An Active Queue Management based Deterministic Denial of Service Prevention

Saif Bilal[1], Ghulam Abbas[2], Ziaul Haq Abbas[3]
[1,2]Faculty of Computer Sciences and Engineering, GIK Institute of Engineering Sciences and Technology, Topi 23640, Pakistan
[3]Faculty of Electrical Engineering, GIK Institute of Engineering Sciences and Technology, Topi 23640, Pakistan
[1]gcs1417@giki.edu.pk, [2]abbasg@giki.edu.pk, [3]ziaul.h.abbas@giki.edu.pk

***Abstract*— Denial of Service (DoS) attacks are one of the major threats to the security of networks and online servers. Active Queue Management (AQM) is an effective mechanism to prevent DoS attacks at edge routers. However, some DoS flows may have low bit rates, as they do not consume more than a fair share to avoid being detected by an AQM. AQM schemes also fall short of detecting DoS attacks conducted through IP spoofing. This paper proposes a novel AQM scheme, called Deterministic DoS Prevention (DDP) to avoid low-rate DoS attacks on infrastructure and application levels, attacks caused by unresponsive or responsive flows using IP spoofing, and the attacks having high bit rates. The performance of DDP is evaluated in comparison with an eminent AQM based DoS prevention scheme. Simulation results demonstrate the effectiveness of DDP in effectively detecting IP spoofing and filtering malicious flows that orchestrate high-rate and low-rate DoS attacks.**

***Index Terms*—AQM, DoS, Ingress filter, IP Spoofing.**

I. INTRODUCTION

Cyber-attacks are becoming a greater threat as smart devices are getting cheaper and average bandwidth available to individuals is increasing. Among the cyber-attacks, DoS attacks occur more frequently [1], [2]. The purpose of DoS attacks is to make local or remote services unavailable for authentic users through flooding by either using the available online tools or through botnets to send a huge volume of requests. There are two broad categories of DoS attacks, namely, infrastructure-based attacks and application-based attacks [1], [3], [4]. The main difference between these types of attacks is that the former targets the network layer, while the later targets the transport layer. Most common types of infrastructure-based attacks are UDP floods and Domain Name Server (DNS) based floods, while application layer attacks are mainly Hyper Text Transfer Protocol (HTTP) floods [3]. Mostly, flooding attacks have higher bit rates, but the low-rate DoS attacks can evade detection and defense mechanisms designed for the flooding based attacks. There is another type of attack, which differs from flooding on the basis of bit rates. This attack specifically targets protocols on application level and behaves like a legitimate flow due to its low bit rate, which makes its detection difficult if a defense mechanism is only focusing on flooding at the infrastructure level. Moreover, attackers can also hide their identity by means of IP spoofing and, thus, frames an Internet service provider to wrongly penalize the IP address of a legitimate user [2]–[4].

A variety of techniques have been proposed to address the growing challenge of DoS attacks either by means of middle-boxes or through defense mechanisms implemented in servers or intermediate nodes [1]–[3]. Router-based AQM techniques offer an efficient way to detect and mitigate infrastructure level flooding based DoS attacks [1], [5], [6]. However, AQM techniques cannot deal with low-rate DoS flows, which do not consume more than a fair share to avoid being detected. AQM schemes also fall short of detecting DoS attacks conducted through IP spoofing. These limitations impose a need for more advanced AQM techniques to detect IP spoofing and to filter the low-rate DoS attack at network level before it reaches the victim.

This paper proposes a novel AQM approach, namely, Deterministic DoS Prevention (DDP), to overcome the DoS attacks on infrastructure and application levels. The proposed approach classifies incoming flows on the basis of fair-share, bit rate, uplink data transmission and IP spoofing. Users that try to either consume more than their allocated fair share even after ECN notification, or have higher bit rates are classified as the malicious users. DDP also focuses on the users having normal bit rates that are trying to transmit huge data volume on the uplink to execute a low-rate DoS attack by means of bypassing the filter designed for the high bit rate attacks. Moreover, it filters the users that try to hide their identity by spoofing their IP addresses. When a user is classified as malicious, DDP drops every incoming packet from that user. DDP removes a user from the malicious category if it reduces its bit rate, stops consuming more than its allocated fair share, does not spoof IP address and reduces its data upload.

The rest of this paper is organized as follows. Section II presents the related work. Section III presents the proposed DDP scheme. Section IV presents performance evaluation and Section V concludes the paper.

II. RELATED WORK

AQM on the abstract level can be classified into two broad categories [5]. The first category gives each and every incoming flow fairness whenever the traffic is only coming from responsive flows. These type of techniques include random early detection (RED) [7] and AQMRD [8]. The second category ensures fairness when traffic is coming from both responsive and unresponsive flows. Eminent techniques in this category include CHOKe [9], SFB [10], RED-PD [11], PURGE [12], CHOKeH [13] and Deterministic Fair Sharing (DFS) [6]. CHOKe is a modification of RED and is aimed at lowering the rate of unresponsive flows. It make enhancements in the dropping mechanism of RED by matching the flow id's of an incoming packet and a random packet selected from the queue. SFB uses Bloom filter to uniquely classify flows through

multiple independent hash functions for restricting unresponsive flows. The memory is divided into different bins and each bin is kept for one or more flows. On the arrival of each packet, hashing is performed on the basis of flow id for identification of high bandwidth flows. RED-PD uses the RED drop history to identify and restrict high bandwidth flows. PURGE uses a pricing based approach and buffer occupancy fraction to curb unfairness. CHOKeH is a stateless AQM that enforces fairness by splitting the queue into dynamic regions and treats each region differently for performing matched-drops using a dynamically updated drawing factor. DFS particularly focuses on high-rate DoS attacks and works on the basis of weighted fair share. It employs two types of data structures, namely, B-Tree and Cache. Flows that deviate from their allocated fair share and have higher bit rates are marked. Once the unfairness index of a marked flow exceeds the maximum threshold, the flow id of that flow is stored in Cache and every incoming packet from that flow is dropped. Legitimate flows are stored in B-tree. When flows stored in Cache change their behavior (i.e., reduce their bit rate to fair share) and their unfairness index drops, they are moved from Cache to B-Tree. If a flow is in B-tree and its unfairness index exceeds a certain threshold, it is moved from B-tree to Cache.

Although DFS has defined a strict criterion to filter out high-rate flooding attacks, there are vulnerabilities that can be exploited by malicious users, which can jeopardize the effectiveness of DFS against DoS inflicting flows. The first vulnerability of DFS is its inability to detect low-rate DoS attacks against application servers (LoRDAS). A low-rate DoS attack is launched through multiple flows originating from unique IP sources, each of which is responsive, obeys its fair share and sends packets at a low rate to avoid filtering [1]–[4]. The second shortcoming of DFS lies in its incapability to detect IP spoofing attacks. Another shortcoming is that DFS is a full state algorithm and uses the bookkeeping method to determine the number of active flows, which means that it analyzes every incoming and outgoing packet and stores it in a pre-allocated memory location for each incoming flow. This requires a large state space that grows with the increasing number of active flows. The next section presents the proposed DDP scheme that overcomes the aforementioned vulnerabilities of DFS.

III. DETERMINISTIC DOS PREVENTION (DDP)

In this section, improvements are proposed to overcome the vulnerabilities found in DFS. The proposed DDP scheme consists of four constituent algorithms, namely, the Enqueue Operation (Algorithm 1), Ingress Filter (Algorithm 2), LowRateDoS Filter (Algorithm 3) and Malicious Operation (Algorithm 4). Enqueue Operation is the main part that decides whether to admit, drop or mark an arriving packet based on the decisions returned by the other constituent algorithms. Ingress Filter provides a defense mechanism against IP spoofing, LowRateDoS Filter provides defense against the low-rate DoS attacks, whereas Malicious Operation provides a defense mechanism against those attacks which consume more than their fair share and have high bit rates. Each of these algorithms is explained in detail in the following subsections. All DDP parameters are listed in Table I. We start by outlining a partial state keeping technique to estimate the number of active flows (N_{act}), which will later be used in the Enqueue Operation.

TABLE I. LIST OF NOTATIONS AND FUNCTIONS

Parameter	Semantics
br_i	Stored bit rate of flow i
β	Scaling factor for EWMA of wfs
data_limit	Maximum limit on data sent by a user
fid_i	Flow id of flow i
freeze_time	Time to wait before updating p_{m_i}
$mark_i$	Boolean variable for marking fid_i as malicious
max_{th}	Maximum value of p_{m_i} for each flow
min_{th}	Minimum value of p_{m_i} for each flow
p	An arriving packet
p_{m_i}	Unfairness indicator of flow i
p_{time_i}	Timestamp when value of $p_{m_i}$was updated
prune_interval	Time window for observing a flow's behavior
prune_cycle	Two *prune_intervals*
$store_{br_i}$	Boolean variable to store bit rate of a flow
sz_i	Number of packets of flow i in buffer
δ_i	Increment factor for p_{m_i}
δ_d	Decrement factor for p_{m_i}
tolerance_factor	To identify trending flows
Allow(p)	Allows packet p to enter the queue
BitMapZero	Returns number of zero bits in the Bitmap
Drop(p)	Drops packet p
ECN-Mark(p)	Marks ECN bit in packet header if enabled
EWMA(*wfs*)	Exponentially weighted moving average of *wfs*
InsertNewFlow(i)	Make new flow entry i in B-Tree
MarkBitMap(i)	Uses fid_i to set a bit in the Bitmap
ResetFlow(i)	Resets sz_i, $mark_i$, br_i, $store_{br_i}$ *to 0*; and $p_{m_i}$to min_{th}
Size(p)	Extracts size of packet p from its header

A. Estimating the Number of Active Flows (N_{act})

Instead of using the bookkeeping approach to determine the number of active flows in a router's buffer as in DFS, the proposed scheme uses the Bitmap based flow estimation technique [14], in order to reduce the state space cost. In this technique, instead of allocating fixed memory locations for the active flows, a Direct Bitmap is used. The technique employs hashing to estimate N_{act}. The Hash function uses the flow id of each packet as input to map against the bits in the Bitmap. All bits in the Bitmap are initially marked as zero. When a new packet arrives, the Hash function hashes its flow id to one of the bits in the Bitmap and this bit is set to one. No matter how many packets a single flow id sends, they all map to the same bit. In the end, the number of zero bits is used to estimate N_{act} as [14],

$$N_{act} = sb \ln\left(\frac{sb}{z}\right), \quad (1)$$

where sb represents the size of the Bitmap and z is the number of zero bits. The reason for using zero bits is that there is a possibility of collision that may occur, which can be avoided by counting the zero bits. N_{act} is then used to calculate the weighted fair share, wfs, as [6],

$$wfs = \frac{b}{N_{act}}\left(\frac{100}{q_p} - 1\right), \quad (2)$$

where b is the buffer size and q_p is the instantaneous queue size. The objective of using the proposed N_{act} estimation technique is to reduce the space and time complexity of DDP.

B. Enqueue Opertion

Enqueue Operation (Algorithm 1) either allows or drops the packet based on the processing of Ingress Filter, LowRateDoS Filter and Malicious Operation. When an incoming packet, p, enters the router, the queue is checked. If the queue is already full, then p is dropped. Otherwise, its flow id, fid_i, and packet

Algorithm 1. Enqueue Operation

1. **procedure** ENQUEUE(p)
2. **if** q < b **then**
3. $size_p$←*PacketSize(p)*
4. fid_i←*FlowID(p)*
5. bool *drop* ←*false*
6. **if** INGRESS_FILTER(p) **then**
7. Drop(p)
8. **else if** LOWRATEDoS(p) **then**
9. Drop(p)
10. **if** fid_i ϵ HBF_Cache || B-tree **then**
11. *MarkBitMap*(fid_i)
12. z ← *BitMapZero*
13. N_{act}← $sb \ln(\frac{sb}{z})$
14. **if** *MALICIOUS*(p,i,N_{act}) **then**
15. Drop(p)
16. **else**
17. Allow(p)
18. $LR\_Cache_i = LR\_Cache_i + size_p$
19. $sz_i \leftarrow sz_i + size_p$
20. $lr\_sz_i = lr\_sz_i + size_p$
21. **if** $p_{m_i} \leq min_{th}$ & fid_i ϵ HBF_Cache **then**
22. move fid_i to B-tree
23. **else if** $p_{m_i} \geq max_{th}$ & fid_i i ϵ B-tree **then**
24. move fid_i to HBF_Cache
25. **else**
26. *InsertNewFlow(p) to B-Tree*
27. **end procedure**

packet size is extracted from its header. The fid_i is passed to the hashing function to map it into the Direct Bitmap. The number of zero bits in the Bitmap is calculated and stored in variable z. The number of active flows, N_{act}, is then estimated and is passed to the Malicious Operation algorithm to calculate the fair share of an incoming flow.

If a packet fulfils the defined criteria for Ingress Filter (Algorithm 2) and LowRateDoS Filter (Algorithm 3), it is allowed, otherwise it is dropped by the corresponding algorithm. Flow ids of legitimate flows are stored in B-Tree while illegitimate flow ids are stored in High Bandwidth Flow Cache (HBF_Cache) for future use. Illegitimate flows are those which have higher bit rates. When a packet makes its way from both the filters, its fid_i is searched in the HBF_Cache and then it is searched in the B-tree. If the fid_i of the received packet is found in any of these repositories, it is passed to Malicious Operation algorithm for further processing. If no match is found, a new entry is made into B-tree for the new flow. Legitimacy of a flow is maintained by its p_{m_i} whose maximum value is one and minimum value is zero. Malicious Operation algorithm decides how this packet is placed according to its unfairness indicator, p_{m_i}, as either the packet will continue its way out of the router on the outgoing link or it will be marked for dropping later by the Enqueue Operation. Legitimate flows that are trending to be malicious are moved from B-Tree to HBF_Cache. Similarly, illegitimate flows that are trending to be non-malicious are moved from HBF_Cache to B-Tree. If an fid_i is found in B-tree and its p_{m_i} becomes greater than the defined threshold, DDP will move that flow from B-tree to HBF_Cache. In the same way, if an fid_i belongs to HBF_Cache for certain period of time and its flow rate has decreased along with its p_{m_i}, DDP will move that fid_i from HBF_Cache to B-tree. In the end, an outgoing packet that is at the front of the queue will have its fid_i extracted and its size decremented from the HBF_Cache.

Algorithm 2. Ingress Filter

1. **procedure** INGRESS_FILTER(p)
2. src_i ←*SrcIP*(p)
3. **if** (src_i ϵ $subnet_i$)
4. *Allow*(p)
5. **else**
6. *drop* ← *true*
7. **return** *drop*
8. **end procedure**

C. Ingress *Filter*

To cope with the IP spoofing attacks, an Ingress Filter is employed in DDP as given in Algorithm 2. This filter focuses on the subnets assigned to each interface. Every interface of a router is assigned to a specific network ID, subnet mask and default gateway. Every incoming packet on each router's interface is analyzed based upon its IP address to see if it belongs to the same subnet it is coming from [2], [15]. In which case, the packet is allowed to go through the router, else the packet will be dropped. As the passed packet is assigned to the Ingress Filter algorithm, it extracts the fid_i of that specific packet. That fid_i is then checked if it belongs to its assigned subnet, which is maintained in a tabular form at the router. For an illegitimate packet that has an IP address out of its allocated address, a drop is returned by Ingress Filter and the Enqueue Operation will drop that packet. A packet that has an IP address which belongs to the valid address space will be allowed by the Ingress Filter without any further processing. Source traffic coming from any IP that falls out of its assigned subnet will be blocked.

D. *LowRateDoS Filter*

LowRateDoS Filter (Algorithm 3) observes the users on the basis of the data they are transmitting on the uplink. Typical data usage of an average user on the uplink is far less than downlink. However, in case of low-rate DoS attack, data transmitted on the uplink is way more than the downlink [2], [3], [15]. Low-rate DoS attack can be detected by observing users' upstream consumption and filtering out the users having transmitted huge data for a long period of time to a single destination. When a packet is allowed by the Ingress Filter, it is then passed to the LowRateDoS Filter, which will check if the corresponding flow intends to carry out a low-rate DoS Filter attack. To that end, the filter observes different flow ids by the *prune_cycle* (two *prune_intervals*) to collect data on the upstream link at the edge router. Then it analyzes the maximum data sent by the sources in the *prune_cycle* along with its *data_limit*. Data sent by the top users is collected in Enqueue Operation using LR_Cache. If the accumulated data sent by the users exceeds the imposed limit, their flows will be blocked. After completing each turn of *prune_interval*, the value of turn variable is assigned either true or false. After the end of each

Algorithm 3. LowRateDoS Filter

```
1.  procedure LOWRATEDoS(p)
2.  bool  turn ← true
3.  fid_i←FlowID(p)
4.  If(current_time-last_prune > prune_interval
                          && turn==true)
5.      HDF_f ← index of maximum value (LR_Cache)
6.      Reset(LR_Cache)
7.      last_prune ← current_time
8.      turn←false
9.  else if(current_time-last_prune > prune_interval
                          && turn==false)
10.     HDF_s ← index of maximum value (LR_Cache)
11.     Reset(LR_Cache)
12.     last_prune ← current_time
13.     turn←true
14. if(fid_i ϵ HDF_f && fid_i ϵ HDF_s)
15.     if(lr_sz_i > size_limit)
16.         drop ← true
17. return drop
18. end procedure
```

prune_interval, the flow ids are collected and are stored into either High Data Flow (HDF_f) cache for the first *prune_interval* or High Data Flow (HDF_s) cache for the second *prune_interval*, according to their size in the LR_Cache. This is an iterative process in which, when an fid_i with maximum size is collected from LR_Cache, its size is replaced by zero and this flow id is placed at the first index of HDF_f cache. The same process is repeated for the remaining entries. After the *prune_cycle* has elapsed, the fid_i of the packet is verified in HDF_f and HDF_s caches. When an fid_i entry is found in both of the high data flow caches, it is checked against the *data_limit*. The entries (fid_i) having lr_sz_i more than the *data_limit* are classified as malicious and their packets are dropped. Flows that are classified as malicious are restricted for the next *prune_interval* until either of the high data flow caches is updated again.

E. Malicious Operation

When a packet, p, is passed onto Malicious Operation (Algorithm 4), it is clear that it has been allowed by the Ingress Filter as well as by LowRateDoS Filter. Malicious Operation decides on each flow's fairness index, p_{m_i}, whether to entertain the packet or mark it for drop. The decision is made on the basis of analysis of the p_{m_i} and bit rate, br_i, of the incoming flow. When a packet arrives at the buffer, the time is extracted from its header, which is used later to track the time, p_{time}, when the p_{m_i} was updated. Then DDP calculates the *time_gap* by taking difference of the time when packet arrives at buffer and when the p_{m_i} value is last updated. A responsive flow having its p_{m_i} greater than the sum of threshold and *tolerance_factor* * δ_i is ECN marked to warn the flow to reduce its sending rate. If a flow id has p_{m_i} less than the sum of threshold and *tolerance_factor* * δ_i, its occupied size, sz_i, in the buffer is checked against its wfs. If sz_i of that flow in the buffer is greater than its wfs, the packet is ECN marked. If the time gap between the warnings is greater than the time it has to wait, then DDP starts storing its bit rate because this flow is trending and might turn malicious in future. Furthermore, $p_{m_i}$of a flow is

Algorithm 4. Malicious Operation

```
1.  procedure MALICIOUS(p,i,N_act)
2.  time_p←PacketTime(p)
3.  time_gap ←time_p - p_time_i
4.  if p_m_i ≤ min_th + (tolerance_factor*δ_i ) then
5.      if sz_i > EWMA(b/N_act * (100/q_p − 1)) the
6.          ECN-Mark(p)
7.          if time_gap > freeze_time then
8.              store_br_i ← 1
9.              p_m_i ← p_m_i + δ_i
10.             p_time_i ← time_p
11.         else if sz_i = 0 then
12.             p_m_i ← p_m_i - δ_i
13.             p_time_i ← time_p
14.             if p_m_i ≤ min_th then
15.                 ResetFlow(i)
16.     else
17.         ECN-Mark(p)
18.         if(store_br_i==1)
19.             br_i←bitreateof(i)
20.         if mark_i then
21.             drop ← true
22.         if time_gap > freeze_time & store_br_i≠ 0 then
23.             if br_i > fair_bitrate then
24.                 delta_factor ← br_i/fair_bitrate
25.                 p_m_i ← p_m_i + (delta_factor *δ_i)
26.             else
27.                 delta_factor ← fair_bitrate /br_i
28.                 p_m_i ← p_m_i – (delta_factor *δ_i)
29.             p_time_i ← time_p
30.         if p_m_i > max_th then
31.             mark_i ← 1
32.     return drop
33. end procedure
```

incremented by delta and its p_{time} is updated. If the size occupied by the flow in the buffer is less than wfs, then its packet is allowed to go through. Since the value of p_{m_i} is changed as every packet passes through the router, DDP makes sure that the *time_gap* between every update remains greater than the *freeze_time*. The reason for using *freeze_time* is to allow the sending host to get enough time to reduce its sending rate after receiving the ECN. This is how DDP ensures that each responsive flow gets treated in a fair manner. When a responsive flow tends to be malicious, it deviates from its behavior of responding to ECNs, which results in the increment of p_{m_i}value monotonically multiple times. This means that a flow is using a hacked version of TCP which intentionally drops ECN notification issued by the router to avoid reducing its sending rate. Multiple flows showing the same behavior are classified as trending flows. Since these flows show suspicious behavior, DDP keeps record of their bit rates by saving them into the repository. By doing so, DDP observes their $p_{m_i}$and

either increments or decrements it with the help of *delta_factor,* δ, by the degree of their variation form the suggested fair rate. Since the value of δ is directly proportional to the fair bit rate and inversely proportional to the flow's bit rate, this gives DDP the ability to control the flow's state from legitimate to malicious or vice versa. Flows that DDP identifies as malicious will have every packet dropped until they reduce their bit rates. Their p_{m_i} will be decremented once $p_{m_i}$of a malicious flow reaches a minimum (min_{th}= 0), only then they are allowed to send packets across the network. In which case DDP will stop keeping track of their bit rates, they are removed from the malicious flow category and they are unmarked ($mark_i \leftarrow 0$).

IV. Performance Evaluation

This section presents performance evaluation of the proposed DDP scheme, in comparison with DFS [6], using ns–2.35 simulations. The dumbbell topology, shown in Fig. 1, is employed for simulations. Both the routers can hold up to 200 packets of 1 KB each. The latencies range from 1ms to 15ms. Every UDP source has the sending rate of 2 Mbps and is set to send on a constant bit rate. The ratio of TCP to UDP is set to 5:1 [12]. To evaluate the performance of DDP, various scenarios are simulated in which the numbers of total active, legitimate, illegitimate, responsive, and unresponsive flows are varied. A list of scenarios used is given in Table II, and the simulation parameters are given in Table III.

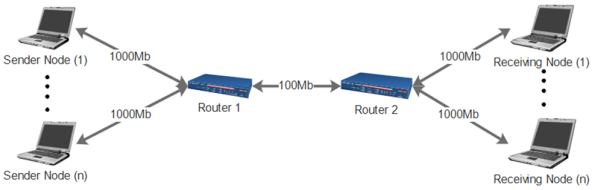


Fig. 1. Dumbbell topology

TABLE II. Scenarios

	Scenarios					
	1	2	3	4	5	6
Total flows	120	120	180	180	240	240
TCP flows	100	100	150	150	200	200
UDP Flows	15	10	20	15	25	20
High data flows	4	9	8	13	12	17
Spoofing flows	1	1	2	2	3	3

TABLE III. Simulation Parameters

Parameters	Values
Buffer size	200 packets
Packet size	1 KB
TCP type	Sack1
UDP rate	2 Mbps
UDP type	CBR
Delay on access links	1~15 ms
Delay on bottleneck link	10 ms
prune_interval	60 seconds
prune_cycle	120 seconds
Simulation time	150 Seconds
freeze_time	10 ms
data_limit	30 MB

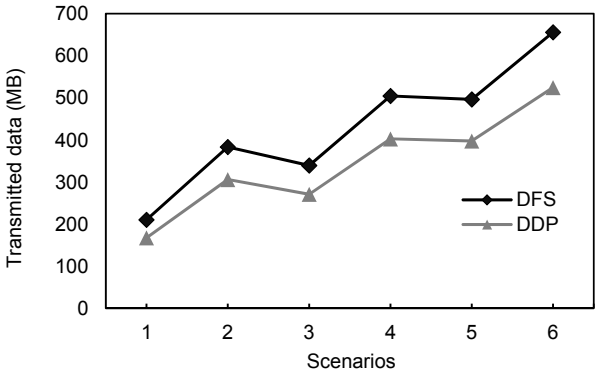


Fig. 2. Data transmitted by high data and spoofing flows

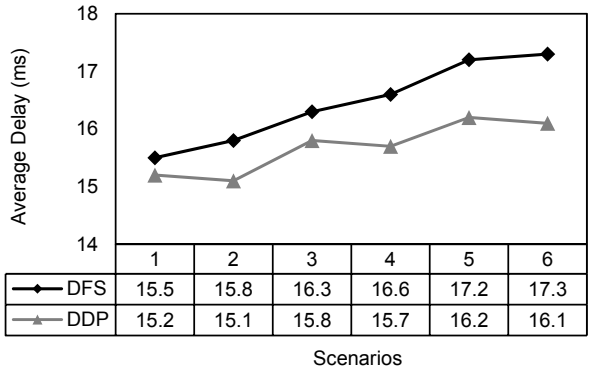


Fig. 3. Average delay of packets under different scenarios

A. Data Transmitted

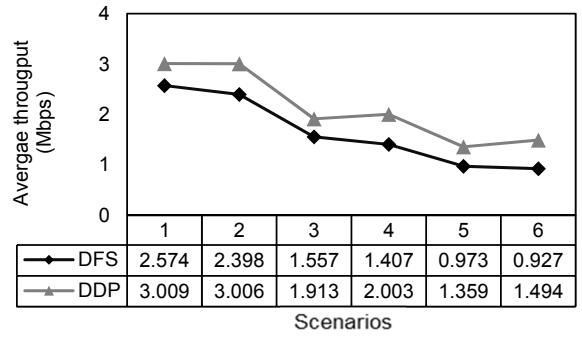
Data transmitted by high data and spoofing flows in all scenarios can be observed in Fig. 2. As the number of high data and spoofing flows increases, the volume of malicious traffic also increases. For every scenario, DDP outperforms DFS due to its Ingress Filter and enhanced Malicious Operation algorithm. Thus, DDP efficiently punishes high data and spoofing flows by limiting their data rates as shown in Fig. 2.

B. Average Delay

In Fig. 3 the average delay of packets is plotted for DFS and DDP. In the figure, Y-axis present the average delay while X-axis list the scenarios used. As shown in the figure, DDP is able to reduce the average delay under various scenarios due to its LowRateDos Filter. An average instantaneous delay is calculated after each *prune_cycle* because the LowRateDoS Filter starts filtering after observing every flow for the *prune_cycle*. As the number of flows increases from left to right, the average delay also increases accordingly.

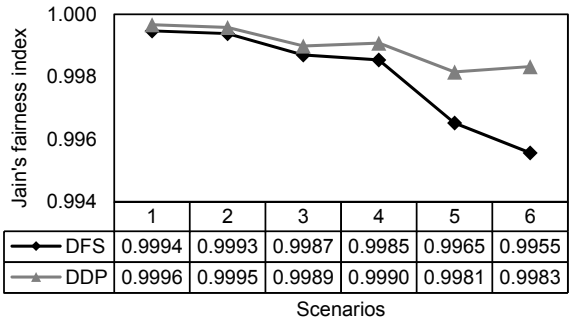
C. Average Throughput

The average throughput of responsive flows for every scenario is plotted in Fig. 4. There is an inverse relationship between throughput and the number of flows on a bottleneck link with constant bandwidth. As the number of flows increases, the average throughput for each flow decreases and vice versa. However, DDP successfully filters the high date unresponsive



	1	2	3	4	5	6
DFS	2.574	2.398	1.557	1.407	0.973	0.927
DDP	3.009	3.006	1.913	2.003	1.359	1.494

Fig. 4. Average throughput of responsive flows



	1	2	3	4	5	6
DFS	0.9994	0.9993	0.9987	0.9985	0.9965	0.9955
DDP	0.9996	0.9995	0.9989	0.9990	0.9981	0.9983

Fig. 5. Jain's fairness index for responsive flows

flows and increases the average throughput for responsive flows. DDP outperforms DFS as, after increasing the total number of flows, the allocated bandwidth for each flow decreases. With the increasing number of flows, the bandwidth allocated for responsive flow chokes, which results in reduced average throughput for each responsive flow. Due to its Ingress Filter and LowRateDoS Filter, DDP allocates more bandwidth to responsive flows and outperforms DFS by effectively filtering out high data unresponsive and spoofing flows.

D. Jain's Fairness Index

Jain's Fairness Index (JFI) is calculated for only responsive flows for every scenario, as shown in Fig. 5. On Y-axis, JFI is plotted while on X-axis various scenarios are listed. Again, DDP outperforms DFS in all scenarios. However, when the total number of flows increases, a drop in JFI is observed in both DDP and DFS. DDP outperforms DFS in all scenarios after filtering out the high data unresponsive flows and treating every responsive flow in a fair manner.

V. CONCLUSION

This paper proposes a novel DoS prevention scheme, called DDP that aims to filter malicious flows initiating DoS attack on network and applications levels. It maintains a partial state to keep track of the number of active flows using the Direct Bitmap technique. Moreover, it determines the legitimacy of a user by keeping record of the data a user is transmitting on the uplink and by employing the Ingress Filter on the incoming traffic. DDP differentiates the traffic on the basis of their fairness consumption, bit rate, data transmission and IP spoofing. Initially, it warns the users through ECN if they are consuming more than their fair share. DDP keeps track of users having high bit rates. When the users exceed the predefined threshold for the bit rate, it drops their packets unless they reduce their bit rate. DDP observes the upload size of users for a specific period of time once they exceed their size limit. It then drops their packet until their cumulative uploaded data size falls below the size limit. DDP also filters out the packets coming from users spoofing their IP addresses. In comparison with a recent AQM based DoS prevention scheme, DDP demonstrates superior performance in terms of filtering low-rate DoS attacks and IP spoofing, reducing average delay of legitimate flows, increased fairness and increased throughput.

REFERENCES

[1] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 2046–2069, Fourth Quarter 2013.

[2] Arbor networks, "Worldwide Infrastructure Security Report," Special Report, vol. XI, 2016. [Online] Available at: https://www.arbornetworks.com/images/documents/WISR2016_EN_Web.pdf

[3] J. Pescatore, "DDoS attacks advancing and enduring: a SANS survey," A SANS Analyst Survey, SANS Institute, February 2014.

[4] U. Habiba, R. Masood, M. A. Shibli, and M. A. Niazi, "Cloud identity management security issues & solutions: a taxonomy," *Complex Adaptive Systems Modeling*, vol. 2, no. 5, pp. 1–37, November 2014.

[5] G. Abbas, Z. Halim, and Z. H. Abbas, "Fairness-driven queue management: a survey and taxonomy," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 324–367, First Quarter 2016.

[6] H. Bedi, S. Roy, and S. Shiva, "Mitigating congestion based DoS attacks with an enhanced AQM technique," *Computer Communications*, vol. 56, pp. 60–73, February 2015.

[7] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[8] Karmeshu, S. Patel, and S. Bhatnagar, "Adaptive mean queue size and its rate of change: queue management with random dropping," *Telecommunication Systems*, vol. 65, no. 2, pp. 281–295, June 2017.

[9] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe—A stateless active queue management scheme for approximating fair bandwidth allocation," in *Proc. 19th Annu. Joint Conf. IEEE INFOCOM*, Tel Aviv, Israel, 26–30 March, 2000, vol. 2, pp. 942–951.

[10] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "Stochastic fair blue: A queue management algorithm for enforcing fairness," in *Proc. 20th Annu. Joint Conf. IEEE INFOCOM*, vol. 3, Anchorage, AK, USA, 22–26 April, 2001, pp. 1520–1529.

[11] R. Mahajan, S. Floyd, and D. Wetherall, "Controlling high-bandwidth flows at the congested router," in *Proc. 9th ICNP*, Riverside, CA, USA, 11–14 November, 2001, pp. 192–201.

[12] G. Abbas, A. K. Nagar, H. Tawfik, and J. Y. Goulermas, "Pricing and unresponsive flows purging for global rate enhancement," *Journal of Electrical and Computer Engineering*, vol. 2010, article ID. 379652, June 2010.

[13] G. Abbas, S. Manzoor, and M. Hussain, "A stateless fairness-driven active queue management scheme for efficient and fair bandwidth allocation in congested Internet routers," *Telecommunication Systems,* DOI 10.1007/s11235-017-0306-3, pp. 1–18, April 2017.

[14] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *Proc. 3rd ACM SIGCOMM conference on Internet measurement*, IMC'03, Miami Beach, FL, USA, 27–29 October, 2003, pp. 153–166.

[15] G. Abbas, A. K. Nagar, and H. Tawfik, "On unified quality of service resource allocation scheme with fair and scalable traffic management for multiclass Internet services," *IET Communications*, vol. 5, no. 16, pp. 2371–2385, November 2011.