmission, to avoid frame collisions among contending nodes while providing their fairness. At each transmission chance, a random value within a size called the *Contention Window (CW)* is selected for the backoff-time. When a node fails in the transmission, the CW size is doubled to reduce the probability of the collision occurrence in the retransmission, which is called the *binary exponential backoff*. When the node succeeds in a transmission, it resets the CW size to the initial one.

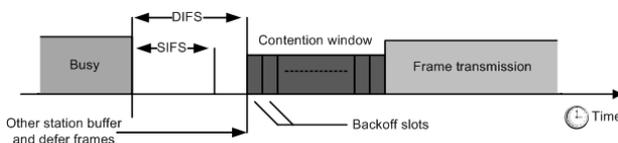


Fig 2: Timing chart for data frame transmission.

Unfortunately, this conventional CSMA/CA protocol is not sufficient for multihop communications in WIMNET. Firstly, heavy congestions of links around the GW can be bottlenecks of whole communications in WIMNET, because these links have to handle a lot of packets to/from the GW for the Internet access. Thus, they should be activated with much higher priorities than other links. Secondly, interferences among these congested links may not be resolved by a random backoff-time in the CSMA/CA protocol because of the limited CW size. Here, we note that the initial CW size is small, and even the maximum CW size is limited. Then, multiple conflicting links can be activated simultaneously by generating the same or similar backoff-times at their source nodes. As a result, any link cannot complete the packet transmission successfully, and needs a retransmission that may cause further conflicts. Hence, using the conventional CSMA/CA protocol, WIMNET may cause a lot of packet losses and intolerable delays, which cannot afford real-time applications such as IP-phones and IP-TVs, although their popularity has been increased with advancements of digital communication technologies.

In order to the abovementioned problem, we have proposed the *Fixed Backoff-time Switching (FBS) method* for the CSMA/CA protocol, and shown its implementation on *QualNet* [5] [6] [10]. *QualNet* [11] adopts a more realistic physical model than other network simulators such as *ns-2* [12].

In the FBS method, the three constant parameters, namely the *target link activation rate*, the *active backoff-time*, and the *passive backoff-time*, must be assigned to every link before starting communications. Here, the *link activation* means that the source node of the corresponding wireless link sends out a signal for a frame transmission. The *target link activation rate* represents the rate of activating the corresponding link that is necessary to handle the link traffics properly. The *active backoff-time* represents a shorter waiting time for the link to be activated preferentially when it holds packets for transmissions. The *passive backoff-time* represents a longer waiting time for the link to be activated only if the contending links using the active backoff-time are not activated, where a larger value than any active backoff-time is used. Besides, for any backoff-time of any link, a different value is assigned from each other to avoid simultaneous link activations as best as possible, and the magnitude follows the descending order of expected traffic loads of links so that congested links can be activated more frequently.

During communications, the *actual link activation rate* is observed by counting the numbers of link activation chances and actually activated times for each link, and taking their fraction. If this value is smaller than the target activation rate, the active backoff-time is selected for the preferential activation of the link. Otherwise, the passive backoff-time is selected. Because different values are assigned to them, contentions among interfered links are expected to be resolved.

In this paper, we present an implementation design of the FBS

method on a Linux PC to evaluate the performance in real networks. As an open source operating system, Linux has been used as a platform to implement new protocols, methods, and devices for advancements of wireless networks including wireless mesh networks [13]- [16]. Our implementation design consists of implementations or modifications of the five programs: *Kernel configuration*, *Debugfs*, *Minstrel*, *iw*, and *FBSdaemon*.

The rest of this paper is organized as follows: Section 2. reviews the FBS method. Section 3. presents our Linux implementation design of the FBS method. Section 4. concludes this paper with some future works.

2. Review of FBS Method

In this section, we briefly review the FBS method for WIMNET.

2.1 Overview of FBS Method

The FBS method uses the *active backoff-time* and the *passive backoff-time* for each link, and selects either of them as a backoff-time at a frame transmission by comparing the *target link activation rate* and the *actual link activation rate*. Any backoff-time is assigned a different fixed value from each other so that no pair of the conflicting links may be activated simultaneously. Besides, the backoff-time for a link with larger traffic is assigned a smaller value than that for a link with smaller one, so that congested links can be activated preferentially. Furthermore, any active backoff-time is assigned a smaller value than a passive one, so that links using active ones have higher priorities in activations than links using passive ones.

During communications, every time a node holding packets detects that the channel for transmissions becomes free, it updates both the target activation rate and the actual activation rate. If the actual one is smaller than the target one, it selects the active backoff-time to let the link be activated, because the current activation rate of the link is not sufficient to handle its traffic. On the other hand, if it is larger, it selects the passive backoff-time to let other links with active backoff-times be activated with higher priorities. A link with the passive backoff-time can be activated only if any conflicting link with the active backoff-time does not hold packets. The following subsections describe how to calculate the three parameters in the FBS method.

2.2 Target Link Activation Rate

For a wireless link l_{ij} transmitting packets from AP_i to AP_j where $i = 1, \dots, N$ and $j = 1, \dots, N$, the target link activation rate rt_{ij} can be calculated by:

$$rt_{ij} = \frac{tm_{ij}}{an_{ij}} \quad (1)$$

where tm_{ij} represents the target number of activating link l_{ij} per second, and an_{ij} does the average number of link activations per second. tm_{ij} can be given from the requested bit rate by:

$$tm_{ij} = \frac{rb_{ij}}{fb_{ij}} \times (1 + fe_{ij}) \quad (2)$$

where rb_{ij} represents the number of bits per second that link l_{ij} needs to be transmitted, fb_{ij} does the average number of bits in one transmitted frame, and fe_{ij} does the rate of causing the frame transmission error. an_{ij} can be given by:

$$an_{ij} = \frac{1}{ft_{ij}} \quad (3)$$

where ft_{ij} represents the average duration time of one frame transmission.

Among the parameters for the target link activation rate, rb_{ij} should be calculated by taking the summation of the bit rates requested by the applications using link l_{ij} in the routing path of WIMNET. The others, fb_{ij} , fe_{ij} , and ft_{ij} , should be updated during communications by the following equations:

$$fb_{ij} = \frac{sb_{ij}}{sf_{ij}} \quad (4)$$

$$fe_{ij} = \frac{ff_{ij}}{sf_{ij} + ff_{ij}} \quad (5)$$

$$ft_{ij} = \frac{t}{sf_{ij} + ff_{ij} + of_{ij}} \quad (6)$$

where sb_{ij} , sf_{ij} , ff_{ij} , and of_{ij} represent the total number of successfully transmitted bits by link l_{ij} , the total number of successfully transmitted frames, the total number of failed frames, and the total number of transmitted frames of the interfered links with link l_{ij} , when t seconds have passed since the communication started in WIMNET, respectively.

2.3 Actual Link Activation Rate

The *actual link activation rate* ra_{ij} for link l_{ij} is obtained by dividing the number of successfully transmitted frames with the number of possibly activating chances for the link:

$$ra_{ij} = \frac{sf_{ij}}{ac_{ij}} \quad (7)$$

where ac_{ij} represents the number of possibly activating chances of link l_{ij} .

In the CSMA/CA protocol, ac_{ij} is hard to be obtained. Unlike the TDMA protocol where the link activations are synchronized by a single clock, the timing of counting the number of activating chances is not clear in the CSMA/CA protocol. Besides, the link activation chances resulting in transmission failures must be considered. In this paper, ac_{ij} is counted every time AP_i detects that the channel becomes free.

2.4 Active/Passive Backoff-time

The *active backoff-time* ta_{ij}^m and the *passive backoff-time* tp_{ij}^m for link l_{ij} are calculated by the following procedure, where m represents the number of consecutively failed transmissions (or retry counter) due to heavy traffics and is saturated by 6. These backoff-times are updated every time the routing path is changed due to the

topology change by adding a new AP or removing an existing AP and the host distribution change by the host join or leave to WIM-NET. Then, they are fixed during communications.

(1) Calculate the number of bits to be transmitted per second rb_{ij} for link l_{ij} by taking the summation of the bit rates for all the communication requests by the hosts using l_{ij} :

$$rb_{ij} = \sum_{k \in H_{ij}} hr_k \quad (8)$$

where H_{ij} represents the set of the host indices using link l_{ij} in the routing path, and hr_k does the requested bit rate (bps) of host k .

(2) Sort every link in descending order of rb_{ij} , where the tiebreak is resolved by the number of hosts using this link for the routing path.

(3) Set this sorted order to the link priority p_{ij} for l_{ij} .

(4) Calculate the active/passive backoff-times for l_{ij} :

$$\begin{aligned} tamin_{ij}^m &= CW_{\min} \cdot \left(2^{m-1} + 2^{m-2} \cdot \frac{p_{ij}-1}{P} \right), \\ tamax_{ij}^m &= CW_{\min} \cdot \left(2^{m-1} + 2^{m-2} \cdot \frac{p_{ij}}{P} \right), \\ ta_{ij}^m &= \text{rand} \left[tamin_{ij}^m, tamax_{ij}^m \right], \end{aligned} \quad (9)$$

where $tamin_{ij}^m$ and $tamax_{ij}^m$ represent the minimum and maximum values for the active backoff-time for l_{ij} when the retry counter is m , CW_{\min} does the initial CW size, and P does the largest priority among the links. In our simulations, $CW_{\min} = 31$ is used in any case.

$$\begin{aligned} tpmin_{ij}^m &= CW_{\min} \cdot \left(2^{m-1} + 2^{m-2} \cdot \frac{P+p_{ij}-1}{P} \right), \\ tpmax_{ij}^m &= CW_{\min} \cdot \left(2^{m-1} + 2^{m-2} \cdot \frac{P+p_{ij}}{P} \right), \\ tp_{ij}^m &= \text{rand} \left[tpmin_{ij}^m, tpmax_{ij}^m \right]. \end{aligned} \quad (10)$$

where $tpmin_{ij}^m$ and $tpmax_{ij}^m$ represent the minimum and maximum values for the passive backoff-time for l_{ij} when the retry counter is m .

3. Design for Linux Implementation of FBS Method

In this section, we present our design for Linux implementation of the FBS method. For convenience, we call a Linux PC implementing the FBS method a *Linux-FBS* in this paper.

3.1 Overview

Basically, in this design for a *Linux-FBS*, we collect the necessary information from the statistics in the devices, to calculate the fixed back-off time in the FBS method, and assign its calculated value to *AIFS* for use as the actual backoff-time in the network device with $CW_{\min} = CW_{\max} = 0$, as shown in Figure 3.

For our implementation design of the FBS method in Linux kernel, we have considered implementations or modifications of the following five programs.

- *Kernel configuration* is modified to activate *Debugfs* and *Minstrel*.
- *Debugfs* is used to obtain the necessary information in the

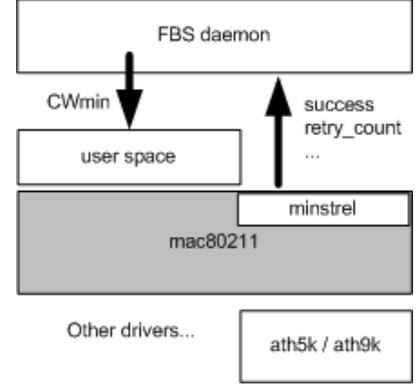


Fig 3: Data Flow for FBS Method in Linux Implementation.

kernel space at the user space through *Minstrel*.

- *Minstrel* is used to obtain the necessary information for the FBS method.
- *iw* is modified to allow the assignment of a specified value (fixed backoff-time) to CW_{\min} .
- *FBSdaemon* is newly implemented as a daemon application to calculate the target/active link activation rates and select the fixed back-off time by comparing them as the main procedure of the FBS method.

3.2 Kernel configuration

For our *Linux-FBS*, we need to activate some features in Linux kernel configurations such as *Debugfs* [17] and *Minstrel* [18] that are used for wireless networks. Therefore, we set up the configuration of the Linux kernel as follows:

```
CONFIG_DEBUG_FS=y
CONFIG_DEBUG_KERNEL=y
CONFIG_WIRELESS=y
CONFIG_CFG8011=m
CONFIG_CFG80211_DEBUGFS=y
CONFIG_LIB80211=m
CONFIG_LIB80211_DEBUG=y
CONFIG_MAC80211=m
CONFIG_MAC80211_RC_MINSTREL=y
CONFIG_MAC80211_RC_MINSTREL_HT=y
CONFIG_MAC80211_RC_DEFAULT_MINSTREL=y
CONFIG_MAC80211_RC_DEFAULT="minstrel_ht"
CONFIG_MAC80211_DEBUGFS=y
```

For the wireless drivers for our implementation, we set up the configuration of the Linux kernel as follows:

```
CONFIG_ATH_COMMON=m
CONFIG_ATH_DEBUG=y
CONFIG_ATH5K_DEBUG=y
CONFIG_ATH9K_DEBUGFS=y
CONFIG_ATH9K_HTC_DEBUGFS=y
```

3.3 Debugfs

Debugfs is a special file system available in a Linux kernel. It is technically referred as a *kernel space-user-space interface*, and is a

simple RAM-based file system that is designed for debugging the kernel. *Debugfs* allows a kernel developer to make information in the kernel space available in the user space. To compile a Linux kernel with *Debugfs*, we need to set `CONFIG_DEBUG_FS` option *yes*. Then, we need to mount *Debugfs* with the following command:

```
mount -t debugfs none /sys/kernel/debug
```

3.4 Minstrel

Minstrel is a *mac80211* rate control algorithm ported over from *MadWifi* that supports multiple rate retries. *Minstrel* has been claimed to be one of the best rate control algorithms. *Minstrel* provides the success/failure information, the actual data rate communication, and the status of interface.

After mounting *Debugfs*, we can use *Minstrel* from a subdirectory of *Debugfs*. Inside the directory of `/sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations`, subdirectories exist where each subdirectory corresponds to each wireless node in the network such as a host (client PC) that is associated with the *Linux-FBS*. The name of a subdirectory is the MAC address of the associated node. For example, `/sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:22:cf:72:21:22/` represents a subdirectory corresponding to a node whose *mac address* is `00:22:cf:72:21:22`. Inside of this subdirectory, we can find the files of the minstrel information for this node.

For the FBS method, we use the following files from *Minstrel*: `rc_stats`, `tx_bytes`, `tx_packets`, `tx_retry_count`, and `tx_retry_failed`.

From the `tx_packets` file, we can get the value for sf_{ij} (the total number of successfully transmitted frames of link l_{ij}). From the `rc_stats` file, we can get the value for ac_{ij} (the number of possibly activating chances) from the `attemp` value. Then, we can calculate the value for ra_{ij} (actual link activation rate). Also, from this file, we can get the value for rb_{ij} (the number of bits to be transmitted per second for link l_{ij}) from the `throughput` value.

Then, we can get the value for sb_{ij} (the total number of successfully transmitted bits by link l_{ij}) from the `tx_bytes` file, the value for of_{ij} (the total number of transmitted frames of the interfered links with link l_{ij}) from the `tx_retry_count` file, and the value for ff_{ij} (the total number of failed frames) from the `tx_retry_failed` file, respectively. Then, we can calculate the value for ta_{ij} (target link activation rate).

3.4.1 Modification of *iw*

iw [19] is a new *nl80211* based *CLI (Command Line Interface)* configuration utility for wireless devices. *nl80211* is a new IEEE 802.11 netlink interface public header. *iw* supports most of the new drivers that have been recently added to the Linux kernel. In our *Linux-FBS* implementation, we use *iw* to assign the fixed backoff-time in the FBS method by changing the values of the variables for *Wi-Fi Multimedia (WMM)* in *IEEE802.11e*, namely CW_{min} , CW_{max} , *AIFS*, and *TXOP*.

However, a default application of *iw* cannot access to or modify the values for them. Thus, we modified the source code of *iw* so that it is possible. In this source code modification, we use a function in the *hostapd* application so that we can change the values for CW_{min} , CW_{max} , *AIFS*, and *TXOP*. Actually, we add the `handle_txq_params` function in the `phy.c` file to access to `TXQ_PARAMS` in wireless Linux kernel parameters.

Listing 1 shows our modification of the source code for *iw*.

```
static int handled_txq_params(struct nl80211_state *state
    , struct nl_cb *cb, struct nl_msg *msg, int argc,
    char **argv)
{
    __u8 queue, aifs;
    __u16 cwmin, cwmax, txop;

    struct nlattr *txq;

    // Sanity checking
    ...

    queue = strtoul(argv[0], NULL, 10);
    cwmin = strtoul(argv[1], NULL, 10);
    cwmax = strtoul(argv[2], NULL, 10);
    txop = strtoul(argv[3], NULL, 10);
    aifs = strtoul(argv[4], NULL, 10);

    printf("Set TXQ_PARAMS for class[%d] : cwmin=%d cwmax
        =%d txop=%d aifs=%d\n", queue, cwmin, cwmax,
        txop, aifs);

    // Range checking for the access class param
    ...

    txq = nla_nest_start(msg,
        NL80211_ATTR_WIPHY_TXQ_PARAMS);
    if (!txq)
        return -ENOBUFS;

    struct nlattr *tx = nla_nest_start(msg, queue);

    NLA_PUT_U8(msg, NL80211_TXQ_ATTR_QUEUE, queue);
    NLA_PUT_U16(msg, NL80211_TXQ_ATTR_CWMIN, cwmin);
    NLA_PUT_U16(msg, NL80211_TXQ_ATTR_CWMAX, cwmax);
    NLA_PUT_U16(msg, NL80211_TXQ_ATTR_TXOP, txop);
    NLA_PUT_U8(msg, NL80211_TXQ_ATTR_AIFS, aifs);

    nla_nest_end(msg, tx);

    nla_nest_end(msg, txq);

    return 0;

nla_put_failure:
    return -ENOBUFS;
}
COMMAND(set, txq_params, "<access_class> <cwmin> <cwmax>
    <txop> <aifs>", NL80211_CMD_SET_WIPHY, 0, CIB_PHY,
    handle_txq_params, "Set TXQ_PARAMS with Queue, CWmin
    , CWmax, TXOP, AIFS\n");
```

Listing 1: "IW modification in phy.c"

Using the `iw phy0 set txq-params 0 0 0 0 10` command, we set the values of the WMM variables for *class 0* (Best Effort), such that $CW_{min} = CW_{max} = TXOP = 0$, and $AIFS = 10$ if the selected fixed backoff-time in the FBS method is 10 for this link.

3.5 FBSdaemon

We implement the procedure for the FBS method by generating a *daemon* application using *Perl*. In this paper, we call this application *FBSdaemon*.

The main cycle for the backoff-time control for the FBS method in *FBSdaemon* consists of the four steps: 1) reading the necessary files from *Minstrel*, 2) calculating both the target and active link activation rates, 3) selecting the fixed back-off time by comparing the both rates, and 4) assigning the selected fixed back-off time to *AIFS* by using the syntax *system* and calling the modified *iw* application. Besides, *FBSdaemon* can give a log report, and can run in the background.

Algorithm 1 shows this procedure in *FBSdaemon*.

```

input : Minstrel files: rc_stats, tx_bytes, tx_packets, tx_retry_count,
        tx_retry_failed
        BO file
output: AIFS
Perl initialization for Daemon, Log, Files;
begin
    Log start;
    Daemonize;
    for (;) do
        acij ← read(rc_stats, attemp);
        sfij ← read(rc_stats, success);
        sbij ← read(tx_bytes);
        sfij ← read(tx_packets);
        ofij ← read(tx_retry_count);
        ffij ← read(tx_retry_failed);
        active ← read(bofile, active);
        passive ← read(bofile, passive);
        Calculate raij, taij;
        if raij < taij then
            | AIFS ← active
        else
            | AIFS ← passive
        end
        system("iw phy0 set txq-params 0 0 0 0 AIFS");
        wait(300s);
    end
end

```

Algorithm 1: FBS Daemon

4. Conclusion

In this paper, we presented a Linux implementation design for the *Fixed Backoff-time Switching (FBS) method* for the CSMA/CA protocol in the *Wireless Internet-access Mesh Network (WIMNET)*.

Our implementation design consists of implementations or modifications of the five programs: *Kernel configuration*, *Debugfs*, *Minstrel*, *iw*, and *FBSdaemon*. In our future works, we will refine the codes of the implementation, generate a testbed with multiple *Linux-APs* implementing the FBS method, and investigate the performance of our proposal in a real network.

Acknowledgment

This work is partially supported by KAKENHI (22500059).

文 献

- [1] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: a survey," *Comput. Netw. ISDN Syst.*, vol. 47, no. 4, pp. 445-487, March 2005.
- [2] Y. Zhang, J. Luo, and H. Hu, *Wireless mesh networking: architectures, protocols and standards*, Auerbach Pub., 2006.
- [3] N. Funabiki edited, *Wireless mesh networks*, InTech - Open Access Pub., 2011, <http://www.intechopen.com/books/show/title/wireless-mesh-networks>.
- [4] Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, IEEE Std. 802.11, 1999.
- [5] S. Tajima, N. Funabiki, and T. Higashino, "A proposal of fixed backoff-time switching method by link activation rate for wireless mesh networks," *Proc. Int. Conf. Complex, Intell., Software Intensive Sys. (CISIS)*, pp. 647-652, 2011.
- [6] N. Funabiki, S. Sukaridhoto, Z. Wang, T. Nakanishi, K. Watanabe, and S. Tajima, "An implementation of fixed backoff-time switching method on IEEE 802.11 MAC protocol for wireless Internet-access mesh network," *Proc. Int. Work. Smart Info-Media Sys. Asia (SISA 2011)*, pp. 67-72, Oct. 2011.
- [7] R. Morris, "TCP behavior with many flows," *Proc. Int. Conf. Netw. Protocols*, pp. 205-211, Oct. 1997.
- [8] A. Vishwanath, V. Sivaraman, and G. N. Rouskas, "Anomalous loss performance for mixed real-time and TCP traffic in routers with very small buffers," *IEEE/ACM Trans. Networking*, vol. 19, no. 4, pp. 933-946, Aug. 2011.
- [9] A. S. Tanenbaum, *Computer Networks*, 5th ed., Prentice Hall, 2010.
- [10] S. Sukaridhoto, N. Funabiki, T. Nakanishi, and K. Watanabe, "A proposal of CSMA fixed backoff-time switching protocol and its Implementation on QualNet simulator for wireless mesh networks," *Proc. WAINA*, March 2012.
- [11] QualNet simulator, Scalable network tech., <http://www.scalable-networks.com>.
- [12] G. A. D. Caro, "Analysis of simulation environments for mobile ad hoc networks," *Tech. Rep.*, no. IDSIA-24-03, Dec. 2003.
- [13] M. Vipin and S. Srikanth, "Analysis of open source drivers for IEEE 802.11 WLANs," *Proc. ICWCSC 2010*, 2010.
- [14] K. Chebrolu and B. Raman, "FRACTEL: a fresh perspective on (rural) mesh networks," *Proc. ACM SIGCOMM NSDR*, Aug. 2007.
- [15] A. Sharma and E. M. Belding, "FreeMAC: implementing a multi-channel TDMA MAC on 802.11 hardware," <http://moment.cs.ucsb.edu/~asharma/freemac-mobisys.pdf>.
- [16] P. Djukic and P. Mohapatra, "Soft-TDMAC: a software TDMA-based MAC over commodity 802.11 hardware," *Proc. INFOCOM*, April 2009.
- [17] Debug - Linux Wireless <http://linuxwireless.org/en/users/Drivers/ath9k/debug>.
- [18] Minstrel - Linux Wireless, <http://linuxwireless.org/en/developers/Documentation/mac80211/RateControl/minstrel/>.
- [19] iw - Linux Wireless <http://wireless.kernel.org/en/users/Documentation/iw>.