Copyright © 2019 Tech Science Press CMC, vol.59, no.3, pp.965-982, 2019

SWRR: The Link Scheduling Algorithm Based on Weighted Round-Robin

Tianbo Lu[1, 2, *], Ru Yan[1, 2], Chao Li[3, *], Lihua Yin[3], Hao Chen[1] and Xieyu Zou[4]

Abstract: With the rapid development of the Internet, people pay more and more attention to the protection of privacy. The second-generation onion routing system Tor is the most commonly used among anonymous communication systems, which can be used to protect user privacy effectively. In recent years, Tor's congestion problem has become the focus of attention, and it can affect Tor's performance even user experience. Firstly, we investigate the causes of Tor network congestion and summarize some link scheduling algorithms proposed in recent years. Then we propose the link scheduling algorithm SWRR based on WRR (Weighted Round Robin). In this process, we design multiple weight functions and compare the performance of these weight functions under different congestion conditions, and the appropriate weight function is selected to be used in our algorithms based on the experiment results. Finally, we also compare the performance of SWRR with other link scheduling algorithms under different congestion conditions by experiments, and verify the effectiveness of the algorithm SWRR.

Keywords: Weighted Round-Robin, Tor, congestion control, link scheduling, anonymous communication.

1 Introduction

In recent years, as people pay more and more attention to the protection of privacy and data [Yu, Tian, Qiu et al. (2018)], the number of Tor users is increasing rapidly, but the number of onion routing does not increase proportionally. Overall, the number of onion routing that each user can use is reduced, which makes network congestion more likely to happen. In addition, some P2P download users not only consume a lot of downstream traffic, but also consume a lot of upload traffic, because they may provide services with other P2P users. By analyzing the data traffic of leaving Tor, the number of connections

[1] School of Software Engineering, Beijing University of Posts and Telecommunications, Beijing, 100876, China.

[2] Key Laboratory of Trustworthy Distributed Computing and Service(Beijing University of Posts and Telecommunications), Ministry of Education, Beijing, 100876, China.

[3] Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou, 510006, China.

[4] Department Computer Science and Engineering, The University of Texas at Arlington, Arlington, 76091, USA.

[*] Corresponding Authors: Tianbo Lu. Email: lutb@bupt.edu.cn; Chao Li. Email: lichao@gzhu.edu.cn.

which is established outside the Tor network built by using the P2P downloading software such as BitTorrent only accounts for 3.33% of the total number of connections, but it consumes about 40.20% of the bandwidth. At the same time, the number of connections established outside the Tor network by using the HTTP protocol accounts for 92.45% of the total number of connections, but it only consumes 57.97% of the bandwidth [McCoy, Bauer, Grunwald et al. (2008)]. It can be seen that a small number of users who conduct P2P download consume a very large bandwidth, so ordinary users may own a long delay.

Tor's congestion problem can affect Tor's performance even user experience [Wang, Chen, Pei et all. (2010)]. Tor's anonymity depends on the number of users, the decrease of users will reduce the anonymity of Tor, and be harmful to the privacy of users, and even increase the possibility of user information being cracked [Tan, Gao, Shi et al. (2018)]. Therefore, Tor's congestion problem is needed to be solved urgently.

Our main contributions in this paper are summarized as follows:

• We propose the link scheduling algorithm SWRR based on WRR. It aims to mitigate network congestion by calculating the corresponding weight for each link and providing them with different QoS (Quality of Service) according to the link weight. We implement the SWRR algorithm in Tor by modifying Tor source code.

• We design three weight calculation functions and evaluate the performance of these three weight functions through comparative experiments in Shadow. Finally, we choose the weight function of linear form to be applied in SWRR.

• We demonstrate the performance of the SWRR algorithm through a set of experiments in Shadow. We evaluate the algorithm SWRR from network throughput, download timeout error, the first byte and a file. Through the analysis of the experimental results, the performance of the SWRR algorithm is better than the other two algorithms when the network congestion is serious. Therefore, the SWRR algorithm can effectively alleviate the congestion of the Tor network.

The remainder of this article is organized as follows. We briefly introduce the related link scheduling algorithm in Section 2. Then the SWRR link scheduling algorithm are described in Section 3. Weight functions used in SWRR algorithm are discussed in Section 4. And the performance of SWRR algorithm is elaborated by comparative experiments in Section 5. Final, we make a conclusion of this paper Section 6.

2 Related work

Scholars attach increasing importance to solve Tor's congestion problems. They have proposed some solutions. For example: UDP-OR [Viecco and Camilo (2008)] and TCP-over-DTLS [Reardon and Goldberg (2008)] proposed in 2008, Gold Star [Dingledine, Roger and Dan (2010)] and Braids [Jansen, Hopper and Kim (2010)] proposed in 2010, Defenestra Tor [AlSabah, Bauer, Goldberg et al. (2011)], Tortoise [Moore, Brad, Wacek et al. (2011)] and Fair Queue [Tschorsch and Scheuermann (2011)] proposed in 2011, DifferTor [AlSabah, Bauer and Goldberg (2012)] and Torchestra [Gopal and Heninger (2012)] proposed in 2012, uTor [Nowlan, Wolinsky and Ford(2013)] and PCTCP [AlSabah and Goldberg (2013)] proposed in 2013, TEARS [Jansen, Miller, Syverson et al. (2013)], IMUX [Geddes, Jansen and Hopper (2014)] and KIST [Jansen, Geddes, Wacek et al. (2014)] proposed in 2014.

The established links that use the same onion route share the same TCP link. Tor provides a corresponding link scheduling algorithm to select the links from which these data can currently be sent. In this way, there are many links on a TCP at the same time, links share the same output buffer, and they will compete for limited output buffer resources when there are multiple links of data of waiting to be sent between two onion routes [Tian, Su, Shi et al. (2018)].

When there are multiple links on a TCP link of waiting to send data, a well-designed link scheduling algorithm can provide different services for applications that consume different bandwidths. There exists a link scheduling algorithm in Tor based on round robin. It is equal for all algorithms to be called for all links. Then some experts and scholars proposed an improved scheme for the scheduling algorithm called the algorithm based on EWMA [Tang and Goldberg (2010)]. Through this algorithm, the priority of links that send less data is elevated and the latency of interactive links is reduced.

2.1 The link scheduling algorithm based on round robin

All active links sharing the same TCP are managed with active link queues. When there are some data units to be transmitted on the link, then the link will be marked as active and then moved to the end of the active link queue. When there exists free space in the output buffer of this TCP, then the link at the head of the active link queue is selected and a unit data of this link is sent by the algorithm. After the data unit is sent, the link will be moved to the end of the active link queue until the next time to be scheduled. If there are N links, assuming that there are an infinite number of data units to be transmitted on these links, the average time for each link to be serviced is T/N for a long period of time T. That is to say, the service time available for each link is the same as them. It can be seen that a relatively fair service is provided by Tor.

2.2 The link scheduling algorithm based on exponentially weighted moving average

This algorithm was proposed by Can Tang and Ian Goldberg in 2010 at the University of Waterloo [Tang and Goldberg (2010)]. Can Tang and Ian Goldberg thought the link which sends Interactive application data should own higher priority. However, except for the export of onion routes, the encrypted onion data unit only can be seen by other onion routes in Tor. Therefore, what application data is sent by the current link cannot be determined by the onion routing node in the middle of the link. Besides, they proposed the concept of exponentially weighted moving average. When the onion route selects the link, the priority of the link should be judged based on the average of the number of onion data units sent by each link in the latest time period.

The total number of data units sent discontinuously by the interactive application is small. The total number of data units sent by the link for downloading is large, and there is almost no time interval between each data unit. If a link owns a higher average value of data units sent over a period of time, it is likely to carry a large file download task, and its priority should be lowered. If a link has a lower average value of data units sent over a period of time, it is likely to carry interactive tasks and should be prioritized. At the same time, the average value of the transmitted data units over time has be reduced over time.

3 The SWRR link scheduling algorithm

3.1 Goals

The ultimate goal of the SWRR link scheduling algorithm is to mitigate network congestion through reasonable link scheduling. We assign different weights to different links through the length of the waiting queue of each link and the time interval during which the link is served, and select these links according to the weight. The design goals of SWRR link scheduling algorithm are mainly discussed as follows:

1. Improve network throughput

Network throughput should be increased by the SWRR link scheduling algorithm, which is the most important method to mitigate network congestion. In the same network configuration, the greater the network throughput of the node, the less the network congestion is.

2. Reduce timeout errors during downloads

In network transmission, if the network is heavily congested, download timeout errors will occur more frequently. The SWRR link scheduling algorithm should be able to reduce timeout errors during downloading by mitigating network congestion.

3. Improve interactive application priority

The SWRR link scheduling algorithm should give different services to different links through reasonable scheduling, improve the priority of the link having the delay sensitive application, and reduce the impact of insensitive data links on transmit delays. The starvation of these links cannot be caused.

3.2 Algorithm description

In the SWRR link scheduling algorithm, two linear tables are used to store the link. The totalist is used to store all active links. The curlist stores all the links that can be selected. And the curlist is a subset of totallist. Usually, the main process of the algorithm is as follows:

(1) Select a link loop in the curlist and record the time point at which the link is selected.

(2) After all the links in the curlist had been select, the link with the lowest weight in the curlist is deleted.

(3) After the deletion is completed, if the length of the curlist is not 0, return to the first step, otherwise go to the fourth step.

(4) Recalculate weights for all active links in the totallist. The current point in time need to be inputted as a parameter in calculating weights. After the link weight is calculated, the link is re-added to the curlist.

(5) Update the weight of all links in the curlist and return to Step 1.

Algorithm 1 is the detailed implementation of the SWRR.

Algorithm 1 SWRR link scheduling

```
1:  Fuction circuit_scheduling(totallist,curlist)
2:   if totallist is empty or curlist is empty then
3:    return -1
5:   end if
6:   while the length of totallist not equal to zero do
7:    foreach circuit in curlist do
8:      circuit.last_served_time ← current time
9:    end do
10:   remove the circuit which has smallest weight
from curlist
11:   if the length of curlist is zero then
12:    foreach circuit in totallist do
13:      Recalculate weight of the circuit
14:      add the circuit to curlist
15:    end do
16:   end if
17:  end do
18:  return 0
```

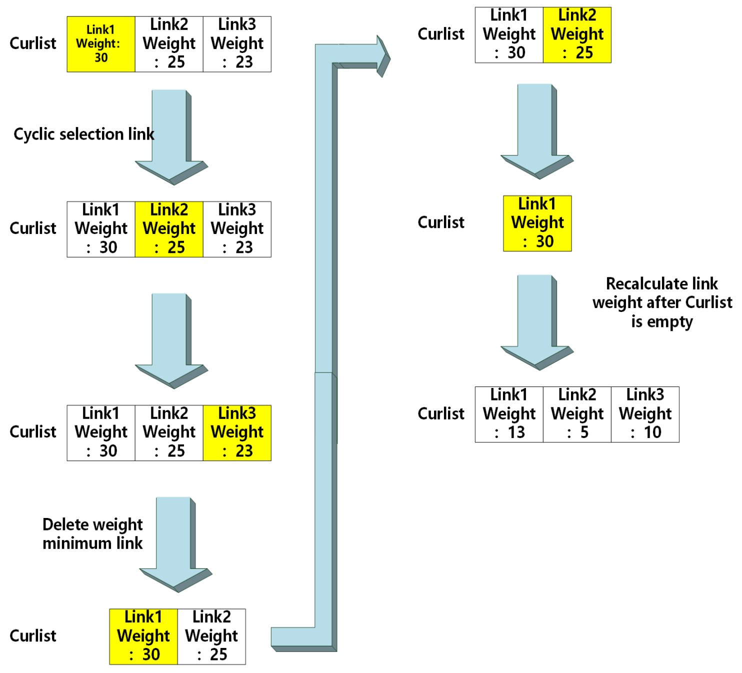

Figure 1: SWRR scheduling algorithm example

We use a doubly linked list for storage. That is to say, we use a doubly linked list and use a tail pointer to represent the selectable range. After each traversal is completed, the tail pointer moves forward to narrow the selectable range. Therefore, we need to sort the links in the doubly linked list from the largest to the smallest, so that the link with the lowest weight in the current linked list pointed to by the tail pointer can be removed. The Fig. 1 is a SWRR scheduling algorithm example.

3.3 Algorithm achievement in Tor

The circuitmux_policy structure in Tor specifies the external call interface that each scheduling algorithm needs to implement, which is defined in the circuitmux.h file. These external call interfaces exist in the form of function pointers, and any scheduling algorithm can selectively implement these interface functions and interact with the upper layer through these functions. In order to implement the SWRR algorithm in Tor, the interfaces we implement are referred to alloc_cmux_data, free_cmux_data, alloc_circ_data, free_circ_data, notify_circ_active, notify_circ_inactive, pick_active_circuit and notify_xmit_cells.

(1) alloc_cmux_data: allocate the space and data required by the scheduling algorithm.

(2) free_cmux_data: release the space allocated by the alloc_cmux_data function.

(3) alloc_circ_data allocate information and space that the algorithm needs to record for each link.

(4) free_circ_data, release the space allocated by the alloc_circ_data function.

(5) notify_circ_active, notify that the status of a link becomes active.

(6) notify_circ_inactive, notify that the status of a link becomes inactive.

(7) notify_xmit_cells, notify that a data unit has been sent for a link.

(8) pick_active_circuit, selects an active link and returns it.

The first four interfaces allocate space and release space for the corresponding structure variables, which will not be described here. We mainly discuss the implementation of the notify_circ_active and notify_xmit_cells.

3.3.1 notify_circ_active

When a link is established, or the link wait queue re-creates a data unit, the link is marked as an active link. Therefore, when the state of a link becomes active, it can be defaulted to have a data unit in its waiting transmission queue. After the module is called, the weight of the active link is calculated, and the doubly linked list is traversed and the new active link is inserted into the first node in the link that is smaller than its weight. The detailed implementation of the interface notify_circ_active is shown in Algorithm 2.

Algorithm2 notify_circuit_active

```
void wrr_notify_circuit_active(wrr_policy* pol, wrr_policy_circ_data*
   cdata){
    //Get current time
    tor_gettimeofday_cached(&now);
    //Calculate the weight of the new node
    set_weight(wrr, &now);
    if (list → size == 0){
        insert_behind(list→ head, newnode);
        list→ cur = newnode;
        list→ cur_tail = list→ tail;
    }else{
  /*Traverse to find the first node whose weight is less than newnode*/
        node* temp = list→ head→ next;
        while(temp != list→ cur_tail){
            if(→weight < newnode→ weight){
                break;
            }
            temp = temp→ next;
        }
        insert_behind(temp→ prev, newnode);
    }
    ++(list→ size);
}
```

3.3.2 notify_xmit_cells

The notify_xmit_cells are called by the upper module after the data unit of the active link is sent. Since we use a doubly linked list to store the link, we need to move the head pointer to send the data of the next active link after the data unit of the active link is sent. If the head and tail pointers are the same after the move, it means that the number of nodes with scheduling is 0. And it also means that the scheduling of this round is over. Therefore, it is necessary to recalculate the weights of all the links in the doubly linked list and reorder. The detailed implementation of the interface notify_xmit_cells is discussed in Algorithm 3.

Algorithm3 notify_xmit_cells

```
void wrr_notify_xmit_cells(wrr_policy* pol,wrr_policy_circ_data* cdata, circuit* circ) {
    linkedlist* list = pol→active_circuit;
    cell_wrr* wrr = &(cdata→wrr);
    //Get current time
    timeval now;
    tor_gettimeofday_cached(&now);
    //Record the time the link was serviced
    wrr→last_served_time = now;
    list→cur = list→cur→next;
    /*When the cur pointer and the cur_tail pointer point to the same node, the cur_tail pointer points to the previous node,and the cur pointer points to the node after the head */
    if(list→cur == list→cur_tail){
            list→cur_tail = list→cur_tail→prev;
            list→cur = list→head→next;
    }
    if(list→cur_tail == list→head→next){
            //Recalculate the weights of all nodes
            reweight_list(list);
            //Reorder all nodes by weight
            sort_list(list);
    //Reset the position of the cur pointer and the cur_tail pointer
            list→cur_tail = list→tail;
            list→cur = list→head→next;
        }
}
```

4 Weight function of SWRR algorithm

4.1 Design

4.1.1 Parameter

There are two parameters in the weight function, the length D of the waiting queue of the link and the time interval T from the last time that the link is served.

The waiting queue of the link refers to the data unit to be sent on the link. The length of the link of waiting queue is related to the application carried by the link. Suppose that there are two links on an onion route, and these two links share the next hop TCP and the applications they carry are different. One of them carries an interactive application and the other carries a large file download application. If the network congestion and bandwidth conditions of the first hop on the two links are the same, the length of the waiting queue for the link carrying the download application will be longer than the waiting queue for the link carrying the interactive application. Because the traffic of the download application is more relative to the traffic of the interactive application and lasts for a long time, the number of data units transmitted by the download application to the onion route will be more than the link of the interactive traffic during the same period of time. One of the purposes of the algorithm is to increase the priority of the link where the interactive traffic is located. That is to increase its weight in the calculation of the weight algorithm. Therefore, in the weight algorithm, we select the link waiting queue length D as a parameter.

This time interval T measures the period in which this link is served. If a link is not serviced for a long time, the weight function should increase the priority of this link in order to prevent the link from being starved for a long time without being served. T is the result that the point time at which the weight algorithm is executed subtracting the point time at which the link was last served.

4.1.2 Function description

The form of the weight function is: Weight = F (T, D).

T represents the waiting time interval in microseconds (1 second=1000,000 microseconds). The waiting time interval is a non-negative number because the time point, at which the link is served last, must be previous to the weight algorithm calculation. As the number of T increases, the time for the link to wait for service increases and the weight of the link increases. That is to say, the link waiting for a long service interval takes precedence, because the specific value of the time interval depends not only on the algorithm itself, but also on network conditions, machine performance, etc. Besides, the upper limit of the value range is not fixed. Suppose that the interval of the time point, when a link was served last and the current time point, is 1 second, then the value of T is 1,000,000. Under the same waiting queue length D, the link with a long time interval Towns a higher priority.

The waiting queue length D unit is a data unit. In theory, D should be a positive integer, but the value of D may be actually 0 when entering the calculation weight function. The macro CELL_QUEUE_HIGHWATER_SIZE is defined in the relay.c file of the Tor source and has a value of 256. This macro is used to limit the length of the waiting queue for the link. When Tor detects that the length of the waiting queue of a link are 256, it stops reading new data from the TCP to which the link belongs, so that the waiting queue length of the link does not exceed 256. Therefore, the waiting queue length D ranges from 0 to 256. In the case of the same time interval, the link with a small queue length owns a higher priority in the algorithm. As D decreases, the weight increases.

According to the above requirements, we have designed three feasible weight function forms:

Weight function in the form of a linear function:

$$F(T,D) = 10000 \times T - D \tag{1}$$

Weight function in the form of a proportional function:

$$F(T,D) = (T+1000)/(D+1000) \tag{2}$$

Weight function in the form of a logarithmic function:

$$F(T,D) = C \times \log_{D+2}(T+1) \tag{3}$$

4.2 Weight function evaluation

We evaluate the performance of the three weight functions from network throughput, download timeout error, time of downloading first byte, time of downloading 327680 bits (we set the size of the single file downloaded by the web user to 327680 bits)

4.2.1 Experiment design

Rob Jansen of the US Naval Laboratory and Nicholas Hopper of the University of Minnesota proposed Shadow in 2012 [Jansen and Hooper (2012)]. Shadow is a discrete event simulator that runs real-world applications in the form of plug-in (Plug-ins) in a simulated environment. A plugin has been written by Shadow for Tor. When some of Tor's functions are executed, these functions will be intercepted by Shadow, such as high-overhead encryption and decryption functions. A customized version of Tor can be accepted by Shadow's Tor plugin, so we implement SWRR algorithm in Tor's source code and simulate the effectiveness of running the verification algorithm through Shadow.

Shadow uses XML files to configure network structure, node parameters, node behavior, etc., and it can simulate the operation of hundreds or thousands of nodes on a single host. For example, the configuration client node requests data from the server node, or configures the server node to listen to requests from the client node on a certain port. Shadow records data for researchers to analyze, such as network throughput and download time during the simulation process.

In the experiment, we set up two kinds of users called the bulk client representing the large file downloader and the web client representing the web browsing user. Bulk client will continuously request a 5MB file from the server. The web client will intermittently request a small file size of 320KB from the server. Shadow's Tor plugin includes a Tor network configuration that includes both of this two kinds of users. In addition, the Tor network configuration also includes the configuration of the ingress route, the middle onion route, and the outlet onion route. We simulate different congestion conditions of the network by adjusting the number of intermediate onion routes.

In order to simulate a congested network environment, we need to configure the information of Tab. 1 in the Shadow configuration file.

In the actual Tor network, the user ratio of the direct connection between onion routing and Tor is 538:1. Therefore, we reduce the number of intermediate onion routes to 20, 10, and 5, corresponding to three levels of network congestion. In addition, in the original network environment configuration, after all the directory servers and onion routes are started, the user browsing the web page is started first, and then the file downloader node is started. In this case, the user browsing the webpage is actually not in a congested network environment. We start the file downloader's earlier to all web browsing nodes. This ensures that when the web browsing node starts running, the network load has already been heavy.

Table 1: Shadow configuration

	Number
servers	100
Tor directory servers	3
exit onion guard node	5
entrance onion guard node	14
exit onion routing node	10
intermediate routing node	20, 10, and 5
bulkclient	80
webclient	320

4.2.2 Experiment results

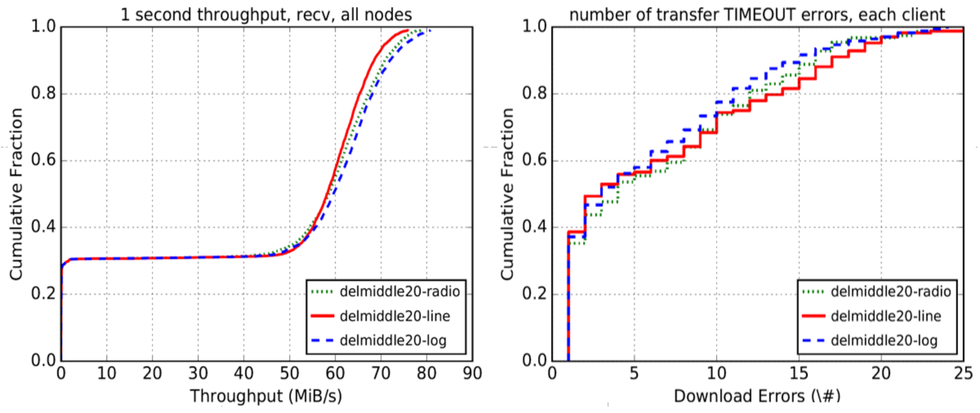

Figure 2: Network throughput and download timeout errors of 20 intermediate onion routing nodes

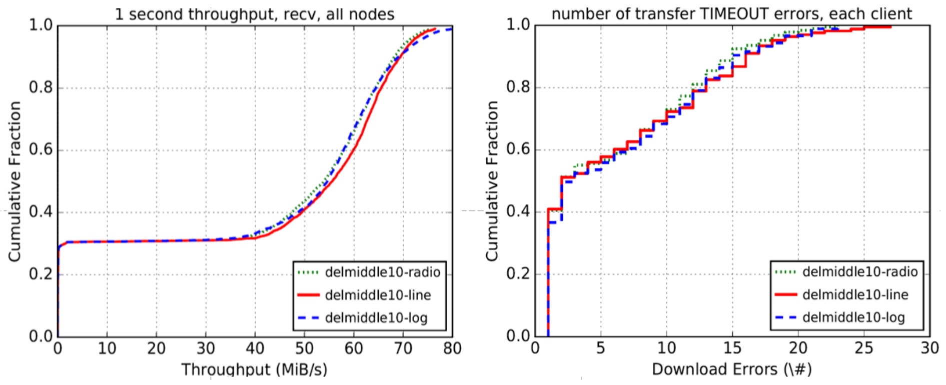

Figure 3: Network throughput and download timeout errors of 10 intermediate onion routing nodes

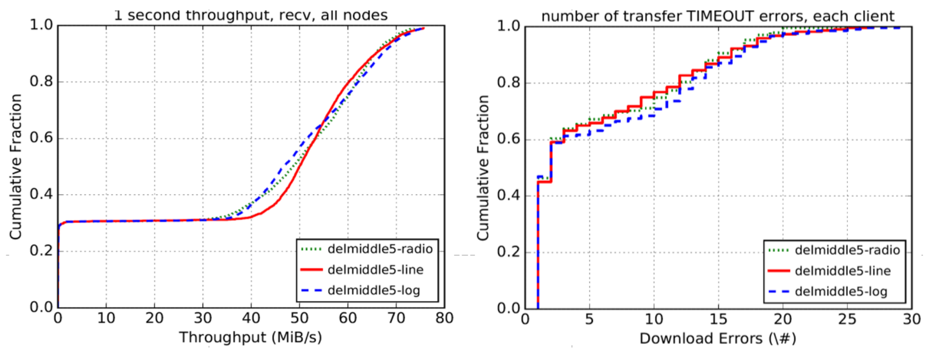

Figure 4: Network throughput and download timeout errors of 5 intermediate onion routing nodes

As can be seen from the above Fig. 2, Fig. 3 and Fig. 4, the network congestion becomes more serious. That is to say, as the number of intermediate onion routing nodes decreases, the overall network throughput reduces gradually, and the number of timeout errors increases at the same time.

When using 20 intermediate onion routing nodes, the linear form of the function shows the lowest network throughput and the proportional form of the function throughput displays middle level, and the logarithmic form of the function network throughput shows the highest. In terms of errors that occur during the download process, there exists the least errors in the logarithmic function, and there are same numerous errors between the linear function and the proportional function.

When using 10 intermediate onion routes, the network throughput of the linear function remains a low level comparing to the proportional and logarithmic functions. The number of downloading timeout errors that occur during transmission is basically same among three functions. When using 5 intermediate onion routes, the network throughput of the linear function begins to exceed the proportional and logarithmic functions. The logarithmic function errors of a download timeout begin to be more than the logarithmic function and the linear function.

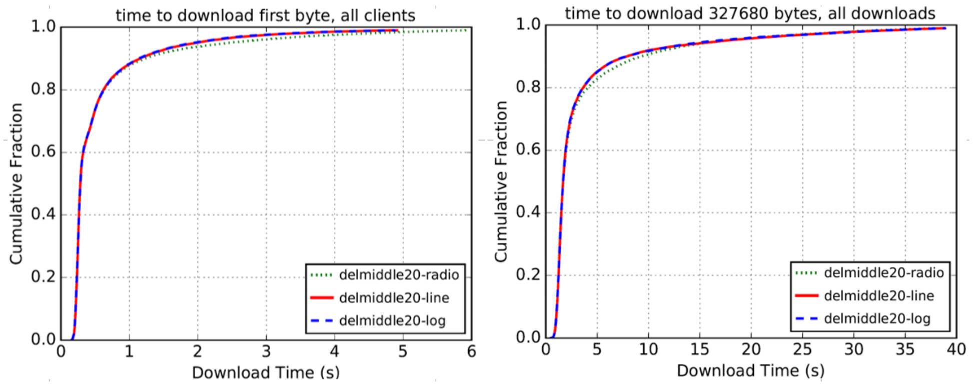

Figure 5: The time of downloading the first byte and 327680 bits of 20 the intermediate onion routing nodes

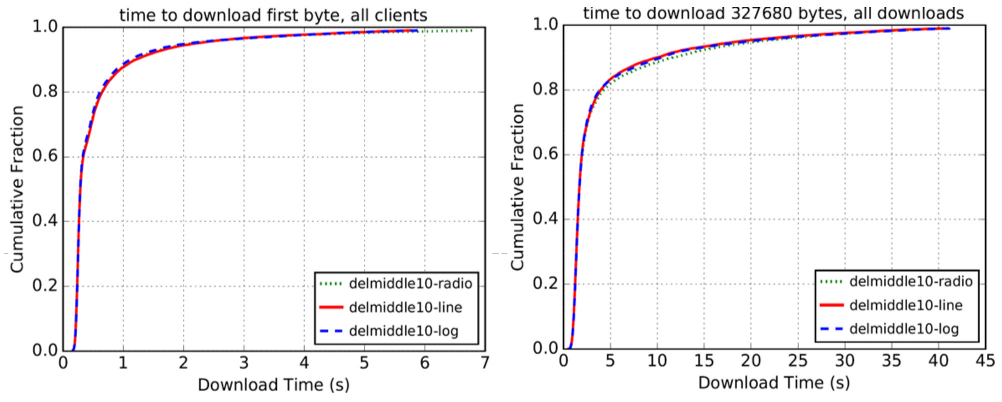

Figure 6: The time to download the first byte and 327680 bits of 10 the intermediate onion routing nodes

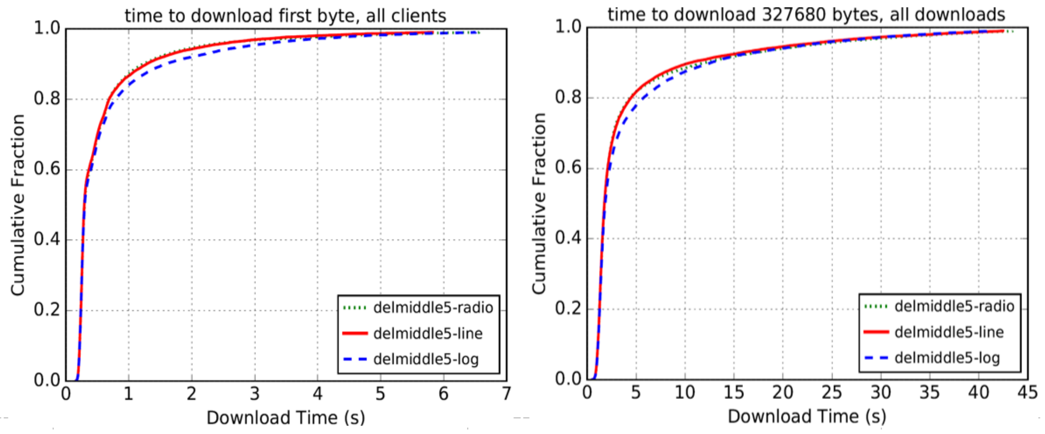

Figure 7: The time of downloading the first byte and 327680 bits of 5 intermediate onion routing nodes

From the above Fig. 5, Fig. 6 and Fig. 7, as the number of intermediate onion routes decreases, the time of downloading first byte and 327,680 bits of the logarithmic function increases with respect to both the proportional function and the linear function.

Through the comparative analysis of these six figures, we can see that the performance of the logarithmic function is gradually becoming poor with the network congestion more severe. However, the performance curve of the linear function shows an upward trend, and the performance of the proportional function is between logarithmic function and linear function. Therefore, we choose the linear function as the weight function to be used in SWRR algorithm.

5 Experiment and evaluation

In this section, the comparison experiments are carried out by using the SWRR link scheduling algorithm, the RR-based link scheduling algorithm and the EWMA-based link scheduling algorithm. We discuss the performance of this three algorithms under different congestion conditions from several aspects, such as network throughput, download timeout errors, time of downloading the first byte, and time of downloading a file.

The experiment is conducted in the same environment as the Section 4.2. In experiment, two versions of the Tor code are used. One is the Tor source, and the other is the Tor code that we modify to achieve the SWRR algorithm. The link scheduling algorithm based on EWMA has been implemented in Tor, but it still needs to modify the Circuit Priority Half-life parameter in the tor file. The RR-based link scheduling algorithm is used in all Tor nodes use by default.

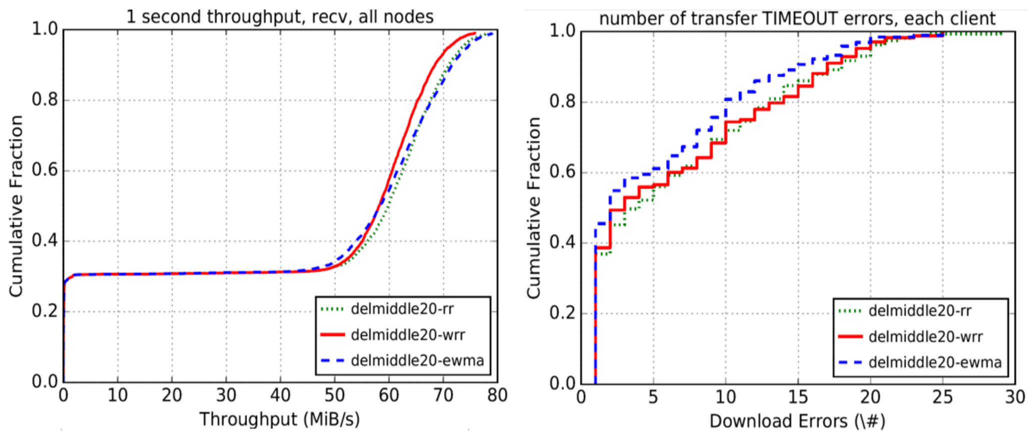

Figure 8: Network throughput and download timeout errors of 20 intermediate onion routing nodes

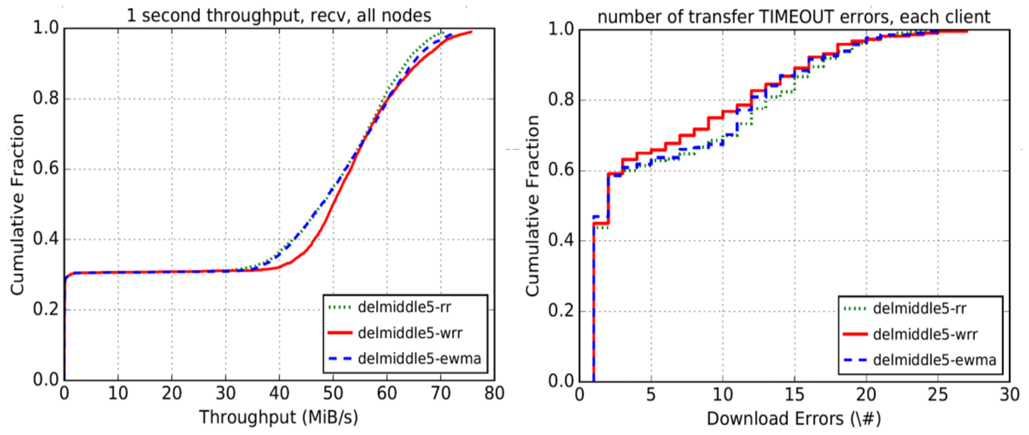

Figure 9: Network throughput and download timeout errors of 10 intermediate onion routing nodes

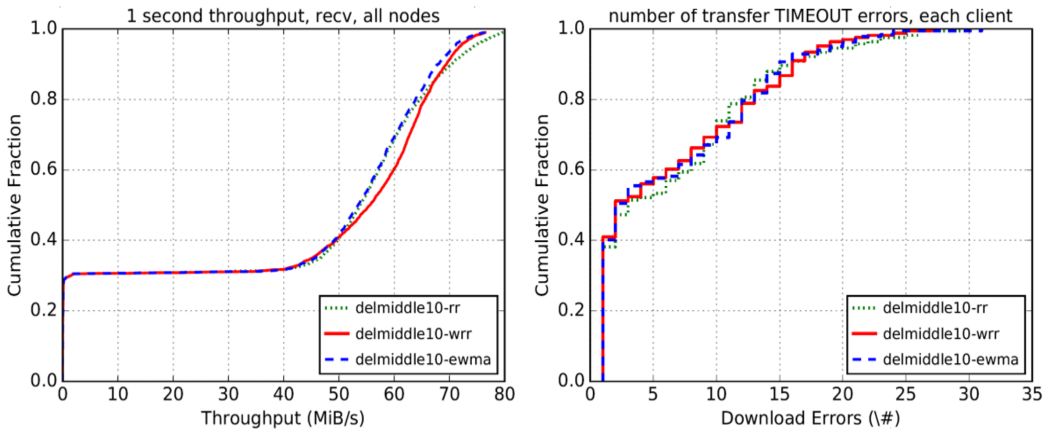

Figure 10: Network throughput and download timeout errors of 5 intermediate onion routing nodes

In Fig. 8, Fig. 9, Fig. 10, the RR curve refers to the RR-based link scheduling algorithm, and the WRR curve refers to the WRR-based link scheduling algorithm SWRR. And the EWMA curve refers to the EWMA-based link scheduling algorithm.

In the Fig. 8, Fig. 9 and Fig. 10, it can be seen that the network throughput of the SWRR algorithm is less than the other two link scheduling algorithms when using 20 intermediate onion routing nodes. That is to say, as the number of intermediate onion routing nodes decreases, the network congestion is worsen, and the network throughput of the SWRR algorithm is gradually beginning to exceed the other two algorithms.

When using 20 intermediate onion routing nodes, there exists the least download error in the EWMA-based link scheduling algorithm, and there is similar performance between the RR-based link scheduling algorithm and the WRR-based link scheduling algorithm. With less intermediate onion routing nodes, the number of downloading timeout errors existing in the SWRR algorithm begins to be less than the other two algorithms.

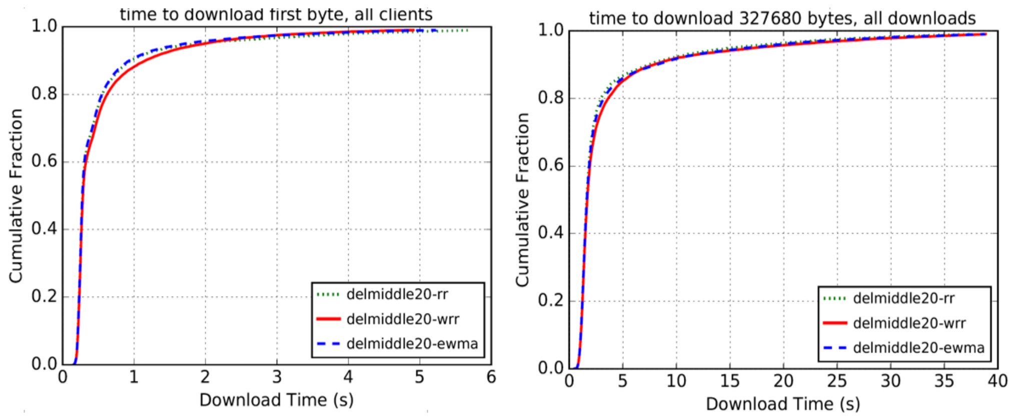

Figure 11: The time of downloading the first byte and 327680 bits of 20 intermediate onion routing nodes

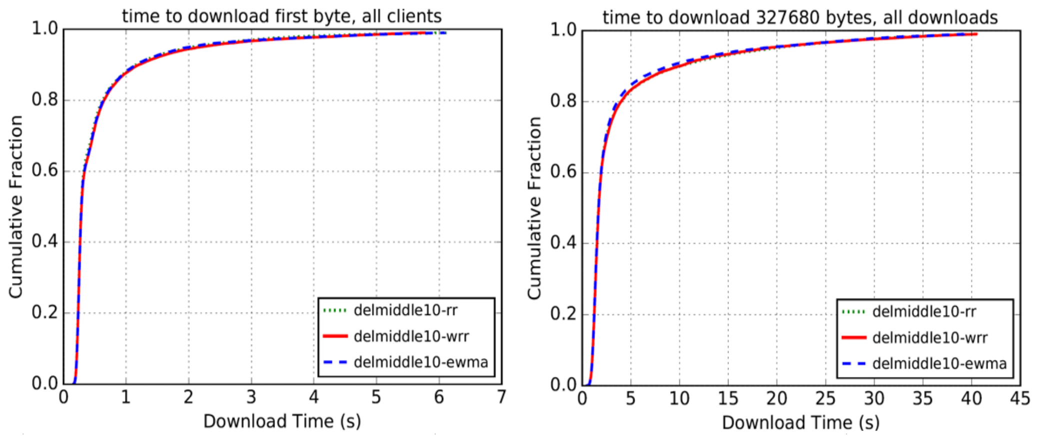

Figure 12: The time of downloading the first byte and 327680 bits of 10 intermediate onion routing nodes

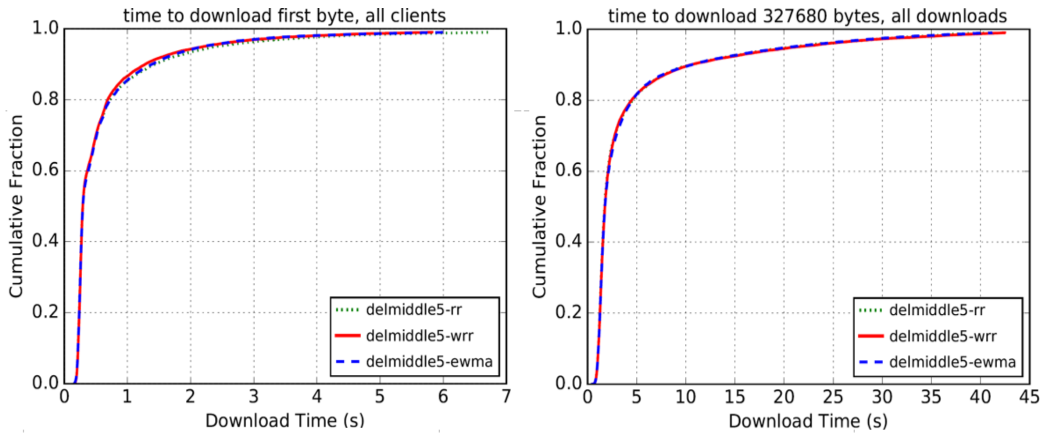

Figure 13: The time of downloading the first byte and 327680 bits of 5 intermediate onion routing nodes

In Fig. 11, Fig. 12, Fig. 13, when using 20 intermediate onion routing nodes, the time of downloading the first byte of the SWRR algorithm is little more than the other two algorithms. As the number of intermediate onion routing nodes decreases, the time of downloading the first byte by using the three algorithms approach to the same value. In terms of the time of downloading 327,680 bits, the three algorithms are used to perform similarly under different congestion conditions.

When the network congestion is heavy, the SWRR algorithm can be used to alleviate network congestion, increase network throughput, reduce the number of timeout errors during transmission and the time of downloading the first byte, and maintain the same performance with the other two algorithms in aspect of time of downloading 327,680 bits. Therefore, the link scheduling algorithm SWRR can effectively alleviate of the Tor network congestion.

6 Conclusion

In the case where there are multiple links sharing the same TCP connection in Tor, it is necessary to provide different quality of service for different links through reasonable link scheduling. In order to alleviate network congestion in Tor, we propose a WRR-based link scheduling algorithm SWRR and implement it in Tor source. Then we design a variety of feasible weight functions and select the optimal weight function for the algorithm through experiments. Besides, we compare the SWRR algorithm with the other two link scheduling algorithms experimentally. Finally, the results show that the SWRR link scheduling algorithm can effectively mitigating network congestion when the network congestion is serious.

Acknowledgement: This work is supported by the National Natural Science Foundation of China (Grant No. 61170273, No. U1536111), and the China Scholarship Council (No. [2013]3050). In addition, we express our sincere gratitude to Lingling Gong, Meng Luo, Zhimin Lin, Peiyuan Li and the anonymous reviewers for their valuable comments and suggestions.

References

Alsabah, M.; Bauer, K.; Goldberg, I. (2012): Enhancing Tor's performance using real-time traffic classification. *Proceedings of the 2012 ACM conference on Reducing Latency in Tor Circuits with Unordered Delivery Computer and Communications Security*, pp. 73-84.

Alsabah, M.; Bauer, K.; Goldberg, I.; Grunwald, D.; McCoy, D. et al. (2011): DefenestraTor: Throwing out windows in Tor. *Proceedings of the International Symposium on Privacy Enhancing Technologies Symposium*, pp. 134-154.

Alsabah, M.; Goldberg, I. (2013): PCTCP: per-circuit TCP-over-IPsec transport for anonymous communication overlay networks. *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pp. 349-360.

Dingledine, R.; Wallach, D. S. (2010): Building incentives into Tor. *International Conference on Financial Cryptography and Data Security*, vol. 6052, pp. 238-256.

Gopal, D.; Heninger, N. (2012): Torchestra: Reducing interactive traffic delays over Tor. *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*, pp. 31-42.

Geddes, J.; Jansen, R.; Hopper, N. IMUX. (2014): Managing tor connections from two to infinity, and beyond. *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pp. 181-190.

Jansen, R.; Geddes, J.; Wacek, C.; Sherr, M.; Syverson, P. F. (2014): Never been KIST: Tor's congestion management blossoms with kernel-informed socket transport. *USENIX Security Symposium*, pp. 127-142.

Jansen, R.; Hooper, N. (2012) Shadow: Running Tor in a box for accurate and efficient experimentation. *Proceedings of the 2012 Network and Distributed System Security Symposium.*

Jansen, R.; Hopper, N.; Kim, Y. (2010): Recruiting new Tor relays with BRAIDS. *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pp. 319-328.

Jansen, R.; Miller, A.; Syverson, P.; Ford, B. (2014): From onions to shallots: rewarding Tor relays with TEARS. *7th Workshop on Hot Topics in Privacy Enhancing Technologies*.

McCoy, D.; Bauer, K.; Grunwald, D.; Kohno, T.; Sicker, D. (2008): Shining light in dark places: understanding the Tor network. *International Symposium on Privacy Enhancing Technologies Symposium*, vol. 5134, pp. 63-76.

Moore, W. B.; Wacek, C.; Sherr, M. (2011): Exploring the potential benefits of expanded rate limiting in tor: Slow and steady wins the race with tortoise. *Proceedings of the 27th Annual Computer Security Applications Conference*, pp. 207-216.

Nowlan, M. F.; Wolinsky, D. I.; Ford, B. (2013): Reducing latency in tor circuits with unordered delivery. *Proceedings of the Symposium on Foundations of Computational Intelligence*.

Reardon, J.; Goldberg, I. (2009): Improving Tor using a TCP-over-DTLS tunnel. *Proceedings of the 18th Conference on USENIX Security Symposium*, pp. 119-134.

Tang, C.; Goldberg, I. (2010): An improved algorithm for Tor circuit scheduling. *Proceedings of the 17th ACM conference on Computer and Communications*, pp. 329-339.

Tan, Q.; Gao, Y.; Shi, J.; Wang, X.; Fang, B. et al. (2018): Towards a Comprehensive Insight into the Eclipse Attacks of Tor Hidden Services. *Access IEEE*, vol. 6, pp. 74854-74684.

Tian, Z.; Su, S.; Shi, W.; Du, X.; Guizani, M. et al. (2019): A data-driven method for future internet route decision modeling. *Future Generation Computer Systems*, vol. 95, pp. 212-220.

Tschorsch, F.; Scheuermann, B. (2011): Tor is unfair-and what to do about it. *Proceedings of 2011 IEEE 36th Conference on Local Computer Networks*, pp. 432-440.

Viecco, C. UDP-OR (2008): A fair onion transport design. *Hot Topics in Privacy Enhancing Technologies*.

Wang, X.; Chen, X.; Pei, Q.; Qu, Y. (2010): Expression preserved face privacy protection based on multi-mode discriminant analysis. *Computers, Materials & Continua*, vol. 57, no. 1, pp. 107-121.

Yu, X.; Tian, Z.; Qiu, J.; Jiang, F. (2018): A data leakage prevention method based on the reduction of confidential and context terms for smart mobile devices. *Wireless Communications and Mobile Computing*.