The ISC Int'l Journal of Information Security

ISeCure

January 2014, Volume 6, Number 1 (pp. 67–76)

http://www.isecure-journal.org

Improving Tor Security against Timing and Traffic Analysis Attacks with Fair Randomization

Asghar Tavakoly [1], and Reza Ebrahimi Atani [1,*]

[1] *University of Guilan, Department of Computer Engineering, P. O. Box 3756, Rasht, Iran*

ARTICLE INFO.

Article history:
Received: 5 March 2014
Revised: 6 October 2014
Accepted: 7 October 2014
Published Online: 26 November 2014

Keywords:
Anonymity, Fairness, Randomness, Timing Attacks, Tor, Traffic Analysis.

ABSTRACT

The Tor network is probably one of the most popular online anonymity systems in the world. It has been built based on the volunteer relays from all around the world. It has a strong scientific basis which is structured very well to work in low latency mode that makes it suitable for tasks such as web browsing. Despite the advantages, the low latency also makes Tor insecure against timing and traffic analysis attacks, which are the most dominant attacks on Tor network in recent past years. In this paper, first all kinds of attacks on Tor network will be classified and then timing and traffic analysis attacks will be described in more details. Then we present a new circuit scheduling for Tor network in order to preserve two properties, fairness and randomness. Both properties are trying to make pattern and timing analysis attacks more difficult and even in some cases impractical. Our scheduler distorts timing patterns and size of packets in a random way (randomness) without imposing artificial delays or paddings (fairness). Finally, by using our new scheduler, one of the most powerful attacks in this area is debilitated, and by it is shown that analyzing traffic patterns and size of packets will be more difficult to manage.

1 Introduction

Tor [1] is a distributed TCP overlay network, which consists of volunteer routers, so-called Onion Routers (OR). Its main goal is preparing anonymity for its users and it is probably the most popular one in this area. It routes data in equally sized cells which are packed by Onion Proxy in user side, along circuits that consist from onion routers. Each router only can add or remove a slice of encrypted information on the way of cells through circuit. The network provides anonymity for its users, Since each router on a circuit only knows its predecessor and successor, which means the last OR (related to destination) is not aware of first OR (connected to user).

In some low latency applications such as web browsing a trade-off is needed between preventing traffic analysis on the flow of user's packets in the network and delivering them in an operant manner. However, Tor deliberately chooses performance and working with low latency versus security. It may make Tor popular, but it cause problem of timing and traffic analysis against global adversaries. It means that if an attacker has the ability to watch all the links between ORs, he/she may be able to link similar patterns to uncover users' activities. Besides the global adversary, which is out of Tor threat model, there are some other attacks, which use time and traffic analysis, inside the model. However, what we present in this paper helps against both of these categories, but we are only in-

* Corresponding author.
Email addresses: atavakoly@msc.guilan.ac.ir (A. Tavakoly), rebrahimi@guilan.ac.ir (R. Ebrahimi Atani)
ISSN: 2008-2045

terested in the second one and we explain the impact of our work on some attacks in this group.

Although there are some old efforts to mitigate timing and pattern analysis in mix systems, but all offered solutions act unfair. As we will see in Section 4 most of these efforts impose additional delays or waste system bandwidth. Using additional delays, or more bandwidth for padding the packets may seem unavoidable, but we present a solution with approximately no overall additional delays or paddings. In this paper, we introduce a new circuit scheduling, named fair randomization and we illustrate it in a simple environment. The results show that the scheduler can make non-recurring and unrecoverable patterns of data, and it can help against timing and pattern based analysis attacks. The important point is that our scheduler tries to play as fair as possible between circuits. It does not bring in any undue delays, and even sometimes in user's point of view it work faster.

Another important point worth to mention is the ability of applying our algorithm at any environment with multiple queues. The main idea is to disarrange the turns of queues. Therefore, it is conceivable, for example in an operating system, for TCP queues or any other network hardware devices with multiple queuing. However, implementing this scenario in real environment needs serious scrutiny and we consider it as our next step in further and future researches.

The rest of the paper is organized as follows: Section 2 describes Tor current circuit scheduling structure. Section 3 contains explanations of some recent attacks on Tor, which used timing and traffic analysis. In Section 4, we review some previous researches to counteract these group of attacks. In Section 5, we explain our new algorithm in details. Section 6 describes the experimental results, and finally the conclusion is drawn in Section 7.

2 Tor's Scheduling Structure and Circuit Management

An Onion Proxy (from computer of users), selects a subset of Tor ORs (typically three ORs namely entry, middle, and exit) and exchanges private keys with them to set up a circuit across them. Then, it encrypts data with those keys as the number of ORs in the circuit. Data travels through the circuit with cells of fixed size (512 bytes). Each OR receives incoming cells, applies the cryptographic operations to unwrap its layer and then forwards the cells along the circuit. It is important to note that TCP connections are common between all circuits that traverse the same pair of onion routers.

On the one hand, Tor's original structure of circuit

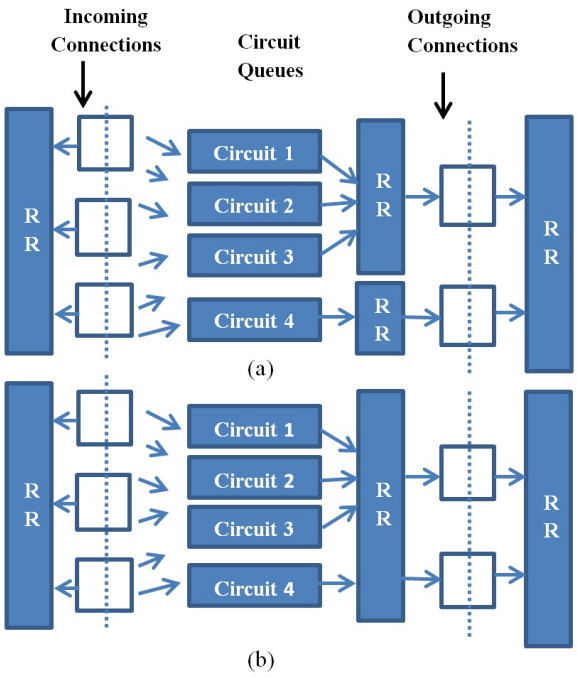

Figure 1. (a) Tor original circuit scheduling. (b) Max-Min circuit scheduling

scheduling is illustrated in Figure 1(a) as it has been in [2]. As the figure shows there is a Round Robin scheduling mechanism in both sides that distributes the maximum bandwidth evenly between incoming and outgoing connections.

After receiving cells from incoming connections, they rearrange in circuit queues. As the circuits traverse a fixed path, all the cells of each circuit forward to one outgoing connection buffer. As it is explained in [2], the mentioned structure is unfair. The unfairness problem is that Tor does not count the number of attached circuits to connections when it is distributing shares. Thus, connections with higher number of circuits get and divide equal part of bandwidth between more circuits and the other connections, as it is clear from Figure 1(a), use the rest of the bandwidth even if they have fewer circuits (like the connection dependent to circuit4 in Figure 1(a), which has only one circuit but it receives equal bandwidth as other connection that is dependent to circuit 1,2 and 3). As an example, suppose that all present bandwidth is A; as Figure 1(a), shows circuit 4 receives A/2 of bandwidth, and each circuit 1,2, and 3 receive only A/6 of the whole bandwidth.

On the other hand, part (b) of Figure 1 demonstrates a fairer scheduling. Both parts of the figure are almost similar, but there is an important difference. In part (b), there is only one Round Robin for all circuits, which is called Max-Min scheduling. It can be seen simply that the new algorithm has a better fair behavior in distributing bandwidth. . In Tor Sched-

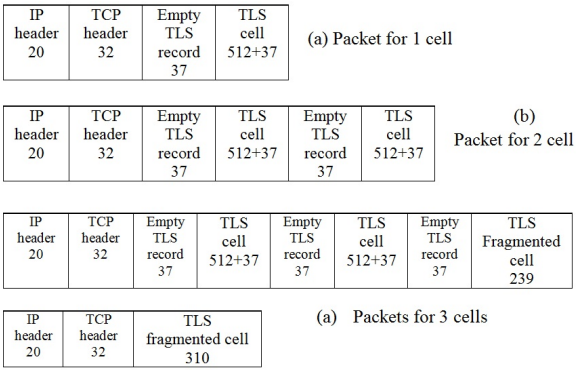

Figure 2. Combining Cells in Packets

uler, incoming cells first will be queued in circuits and then flushed into outgoing connection buffers. Then, cells pack as TCP packets and leave the network. The number of cells in each TCP packet can be different. As it has been illustrated in [3], Figure 2 shows how Tor combines different number of cells in a packet. The MTU is 1500 byte, and when size of the available cells exceeds the MTU, they will be packed in next ones. It is a very important point to have in mind, since different packets probably make different delays.

Circuits share TCP connections with each other in a common path between pair of ORs. Therefore, the buffers of connections are common too, and when cells of common queues are waiting in buffers, they can be complex with each other in TCP packets. It has an effect on the number and size of packets that leave the network and as we explain later, it has been used in some new attacks that we describe some of them in next section.

3 Traffic Analysis Attacks on Tor

Other than traffic analysis and estimation, Tor network is vulnerable to some other attacks. We mention a few example of these attacks. There are attacks based on weakness of Tor protocol's design. For example, there is an attack for Tor authentication procedure in [4]. However, this protocol works fine when running only in one instance, but it does not work properly when it is running in multiple concurrent instances .Other problem based on weakness of Tor protocol, threats users for using the network whenever they want as explained in [5].

Another group of attacks like [6, 7] try to increase the chance of attacker's routers to be chosen. After that, attacker uses selected routers to apply other kinds of attacks. Next attack category use data mining techniques to induce information about network and its users as [8]. It shows Tor network is recognizable by its data transmission behavior in a network. The Last category belongs to Denial-of-Service attack such as [9], that shows the effects of these attacks on Tor's anonymity. This group also cooperates with other ones, almost as prerequisite. It means that, at first attacker disable a router and then observes its effects on the network, "did it tear any circuit off?"

The important attacks in the recent past years on Tor network, belongs to traffic analysis attacks. However, by growing the extent of the Tor, most of them are no longer practical today, but there are still a few practical ones. Traffic and timing analysis is extracting information about the volumes and timing patterns of network packets to infer who is connecting to whom or to where. These attacks categorize in two categories: passive, and active attacks. On the one hand, in passive traffic analysis the attacker records incoming and outgoing traffic of both source and destination parties. Then he tries to analyze the volumes and timing relations between data and seeks for repeated and similar patterns in both sides to infer the correlation between them. These kind of attacks need a long period of observation to reach reasonable results.

On the other hand, active attacks are more powerful and their success rates are higher even with shorter period of running time. In these attacks, the attacker makes a particular pattern like a signal in one side, and then he watches the other side to find the embedded pattern. Since all traffic patterns of connections of Tor are not distorted by each Onion Router through circuits, and also Tor scheduler works in a similar and repetitive fashion monotonously, both kinds of mentioned attacks can be apply on Tor. The two mentioned attributes cause Tor to be predictable and it is the reason to some practical pattern and timing analysis attacks.

We categorize these attacks in two groups and mention few recent samples for them. The first group consists attacks like [10–13]. In these attacks, the attacker tries to estimate and use the RTTs of routers to correlate patterns on different links. The most recent one in this area by [10] is an attack to locate hidden servers in Tor. They build a particular fingerprint of traffic in the user side and then apply the estimated delays of ORs to calculate the output pattern. By watching the fingerprint in the server side, attacker can locate the destination server that is responding to user requests.

The second group of attacks like [3, 14, 15], also imply traffic patterns, but they are more interested in the size of packets instead of delays to find the pattern in destination. In [14] attacker assumes control of both entry and exit router over the victim's circuit. Then exit router manipulates the responding cells of a website to make a particular pattern. To encode 0, the attacker flushes only one cell in the output buffer and makes a TCP packet with the cell. To encode 1, the attacker makes a packet with three cells. After building

the pattern, the entry router counts the number of incoming cells to recognize the pattern. Detecting same sequence at entry router is a confirmation on the correlation between the user and the website. [3] is an attack similar to previous one, of course with making different kind of pattern. It builds pattern on the web server when it is responding to the user and then it tries to find the pattern on user's connection, for example its wireless network. Thus, it does not need to control any OR at all. Therefore, it seems more practical.

Since the attacker does not have control over any OR, he must choose the amount of leaving data from web server to exit OR very precisely. Thus, for making a 0 in the pattern, 498 bytes of data will be sent to exit router, which will be pack exactly in one cell. A little delay assures packing the cell in one TCP packet. Then, for making 1 in the pattern, 2444 bytes of data will be send to exit router. This amount has been chosen very carefully, which forces the OR to pack them in to two TCP packets. The payload part of the first TCP packet will have 1448 bytes of data and the payload of the second packet will be 996 bytes. Making suitable delays assures to receive the exact pattern in other side of the network, means in user's attached network, which shows the relation between the user and the web server.

The other interesting work that depends on size of packets is [15]. They considered size of packets and their results show that Tor makes packets with diagnosable sizes with clear patterns for different protocols. Thus, it used to recognize the Tor network and consequently its users.

Since Tor scheduler acts almost in a clear way, such analyses and attacks can be practical. However, we are not categorizing Tor's attacks in details and the overall impact of our work on them, but results of our work in Section 5 indicate how simply it makes them harder to implement. The important point is that our work has an overall impact on every attack that uses timing and traffic analysis on Tor. In Section 5 we describe how to make non-recurring and unrecognizable patterns with unpredictable size and number of packets for each circuit.

4 Defenses Against Traffic Analysis Attacks in Mix-Networks

In this section we explain some related works in defense against traffic analysis attacks in Mix-Networks, which are a little old but completely relevant. There are also some more recent researches like [16, 17] to mitigate effects of traffic analysis. However, they show the importance of problem, but none of them has been designed for Mix-Networks. For example, in [16] they offered a heuristic traffic reshaping model for wireless networks that scatters the traffic on multiple virtual routes.

Since our priority is traffic analysis attacks in Tor network, it must be mentioned that there is another group of defenses for Tor network that work with morphing methods [18–20]. All of these researches tried to disguise Tor's data to shape of traffic of other software, i.e. Skype [21]. The main reason of these researches is trying to defeat filtering of publicly listed relays. Tor users build circuits with relays and since these relays are publicly visible, blocking them is not so difficult. Consequently, Tor introduced unlisted entry points in the network, known as bridges.

Tor itself also designed "obfsproxy" [22] for the same reason. It passes all the traffic through a stream cipher to reshape it. It helps hiding Tor bridges to survive Deep Packet Inspection (DPI), which has been implemented by some countries to defeat Tor [23]. In the following we explain three old different defenses against traffic and timing analysis attacks, which designed specifically for Mix-Networks.

4.1 Stop-and-Go

First method in [24] as its name implies, tries to stop packets and then leave them to go. It picks random intervals as same number of routers in the circuit that packet must walk through. Then it embeds these numbers inside the packet. Each router stops the packet with respect to associated random number. The method distorts incoming pattern well, but as it is clear, it imposes additional delays on packets.

4.2 Timing Attacks in Low-Latency Mix-Based Systems

In [25] additional padding has been utilized for solving the problem. Their algorithm works in this way:

- First, put additional data as padding, in packets and then send them out.
- In packet's way through routers, each router randomly decides to remove part of additional data or not.
- Finally, last router removes all the remaining additional data.

This algorithm works as well as previous one in distorting the traffic pattern. But as previous one, it reduces system's performance by wasting its bandwidth. However, the amount of paddings may be inconsiderable, but when the number of packets in a system increases, it probably makes a problem.

4.3 Generalizing Mixes

Probabilistic randomization in [26] is another way to make patterns undistinguishable for an attacker. This method works in the following way:

- First, as we explained in Section 2 about Tor's scheduling structure, incoming data will be queued.
- Then for circuits that are common in one TCP connection, every time router tosses a coin and decides randomly to send data of the circuit out or not.

Thus, the outgoing pattern will be completely random. But again, like previous method, this one acts a little bit unfair too. Since for each circuit at every turn, there is only 50 percent chance to send its data, and the algorithm never tries to compensate the loss of the turns, it is not 100 percent fair. However, since the probability is equal for all circuits, we can optimistically assume it plays nearly fair. Besides the optimistic view, which seems to need long periods, it seems that the algorithm is not capable of providing the fairness at least in short periods and with respect to round-trip-times of cells.

5 Fair Random Circuit Scheduling

In this section, we explain how our algorithm works and affects transmitting data. Our algorithm contains two different procedures for two different purposes, (1) makeing delays unpredictable, and (2) changing (or manipulating) size of packets.

5.1 Making Delays Unpredictable

Fair schedulers, like FIFO (First In, First Out) or Round Robin (RR), apply a clear and defined order and arrangement to all of their queues. If all routers use these mechanisms in a network like Tor, repetitive patterns and predictable delays are unavoidable. This predictable structure helps attackers to apply timing and traffic analysis attacks. First point in mind is changing this arrangement to a chaotic and random shape. In order to make these arrangements to have the fairness property, we introduce the concept of a fair random situation, which is a different arrangement of turns form original state. Let's describe fair randomization with an example. In Figure 3, each box shows one turn that belongs to a circuit with the same number. Each vertical row, presents one round of scheduling. Left side of Figure 3 shows how Tor schedules queues (circuits). As it is clear, three queues in FIFO manner being scheduled into a Round Robin mechanism. But in the right side of the figure, we change the arrangement of turns of the queues (circuits) to a fair random shape. It means in the right side of the figure,

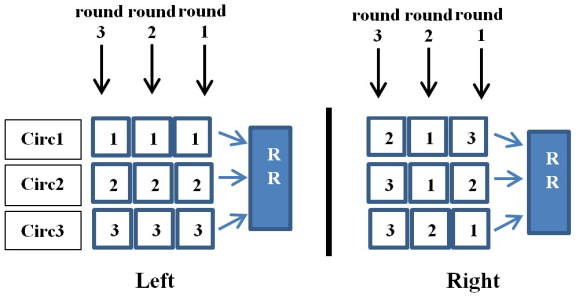

Figure 3. Building a fair random arrangement

we reorder the turns to another arrangement, in exact three rounds same as left side, but we try to make result of turn distribution to be nearly same as left side. As it is clear, first turn of Circuit1 shifted down two steps and the third turn shifted up same amount to make an almost equal overall. It also happens for other queues, of course with other amounts of movements.

There are some important points that worth noting. First, one should not be confusing turn with cell. It means the new scheduler reorders turns of circuits, but it is not important how many cells it is going to pack for any circuit. In fact, some circuits may even be empty when they reach to their turns, which the scheduler simply will leave them and go to next circuits.

Second, the scheduler only tries to reorder the turns of circuits and it is not important that they are empty or not. Finding a new arrangement will be done at the beginning of each session, which number of circuits is a specific number in that moment. If the number of circuits change in middle of a session, for example by entering a new circuit, the algorithm can rearrange again. Also if the OR is not very busy and it has small number of circuits, by picking small round numbers, the algorithm can wait for session to complete before rearranging again . The scheduler works fine even with small round numbers (for more explanation refer to Section 6). However, both solutions seem unfair, but the better solution is specifying a larger number, than the number of available circuits, in beginning of the session. This way we are reserving places for circuits, which will arrive after beginning of scheduling and we can assign empty reserved circuits to them.

From many available fair random arrangements like this, choosing one of them randomly in each session of scheduling makes cell distributing chaotic. However, we show that our algorithm preserves the overall latency with only some ignorable differences and even sometimes offers shorter RTTs.

However, our experiment in next section is very simple, just because of clarity of the concept, but with huge number of queues and different number of rounds

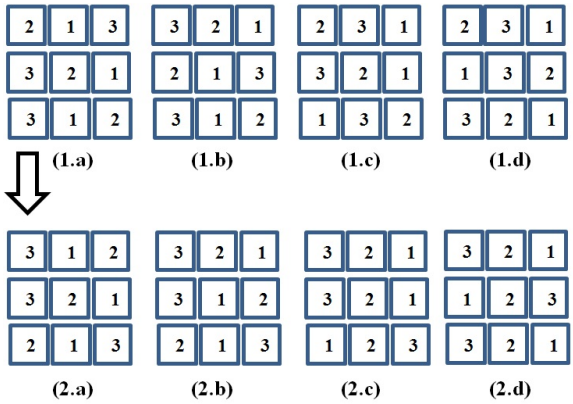

Figure 4. Little part of possible arrangements for new scheduling.

in each session, there are many number of fair random arrangements. The formula below shows the order of this number:

$$((No.Rounds1) * (No.Circuits - 1))^{(Circuits-1)} \quad (1)$$

Simply the formula states that for all turns of a circuit except the last turn ("rounds-1") and for all circuits except the last circuit ("circuit-1"), we can decide about the new places. The place of last turn of each circuit is mandatory to make an equal overall. The last circuit also has no choice, only to fill the blank places. Moreover, for example after finding a new arrangement for circuit1, other circuits, again except the last one, can move between their places (it is why the exponent is "circuit-1"). We give an example in Figure 4, which shows a little part of new arrangement. By Comparing both rows of the Figure 4, for example after placing turns of circuit1 in the situation of (1.a), (2.a) has no changes for circuit1, but the places for circuit2 and circuit3 are different ("circuit-1" in the exponent of the formula).

The whole explanation means in the real world the maximum amount of scattering and disordering results from our algorithm is more than what we illustrated here. It results to more fluctuations in RTTs and thus it brings more distorted patterns.

The following pseudocode shows how to make fair random arrangements. Imagine a two dimensional array[i][j] like Figure 3, for applying the new scheduler. i is the number of circuits and j is a random integer as number of rounds for this session. For circuit1, the algorithm starts with finding a random number between one and the maximum number of circuits, and marks that place in the array for circuit1. Then for next turns, except last one, it puts the turn further between the last marked place and number of queues multiply by current number of round (for example for finding second place the round number is 2). The last one must be in a place that sum of places in new arrangement in the array be equal with sum of places in original arrangement (for real places imagine an array like left side of Figure 3). The algorithm repeats this procedure for all circuits until penultimate one. Last circuit does not need to do anything and it easily fills all blank places in the array.

We note the point that the algorithm is trying to set a limitation, which is twice as the number of circuits, for finding new places in the new arrangement. It is very important since we are not putting any limitation on the number of rounds in the algorithm. Since Tor network is a secure and monitored network, without this limitation, it is like the one who is using the algorithm, is the one who is making the attack. The attack that has been chosen for proving the effectiveness of our algorithm, also does not try to put any unordinary amount of delays on packets. It is because doing so, causes suspicious activity, since it simply shows that there is something wrong with these unordinary delays. Thus, there is not any limitation on the number of rounds when picking a random number for that at each session, because the amount of movements and shifting of turns are under control and limited.

5.2 Making Size of Packets Unpredictable

In the previous section, we reordered arrangement of turns. We show in next section, it makes random and different delays for circuits periodically, which results to different RTTs. Random delays may make packing of cells into TCP packets a little disordered (for example if two turns of a circuit join together, more cells may pack in one TCP packet). It does not seem enough and we want to be sure that no one can easily use number and size of packets to infer any information that breaks anonymity. As we explained in Section 3, some attacks like [3, 14] use patterns of traffic to infer information. Since they act very subtle, we improve our algorithm to make it difficult to apply these attacks.

There is also another problem, which is extensive amount of delays. It happens sometimes specially for primary packets of each circuit and it can make unpleasant experiences for users. If a circuit waits for N turns more than its real turn and the number of all circuits in the router is M, simply the additional amount of delay is (N/M) portion of router's delay. Therefore, we add new cell distribution structure to our scheduler to solve both problems. This procedure only applies to circuits that are common in outgoing TCP connections. Thus, the complete structure of our scheduler acts this way:

- First, we use fair randomization algorithm. Num-

Algorithm 1 Euclid's algorithm

```
1: Random randomNumbers = new Random(); {/*pick a random number for rounds*/}
2: int numberOfRounds = randomNumbers.nextInt();
3: int N = numberOfCircuits;
4: int A[ ][ ] = new int[numberOfCircuits][numberOfRounds];
5: for ( int i from circuit 1 to N-1 ) do
6:    lastPlace = 0;
7:    int temp= randomNumbers.nextInt(); {/*start of first round for each circuit*/} {/*Find a random place
      from 0 to numberOfCircuits */ }
8:    jTemp = j;
9:    temp % = numberOfCircuits; {/* put i in the temp place*/}
10:   A[temp][j] = i;
11:   lastPlace += temp; {/*end of first round for each circuit*/} {/*start of the period of second round to
      penultimate round for each circuit*/}
12:   for  ( numberOfRounds - 2 ) do
13:      int temp= randomNumbers.nextInt(); {/*Find a random place from lastPlace to last-
         Place+(numberOfCircuits* current j ). If it grows more than number of circuits, it means its position
         will be one step further in horizontal direction*/}
14:      temp %= (( j * numberOfCircuits) - lastPlace);
15:      if (temp > numberOfCircuits) then
16:         jTemp = j+1;
17:      end if
18:      A[temp][jTemp] = i;
19:      lastPlace += temp;
20:   end for{/*end of the period of second round to penultimate round for each circuit*/} {/*start of last
      round for each circuit*/} {/*Find a place that sum of new places be equal with old ones*/}
21:   Temp2 = (sumOfRealPlaces - sumOfNewPlaces) % numberOfRounds;
22:   if (Temp2 == 0) then
23:      Temp3 = (sumOfRealPlaces-sumOfNewPlaces) / numberOfRounds;
24:   else
25:      Temp3 =( (sumOfRealPlaces-sumOfNewPlaces) / numberOfRounds) +1;
26:   end if
27:   A[((sumOfRealPlaces-sumOfNewPlaces) % numberOfCircuits)-1][ Temp3 ]= i; {/*end of last round for
      each circuit*/ }
28: end for{/*start of last circuit */}
    {/*Fill all empty places for last circuit*/}
29: for (int i to number of circuits) do
30:   for (int j to number of rounds) do
31:      if (A[i][j] == 0)  then
32:         A[i][j]= last circuit number;
33:      end if
34:   end for
35: end for{/*end of last circuit */}
```

ber of circuits in the real world depends on the OR, but number of rounds for making new arrangement must be random at each session.

- Then we put Round Robin as Figure 1(b).
- Henceforward each circuit shares its turn with other circuits, which are common in the same TCP connection. In this situation, a particular cell managing behavior happens. In this way, the scheduler puts one cell from each circuit up-to-down, and it continues this action until last circuit. Then it reverses the direction from down-to-up. At this time other involved circuits to this sharing behavior, whenever reach to their turns, act same way.

Figure 5 illustrates above explanations. This procedure makes a gap between cells of each circuit inside TCP packets. Assuming that the number of common circuits of TCP connection is n, the cell interval between first and second cell of present circuit would be $2(n-1)$. It is important to note that in each turn, this maximum amount of gap only applies to the circuit that is the owner of the turn and not to the other

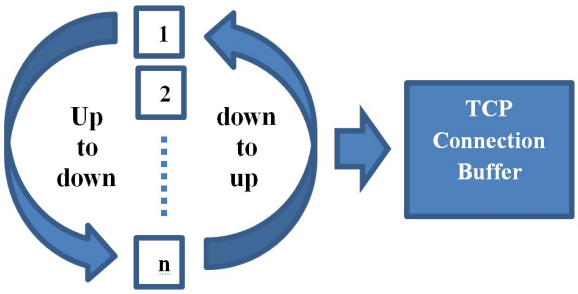

Figure 5. new cell distributing structure

common circuits. For common circuits in next places with respect to owner of the turn, the amount of gap will be different and in a decreasing trend. Since each circuit will have their own turns, thus beside other additional gap that will be provide for them in different places, they will have the maximum available gap at least in their own turns. This way we try to make maximum distance between cells to seclude them from being in same TCP packet. It increases the probability of unequal delays for different TCP packets, and therefore it helps against traffic analysis with respect to number of cells in each packet.

After these steps, the scheduler is on its 100 percent performance and capability, only when multiple circuits are sharing one TCP connection. With respect to calculations has been done in [27] with data gathered in year 2010 about number of ORs and reported bandwidth, by increasing number of users and loads on ORs, there is vast amount of hope on sharing connections between circuits in all ORs. Moreover, our results show for distorting a circuit's pattern, it is enough only when the circuit shares a connection with other circuit(s) only in one OR in his path from entry to exit.

Other important point is that there can be any other constraints or criterion in bandwidth distributing. We suppose and offer, it may be better to give some priority to web users against bulk downloaders in number of turns or in number of cells to flush into TCP connection buffers, as in [27].

6 Experimental Results

To prove effectiveness of our algorithm, we try to counterfeit the attack in [14]. we chose this one, because it has two conditions: 1) it uses two ORs, entry and exit, in a circuit to apply the attack, thus it is a powerful and well-established attack, and 2) there is only one point in the circuit to break the attack, which is the middle router. However, this attack is not practical, since it needs two ORs in user's circuit, which taking control of both entry and exit at the same time is very difficult to do. If assumption of this attack for controlling both ORs happen in the real world, breaking anonymity will be an easy task to do. Thus, since the only helpful position to stop the attack is middle router, which is out of attacker's control, this attack is one of the hardest to break. We show even in this situation, we can stop and unsettle the attack. We only apply our algorithm in one router, middle router, and results prove that the attack no longer is easily applicable. It must also be reminded that if applying the new algorithm to only one router can stop this easy setup attack, it certainly can defeat others, too. Most of other attacks use only one OR, or even they only intercept data between ORs, which the common and most useful places includes OP to entry OR and exit OR to destination. Since, in the mentioned attacks there are more than one position to apply the new algorithm, and also they implemented in very harder situation, the probability of breaking the attacks is even more than what we show here.

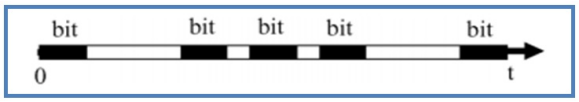

Figure 6. Time-hopping technique [14]

The evaluation platform is planetLab deployment, which uses real Tor relays. The platform is very simple and it consists of an entry router, an exit router and only one modified OR (middle router) which is equipped with the new scheduler. It must be mentioned that before using the modified version, we use a normal OR as middle router to compare the results. The flow of packets of data is sent towards the entry OR with a particular pattern from three OP (making three circuits in all of ORs) with exact timing for each circuit. The pattern is built as exactly as the attack that has been chosen for the testing. All circuits share one outgoing connection. The periods are picked very precisely and not too long, exactly as time-hopping technique that has been explained in [14] (and in Figure 6). It is just for the attack invisibility and can reduce the probability of being recognized as an attack by an OR. It uses a small amount of random intervals between signal bits. These intervals like coding will be used at other side of network to recover the original signals. After applying these requirements in both sides, we observe the results on the other side of the network (exit OR). The only difference between the chosen attack and our experiment is the direction of the procedure, as we move from entry to exit but they did reversely.

We consider the results on the outgoing connection of the exit OR and analyze them to find out the practicality of the attack with respect to estimation of delays and size of packets. We first consider the attack with Tor's original circuit scheduler in this

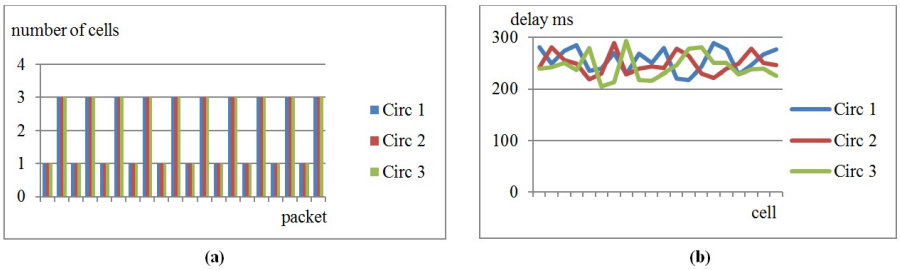

Figure 7. Tor circuit scheduler. (a) Size of packets, (b) Pattern of delays

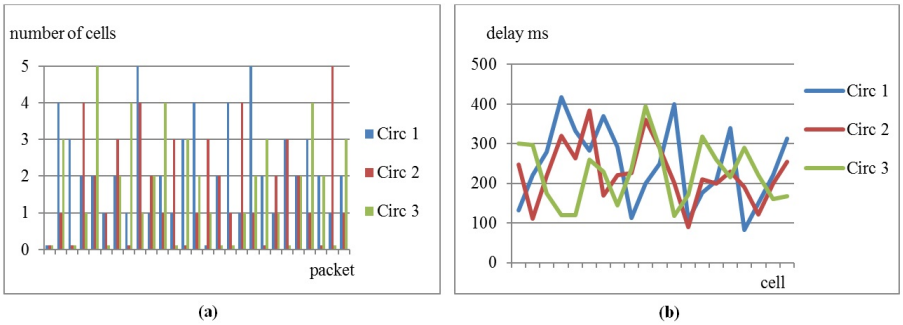

Figure 8. Fair random scheduler. (a) Size of packets, (b) Pattern of delays

environment and then we consider our scheduler with the same input data. Results of our experiment show that the original Tor scheduler preserves the pattern by making predictable delays continuously in an almost fixed range. It also shows that attacker can estimate number of cells per packet easily. Both of the these results are available in Figure 7, which shows only a small part of the flow of data.

However, the second step of the experiment with new scheduler shows different results. As Figure 8(a) shows, on outgoing connection of exit router, the number of cells per packet has considerable fluctuations. It also happens to amount of delays for each packet. As we hoped, it helps to break monotony and predictability and therefore, it makes creating recognizable patterns difficult. When delay of cells changes, the RTTs of them changes, too. This causes an irregular timing for circuits and it makes timing analysis difficult. We applied our scheduler only in one OR on the route of packets from entry to exit router. Passing data from more ORs equipped with new scheduler, can make estimation harder and even impossible.

7 Conclusions

In this paper, we first explained the structure of Tor's circuit scheduler. Then we described some attacks, which use traffic pattern based analysis. Then we presented a new circuit scheduler for Tor with a new mixing strategy. The aim of the scheduler was making different amount of delays for circuits to eventuate unrecognizable patterns of data. It also produces packets with disparate number of cells for each circuit. Both actions help against timing and pattern based analysis attacks. Finally, we tested the new scheduler in a simple environment and results show our new algorithm makes these attacks more difficult. Since the new scheduler makes unpredictable patterns, it makes most of the timing and traffic analysis attacks of Tor network more difficult and in some cases unpractical.

References

[1] R. Dingledine, N. Mathewson and P. Syverson, *Tor: The Second-Generation Onion Router*, In Proceedings of the 13th USENIX Security Symposium, 2004.

[2] F. Tschorsch, B. Scheuermann, *Tor is Unfair and What to Do About It*, 36th Annual IEEE Conference on Local Computer Networks, 2011.

[3] Z. Ling, J. Luo, W. Yu and X. Fu, *Equal-Sized Cells Mean Equal-Sized Packets in Tor?*, IEEE International Conference on Communications (ICC), 2011.

[4] Y. Zhang, *Effective Attacks in the Tor Authentication Protocol*, In Third IEEE International Conference on Network and System Security, 2009.

[5] J. McLachlan and N. Hopper, *On the risks of serving whenever you surf: vulnerabilities in tor's blocking resistance design*, 8th ACM workshop on privacy in the electronic society, 2009.

[6] V. Pappas, E. Athanasopoulos, S. Ioannidis, and E. P. Markato, *Compromising anonymity using packet spinning*, 11th Information Security Conference (ISC), 2008.

[7] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and

D. Sicker, *Low-resource routing attacks against tor*, ACM workshop on Privacy in electronic society, 2007.

[8] J. Barker, P. Hannay and P. Szewczyk, *Using Traffic Analysis to Identify the Second Generation Onion Router*, International Conference on Embedded and Ubiquitous Computing (EUC), 2011.

[9] N. Borisov, G. Danezis, P. Mittal and P. Tabriz, *Denial of Service or Denial of Security?*, 14th ACM conference on Computer and communications security, 2007.

[10] J. A. Elices and F. Perez-Gonzalez, *Fingerprinting a flow of messages to an anonymous Server*, International Workshop on Information Forensics and Security(WIFS), 2012.

[11] N. Evans, R. Dingledine and C. Grotho, *A Practical Congestion Attack on Tor Using Long Paths*, In Proceedings of the 18th USENIX Security Symposium, 2009.

[12] N. Hopper, E. Y. Vasserman and E. Chan-Tin, *How much anonymity does network latency leak?*, ACM Transactions on Information and System Security, vol.13, 2010.

[13] S. J. Murdoch and G. Danezis, *Low-cost traffic analysis of Tor*, in Proceedings of the 2005 IEEE Symposium on Security and Privacy, 2005.

[14] Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, W. Jia, D. Xuan and W. Jia, *A new cell counting based attack against Tor*, IEEE/ACM TRANSACTIONS ON NETWORKING, 2012.

[15] J. Barker, P. Hannay and P. Szewczyk, *Using Traffic Analysis to Identify the Second Generation Onion Router*, International Conference on Embedded and Ubiquitous Computing (EUC), 2011.

[16] F. Zhang, W. He and X. Liu, *Defending Against Traffic Analysis in Wireless Networks Through Traffic Reshaping*, 31th International Conference on Distributed Computing Systems, 2011.

[17] S. Chen, R. Wang, X. Wang and K. Zhang, *Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow*, IEEE Symposium on Security and Privacy, 2010.

[18] T. Sochor, *Fuzzy Control of Configuration of Web Anonymization using TOR*, Technological Advances in Electrical and Electronics and Computer Engineering (TAEECE), 2013.

[19] C.V. Wright, S.E. Coull and F. Monrose, *Traffic Morphing: an efficient defense against statistical traffic analysis*, 16th network and distributed system security symposium, 2009.

[20] H. Mohajeri Moghaddam, B. Li, M. Derakhshani and I. Goldberg, *SkypeMorph: protocol obfuscation for Tor bridges*, ACM conference on Computer and communications security (CCS), 2012.

[21] Skype Software. http://skype.com [Online; accessed September 2013]

[22] N. Mathewson. The Tor Project, A simple obfuscating proxy. https://gitweb.torproject.org/obfsproxy.git [Online; accessed September 2013].

[23] P.Winter, S. Lindskog, How the Great Firewall of China is Blocking Tor, Technical Report, www.cs.kau.se/philwint/static/gfc/. [Online; accessed September-2013].

[24] D. Kesdogan, J. Egner and R. Schkes, *Stop-and-go MIXes: Providing probabilistic anonymity in an open system*, Second International Workshop on Information Hiding, Springer, LNCS 1525, 1998

[25] B. N. Levine, M. K. Reiter, C. WANG and M. K. Wright, *Timing attacks in low-latency mix-based systems*, In Proceedings of Financial Cryptography, 2004.

[26] C. Diaz and A. Serjantov, *Generalizing mixes*, In Proceedings of Privacy Enhancing Technologies workshop (PET), 2003.

[27] C. Tang and I. Goldberg, *An improved algorithm for Tor circuit scheduling*, 17th ACM Conference on Computer and Communications Security, 2010.

Asghar Tavakoly received the B.S. degree in computer engineering (software engineering) from University of Kashan, Iran, in 2010, and received the M.S. degree in the same field from university of Guilan, Iran, in 2013. He is currently pursuing the Ph.D. degree in software engineering at Iran University of Science and Technology (IUST). His research interests include anonymity networks and electronic voting protocols.

Reza Ebrahimi Atani studied Electronics Engineering at the University of Guilan, Rasht, Iran and got his B.S. degree in 2002. He followed his masters and Ph.D. studies at Iran University of Science & Technology (IUST) in Tehran, and received the the Ph.D. degree in 2010. He has an assistant professor position in Department of Computer Engineering at the University of Guilan. His research interests include Cryptology, Computer and Network security, Computer Networks and Cryptographic hardware and Embedded systems..