Hameed and Ali *EURASIP Journal on Information Security* (2018) 2018:11
https://doi.org/10.1186/s13635-018-0081-z

EURASIP Journal on
Information Security

RESEARCH

Open Access

HADEC: Hadoop-based live DDoS detection framework

Sufian Hameed* and Usman Ali

Abstract

Distributed denial of service (DDoS) flooding attacks are one of the main methods to destroy the availability of critical online services today. These DDoS attacks cannot be prevented ahead of time, and once in place, they overwhelm the victim with huge volume of traffic and render it incapable of performing normal communication or crashes it completely. Any delays in detecting the flooding attacks completely halts the network services. With the rapid increase of DDoS volume and frequency, the new generation of DDoS detection mechanisms are needed to deal with huge attack volume in reasonable and affordable response time.

In this paper, we propose HADEC, a Hadoop-based live DDoS detection framework to tackle efficient analysis of flooding attacks by harnessing MapReduce and HDFS. We implemented a counter-based DDoS detection algorithm for four major flooding attacks (TCP-SYN, HTTP GET, UDP, and ICMP) in MapReduce, consisting of map and reduce functions. We deployed a testbed to evaluate the performance of HADEC framework for live DDoS detection on low-end commodity hardware. Based on the experiment, we showed that HADEC is capable of processing and detecting DDoS attacks in near to real time.

Keywords: DDoS, Flooding attacks, DDoS detection, Hadoop

1 Introduction

DDoS flooding attacks are one of the biggest threats faced by the critical IT infrastructure, from the simplest enterprise network to complex corporate networks. DDoS attacks are here since the very advent of computer networks, and they are not going anywhere anytime soon. The first DDoS attack incident [17] was reported in 1999 by the Computer Incident Advisory Capability (CIAC). Since then, most of the DoS attacks are distributed in nature and they continue to grow in frequency, sophistication, and bandwidth. The main aim of these attacks is to overload the victim's machine and make his services unavailable, leading to revenue losses.

Over the years, DDoS has hit major companies and Internet infrastructures, incurring significant loss in revenues. Yahoo! experienced one of the first major DDoS flooding attacks that made their services offline for about 2 h [14]. In October 2002, 9 of the 13 domain name system (DNS) root servers were shut down for an hour because of a DDoS flooding attack [10]. During the fourth quarter of 2010, a hacktivist group called *Anonymous* orchestrated major DDoS flooding attacks and brought down the Mastercard, PostFinance, and Visa websites [9]. Most recently, online banking sites of nine major US banks (i.e., Bank of America, Citigroup, Wells Fargo, U.S. Bancorp, PNC, Capital One, Fifth Third Bank, BB&T, and HSBC) have been continuously the targets of powerful DDoS flooding attack series [17]. The legacy of DDoS continue to grow in sophistication and volume with recent attacks breaking the barrier of hundreds of Gbps [41].

On 21 October 2016, IoT devices (such as printers, cameras, home routers, and baby monitors) were used to generate DDoS attack involving malicious DNS lookup requests from tens of millions of IP addresses [1]. This attack, at that time, was considered largest of its kind in the history with an unprecedented rate of 1.2 Tbps. The main target of the attack was the servers of DYN Inc., a company that controls much of the Internet's DNS infrastructure [25]. This attack was carried out multiple times which rendered major Internet platforms and services unavailable to large swaths of users in Europe and North America for several hours throughout the day.

*Correspondence: sufian.hameed@nu.edu.pk
IT Security Labs, National University of Computer and Emerging Sciences (NUCES), Karachi, Pakistan

Study of recent attacks reveals that with little effort, next generation attack tools would be able to enact DDoS attacks that are thousand times stronger than the ones we see today [33]. One of the major concerns is that performing DDoS attack is extremely simple with websites known as *Booters* or *Stressers* that offer *DDoS as a service*. These booters provide cheap services, and the costs to perform a series of attacks is typically just a few dollars [36]. Recently, GitHub was hit by 1.35 Tbps of traffic all at once [2]. It was the most powerful attack in the history of DDoS, and the alarming fact is that no botnet was required to achieve such high traffic volume. The attackers spoofed GitHub's IP addresses and took control of its memcached (a distributed memory system known for high performance and demand) instances. The memcached systems then return 50 times the data of the requests back to the victim. GitHub called Akamai Prolexic for DDoS mitigation service, and the assault dropped off after 8 min. GitHub was lucky enough to afford robust DDoS mitigation services, but for small to medium enterprises and financial institutions, there is a need for new low cost DDoS defense mechanisms and architectures that can be easily deployed on commodity hardware.

The explosive increase in the volume of Internet traffic and sophistication of DDoS attacks have posed serious challenges on how to analyze the DDoS attacks in a scalable and accurate manner. For example, two of the most popular open-source intrusion detection systems (IDS), Snort [35] and Bro [32], maintain per-flow state to detect anomalies. The Internet traffic doubles every year, and due to that monitoring, large amount of traffic in real-time anomaly detection with conventional IDS has become a bottleneck. Existing solutions require a lot of resource and do not add any value to limited resource environment. Therefore, we need new solutions that can detect DDoS attack efficiently in near to real time.

In [27], Lee et al. have proposed a DDoS detection method based on Hadoop [3]. They have used a Hadoop-based packet processor [26] and devised a MapReduce [7]-based detection algorithm against the HTTP GET flooding attack. They employ a counter-based DDoS detection algorithm in MapReduce that counts the total traffic volume or the number of web page requests for picking out attackers from the clients. For experiments, they used multiple Hadoop nodes (max. 10) in parallel to show the performance gains for DDoS detection. Unfortunately, their proposed framework, in its current form, can only be used for offline batch processing of huge volume of traces. The problem to develop a real time defense system for live analysis still needs to be tackled.

In this paper, we propose HADEC, a Hadoop-based live DDoS detection framework. HADEC is a novel destination-based DDoS defense mechanism that leverages Hadoop to detect live DDoS flooding attacks in wired networked systems. The real motivation behind this study is to check the efficacy of scalable defenses against DDoS flooding attacks using new distributed architectures that can run on low cost commodity hardware. This will help small and medium organizations and financial institutes to protect their infrastructure with in-house low-cost solutions. HADEC comprises of two main components, a capturing server and a detection server. Live DDoS starts with the capturing of live network traffic handled by the capturing server. The capturing server then processes the captured traffic to generate log file and transfer them to the detection server for further processing. The detection server manages a Hadoop cluster, and on the receipt of the log file(s), it starts MapReduce-based DDoS detection jobs on the cluster nodes. The proposed framework implements a counter-based algorithm to detect four major DDoS flooding attacks[1] (TCP-SYN, UDP, ICMP, and HTTP GET). These algorithms execute as a reducer job on the Hadoop detection cluster.

An early version of this work appeared as a short paper [21] and also as a technical report at arXiv [23]. In this paper, we provide a more detailed illustration of different HADEC components. We have added significant new results (approx. 50% new) with different attack volumes, i.e., 80–20 (80% attack traffic and 20% legitimate traffic) and 60–40 (60% attack traffic and 40% legitimate traffic). 80–20 traffic volume is used to emulate flooding behavior where attack traffic surpasses the legitimate one, whereas 60–40 traffic volume is used to emulate low volume attacks where legitimate and attack traffic volume is relatively close. We also performed system benchmarks to show the overall CPU and memory utilized by HADEC during different phases of data capturing, transfer, and attack detection. These system benchmarks are important to evaluate how well the proposed solution works on low-end commodity hardware under different stress conditions.

We deploy a testbed for HADEC on commodity hardware (low-end desktop machine comprising of core i5 CPU), which consists of a capturing server, detection server, and a cluster of ten physical machines, each connected via a Gigabit LAN. We evaluate HADEC framework for live DDoS detection by varying the attack volume and cluster nodes. HADEC is capable of analyzing 20 GB of log file, generated from 300 GBs of attack traffic, in approx. 8.35 mins on a cluster of 10 nodes. For small log files representing 1.8 Gbps, the overall detection time is approximately 21 s. Our system benchmark evaluations show that during the packet capture phase, the CPU usage is 50% on average, whereas the CPU usage remained low with an average of 15% during the log transfer phase. The capturing phase remains low on memory usage with a constant at around 1000 MBs (12–13%), while the transfer phase consumes on average 7600 MBs (94%). The benchmarks on Hadoop's NameNode show reduction in CPU usage with the increase in cluster size and parallelism.

The rest of the paper is organized as follows. Section 2 describes the state of the art. Section 3 describes the HADEC framework design. In Section 4, we discuss the testbed deployment. Section 5 demonstrates the performance of the proposed framework. Comparison with existing work and optimization recommendation for HADEC deployment is discussed in Section 6. Finally, we conclude the paper in Section 7.

2 Related work

Since the inception of DDoS flooding attacks, several defense mechanisms have been proposed to date in the literature [41]. This section highlights the DDoS flooding attacks, followed by a discussion on the application of Hadoop to combat network anomalies, Botnet- and DDoS-related attacks.

The DDoS flooding attacks can be categorized into two types based on the protocol level that is targeted: network/transport-level attacks (UDP flood, ICMP flood, DNS flood, TCP SYN flood, etc.) and application-level attacks (HTTP GET/POST request). The defense mechanisms against network or transport-level DDoS flooding attacks roughly falls into four categories: *source-based*, *destination-based*, *network-based*, and *hybrid* (*distributed*), and the defense mechanisms against application-level DDoS flooding attacks have two main categories: *destination-based* and *hybrid* (*distributed*). Since the application traffic is not accessible at the layer 2 and layer 3, there is no network-based defense mechanism for the application-level DDoS. Following is the summary of features and limitations for the DDoS defense categories.

- *Source-based*: In source-based defense mechanism, the detection and response are deployed at the source hosts in an attempt to mitigate the attack before it wastes lots of resources [28, 29]. Accuracy is a major concern in this approach as it is difficult to differentiate legitimate and DDoS attack traffic at the sources with low volume of the traffic. Further, there is low motivation for deployment at the source ISP due to added cost for community service.
- *Destination-based*: In this case the detection and response mechanisms are deployed at the destination hosts. Access to the aggregate traffic near the destination hosts makes the detection of DDoS attack easier and cheaper, with high accuracy, than other mechanisms [34, 37, 38]. On the downside, destination-based mechanisms cannot preempt a response to the attack before it reaches the victim and wastes resources on the paths to the victim.
- *Network-based*: With network-based approach, the detection and response are deployed at the intermediate networks (i.e., routers). The rationale behind this approach is to filter the attack traffic at the intermediate networks and as close to source as possible [30, 31]. Network-based DDoS defenses incur high storage and processing overhead at the routers, and accurate attack detection is also difficult due to lack of sufficient aggregated traffic destined for the victims.
- *Hybrid* (*Distributed*): In hybrid approach, there is coordination among different network components along the attack path and detection and response mechanisms are deployed at various locations. Destination hosts and intermediate networks usually deploy detection mechanisms, and response usually occurs at the sources and the upstream routers near the sources [39, 40]. Hybrid approach is more robust against DDoS attacks, but due to distributed nature, it requires more resources at various levels (e.g., destination, source, and network) to tackle DDoS attacks. The complexity and overhead because of the coordination and communication among distributed components is also a limiting factor in smooth deployment of hybrid-based DDoS defenses. Recent advents in software defined networking (SDN) bring us new approaches to deal with DDoS attacks in a collaborative manner. SDN controllers lying in different autonomous systems (AS) can securely communicate and transfer attack information with each other. This enables efficient notification along the path of an ongoing attack and effective filtering of trafic near the source of attack, thus saving valuable time and network resources [22, 24].

Analysis of logs and network flows for anomaly detection has been a problem in the information security for decades. New big data technologies, such as Hadoop, has attracted the interest of the security community for its promised ability to analyze and correlate security-related heterogeneous data efficiently and at unprecedented scale and speeds [15]. In the rest of the section, we review some recent techniques (other than [27] , discussed in Section 1) where Hadoop-based frameworks are used to build affordable infrastructures for security applications.

BotCloud [19] propose a scalable P2P detection mechanism based on MapReduce and combination of host and network approaches [20]. First, they generate large dataset of Netflow data [16] on an individual operator. Next, they applied a PageRank algorithm on the Netflow traces to differentiate the dependency of hosts connected in P2P fashion for the detection of botnets. They moved the PageRank algorithm to MapReduce, and the PageRank algorithm executes on data nodes of Hadoop cluster for efficient execution.

Temporal and spatial traffic structures are essential for anomaly detectors to accurately drive the statistics from network traffic. Hadoop divides the data into multiple same size blocks and distributes them in a cluster of data nodes to be processed independently. This could introduce a difficulty in analysis of network traffic where related packets may be spread across different block, thus dislocating traffic structures. Hashdoop [18] resolves this potential weakness by using hash function to divide traffic into blocks that preserve the spatial and temporal traffic structures. In this way, Hashdoop conserves all the advantages of the MapReduce model for accurate and efficient anomaly detection of network traffic.

3 Hadoop DDoS detection framework

This section gives insights about the working of our proposed framework. The framework is fully automated, which captures the log, transfers them to the Hadoop detection cluster, and starts the detection automatically. The Hadoop-based live DDoS detection framework (HADEC) comprise of four major phases (see Fig. 1). By using this approach, we are able to detect DDoS flooding attack in close to real time.

1. Network traffic capturing and log generation.
2. Log transfer.
3. DDoS detection.
4. Result notification.

Each of the abovementioned phases is implemented as separate components that communicate with each other to perform their assigned task. Traffic capturing and log generation are handled at the *capturing server*, whereas DDoS detection and result notification are performed by the *detection server*. Log transfer is handled through web services. In the following subsections, we have explained the functionalities for each of the phase/component in detail.

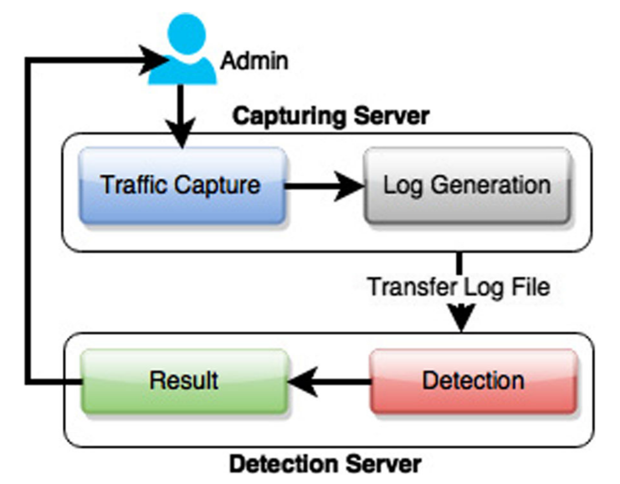


Fig. 1 Different phases of HADEC

3.1 Traffic capturing and log generation

Live DDoS detection starts with the capturing of network traffic. HADEC provides a web interface through which the admin can tune the capturing server with desired parameters. These parameters are file size, number of files to be captured before initializing the detection phase, and the path to save the captured file. Once the admin is done with the configurations, the *Traffic Handler* sends the property file to the *Echo Class* (a java utility to generate logs) and starts the capturing of live network traffic (see Fig. 2).

HADEC use the Tshark library [13] to capture live network traffic. Tshark is an open source library capable of capturing huge amount of traffic. Under default settings, Tshark library runs through command line and outputs the result on console. To log the traffic for later use, we developed a java-based utility (Echo Class) to create a pipeline with Tshark and read all the output packets from Tshark. We have also tuned Tshark to output only the relevant information required during detection phase. This includes information of timestamps, source IP, dst IP, packet protocol, and brief packet header information. The

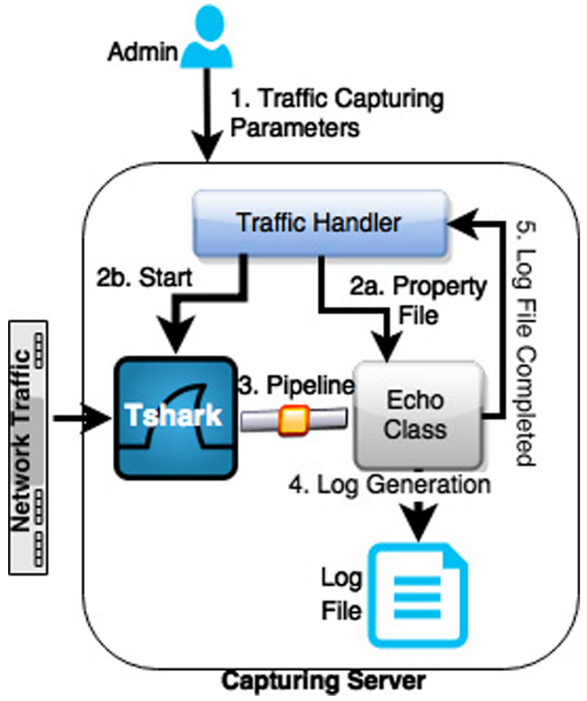


Fig. 2 Network traffic capturing and log generation component

following are the snippets for TCP (SYN), HTTP, UDP, and ICMP packets that are logged in the file.

```
TCP (SYN)
17956 45.406170 10.12.32.1 -> 10.12.32.101
TCP 119 [TCP Retransmission] 0 > 480 [SYN]
Seq=0 Win=10000 Len=43 MSS=1452 SACK_PERM=1
TSval=422940867 TSecr=0 WS=32

HTTP
46737 2641.808087 10.12.32.1 -> 10.12.32.101
HTTP 653 GET /posts/17076163/ivc/dddc?
_=1432840178190 HTTP/1.1

UDP
139875 138.04015 10.12.32.1 -> 10.12.32.101
UDP 50 Src port: 55348 Dst port: http

ICMP
229883 2658.8827 10.12.32.1 -> 10.12.32.1O1
ICMP 42 Echo (ping) request id=0x0001,
seq=11157/38187, ttl=63 (reply in 229884)
```

As discussed above, the *Traffic Handler* sends the property file to the *Echo Class* with the desired set of parameters (file size, file count for detection, and storage path on the capturing server) set by the admin. Echo Class uses these parameters to generate a log file, at the specified location, when it reads the required amount of data from Tshark. Once the log file is generated, the *Echo Class* also notifies the *Traffic Handler* (see Fig. 2).

3.2 Log transfer phase

After the log file is generated, the *Traffic Handler* will notify the *detection server* and also share the file information (file name, file path, server name, etc.) with it via a webservice. The detection server will initiate a Secure Copy or SCP protocol [12] (with pre-configured credentials) with the capturing server and transfer the log file from the capturing server (using the already shared name/path information) into its local file system (see Fig. 3). The capturing server has two network interfaces, one for incoming traffic and one to communicate with the detection server.

Since the detection server mainly works as a NameNode, i.e., the centerpiece of the Hadoop cluster and HDFS (Hadoop distributed file system), it has to transfer the log file(s) from local storage to HDFS. On successful transfer of log file into HDFS, the detection server sends a positive acknowledgement to the capturing server and both the servers delete that specific file from their local storage to maintain healthy storage capacity. Before starting the DDoS detection process, the detection server will wait for the final acknowledgment from the capturing server. This acknowledgement validates that the desired number of files of a particular size (set via parameters by admin) has been transferred to HDFS before the execution of MapReduce-based DDoS detection algorithm. There is no particular restriction on the minimum file count before the detection starts; it could be set to one.

3.3 Detection phase

The Apache Hadoop consists of two core components, i.e., HDFS (storage part) and MapReduce (processing part). Hadoop's central management node also known as NameNode splits the data into same size large blocks and distributes them among the cluster nodes (data nodes). Hadoop MapReduce transfers packaged code for nodes to process in parallel, the data each node is responsible to process.

In HADEC, the detection server mainly serves as the Hadoop's NameNode, which is the centerpiece of the Hadoop DDoS detection cluster. On successful transfer of log file(s), the detection server splits the file into same size blocks and starts MapReduce DDoS detection jobs on cluster nodes (see Fig. 4). We have discussed MapReduce job analyzer and counter-based DDoS detection algorithm in Section 3.5. Once the detection task is finished, the results are saved into HDFS.

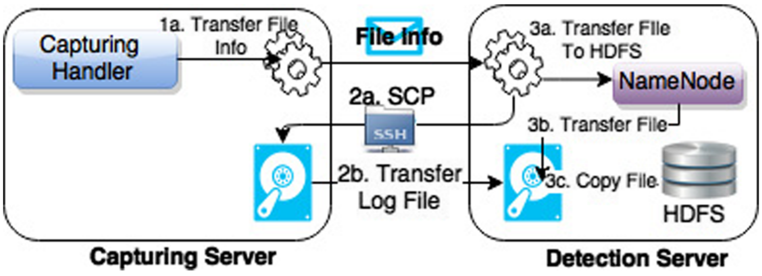


Fig. 3 Log transfer phase

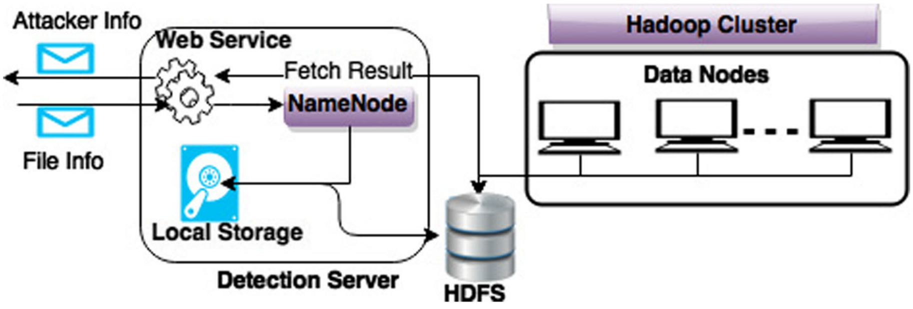


Fig. 4 DDoS detection on Hadoop cluster

3.4 Result notification

Once the execution of all the MapReduce tasks is finished, Hadoop will save the results in HDFS. The detection server will then parse the result file from HDFS and send the information about the attackers back to the administrator via the capturing server. Once the results are notified, both the input and output folders from HDFS will be deleted for better memory management by the detection server. Figure 5 presents a holistic illustration of HADEC framework.

3.5 MapReduce job and DDoS detection

A MapReduce program is composed of a map task that performs filtering and sorting and a reduce task that performs a summary operation. Here, we have explained how HADEC has implemented detection of DDoS flooding

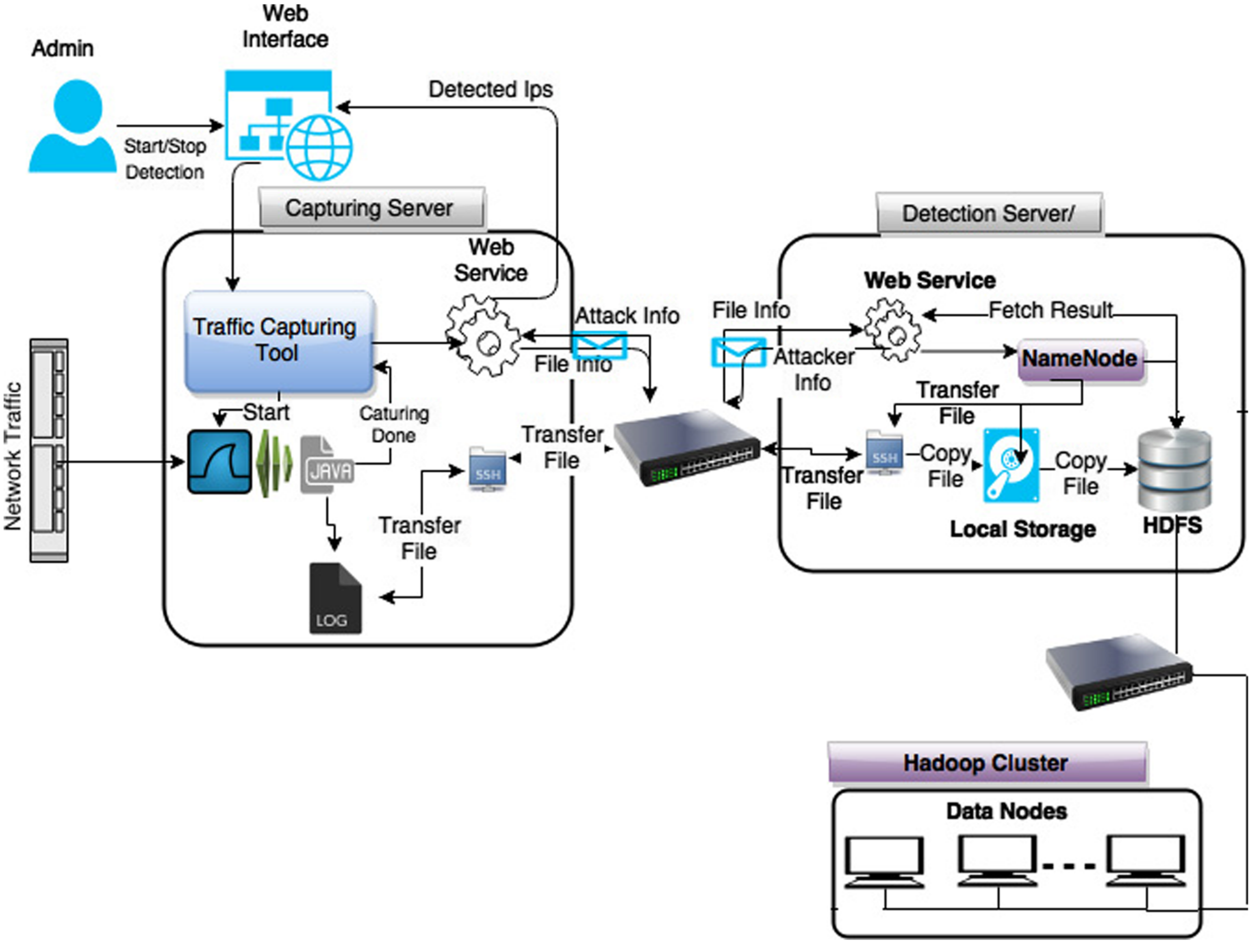


Fig. 5 HADEC: Hadoop-based DDoS setection framework

attacks (UDP, HTTP GET, ICMP, and TCP-SYN) as a MapReduce task on Hadoop cluster using *counter-based algorithm*.

3.5.1 HADEC mapper job

After starting MapReduce task, the first task is a mapper task which takes input from HDFS as a block. In our case, the block will represent a file in text format and the input for each iteration of mapper function will be a single line from the file. Any single line in the file contains only brief information of a network packet captured through Tshark (see Section 3.1). The term network packet used in the rest of this section represents a single line content of the file read as a mapper input.

Mapper job takes pair of data as input and returns a list of pairs (key, value). Mapper output type may differ from mapper's input type; in our case, the input of mapper is pair of any number *i* and the network packet. The output is a list of pair (key, value) with key as the source IP address and value as a network packet. Mapper job also use hashing for combining all the logs of data on the basis of source IP address, so that it becomes easier for reducer to analyze the attack traffic.

After all the mapper have finished their jobs, the data or worker nodes perform a shuffle step. During shuffling, the nodes redistribute the data based on the output keys, such that all data belonging to one key are located on the same worker node (see Fig. 6).

In HADEC, for analysis and detection of UDP flooding attack, the mapper task filters out the packets having UDP information. In particular, the mapper function will search packets having QUIC / UDP information. QUIC stands for Quick UDP Internet connection. For the packet that contains the desired information, the mapper function generates an output in the form of pairs (key, value). The pseudocode for mapper function is as follows.

```
%UDP detection mapper function
function Map is
  input: integer i, a network packet
begin function
  filter packet with QUIC/UDP type
  if packet does not contain information
  then
   ignore that packet
  else
produce one output record(sourceIP,packet)
  end if
end function
```

For ICMP-, TCP-SYN-, and HTTP-GET-based flooding attacks, the mapper function will search for SYN, ICMP, and HTTP-GET packet type information respectively.

3.5.2 HADEC reducer job and counter-based algorithm

Once the mapper tasks are completed, the reducer will start operating on the list of key/value pairs (i.e., IP/packet pairs) produced by the mapper functions. The reducers are assigned a group with unique key; it means that all the packets with unique key (unique source IP in our case) will be assigned to one reducer. We can configure Hadoop to run reducer jobs on varying number of data nodes. For efficiency and performance, it is very important to identify the correct number of reducers required for finalizing the analysis job. HADEC runs counter-based algorithm to detect DDoS flooding attacks on reducer nodes. The reducer function takes input in key/value pair (source IP, packet of type X) and produces a single key/value pair (source IP, no. of packets of type X) output after counting the number instance (see Fig. 6).

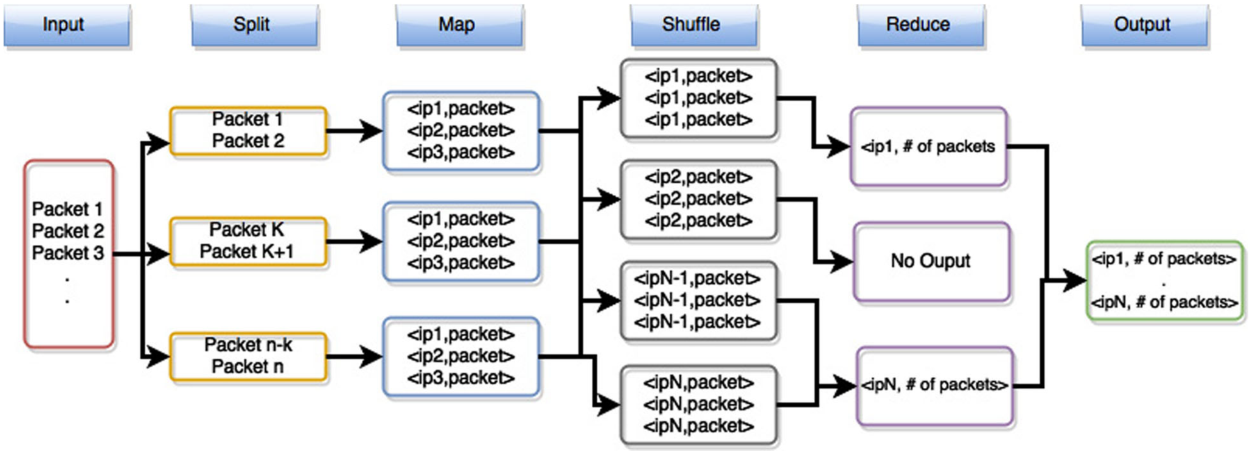


Fig. 6 Mapper and reducer operations

Counter-based algorithm is the simplest, yet very effective algorithm to analyze the DDoS flooding attacks by monitoring the traffic volumes for source IP addresses. The algorithm counts all the incoming packets, of a particular type (UDP, ICMP, HTTP ...etc.), associated with a unique IP address in a unit time. If the traffic volume or count for source IP exceeds the pre-defined threshold, that particular IP will be declared as an attacker. The pseudocode for reducer function using counter-based algorithm for UDP attack is as follows.

```
/*%Reducer function for UDP attack
detection*/
function Reduce is
  input: <source IP, UDP Packets>
  begin function
count :=count # of packets for source IP
    if(count is greater than THRESHOLD)
      begin if
/*This IP declares to be the Attacker IP*/
produce one output <source IP, #of Packets>
      end if
    else
      Ignore (do nothing)
end function
```

4 HADEC testbed

In this section, we have discussed the testbed deployment of HADEC and how we have evaluated the performance of the proposed framework with different experiments.

HADEC performs two main tasks, (a) capturing and transfer of network traffic and (b) detection of DDoS flooding attacks. For capturing the traffic, we use a single node capturing server to capture, process, and send the network traffic to detection server. For DDoS detection, we deploy a single node detection server (also acts as NameNode of Hadoop cluster) and a Hadoop detection cluster consisting of ten nodes. Each node in our testbed (one capturing server, one detection server, and ten Hadoop data nodes) consists of 2.60 GHz Intel core i5 CPU, 8 GB RAM, 500 GB HDD, and 1 Gbps Ethernet card. All the nodes in HADEC used Ubuntu 14.04 and are connected over a Gigabit LAN. We have used Hadoop version 2.6.0 for our cluster and YARN [4] to handle all the JobTracker and TaskTracker functionality.

There are several attack generation tools that are available online, such as LOIC [6], Scapy [11], Mausezahn [8], Iperf [5], etc. For our testbed evaluations, we have mainly used Mausezahn, because of its ability to generate huge amount of traffic with random IPs to emulate different number of attackers. We deployed three dedicated attacker nodes along with couple of legitimate users to flood the victim machine (capturing server) with a traffic volume of up till 913 Mbps (practically highest possible for a Gigabit LAN). HADEC testbed is shown in Fig. 7. For evaluations, we have only focused on UDP flooding attack due to its tendency to reach high volume from limited number of hosts. Any given measurement or datapoint in the result graphs is an average of five readings. We would also like to add that for all the evaluations, we have used only a single reducer, different variations were tried but there was no performance gains.

5 Performance evaluation

We have considered different factors for the performance evaluation of our proposed framework. The overall performance of HADEC depends on:

- The time taken for capturing and transferring the log files.
- The number of attackers and attack volume.
- The execution time of DDoS detection algorithm on the Hadoop cluster.
- The system benchmarks (CPU and memory usage).

For our evaluations, we varied different parameters like log file size, Hadoop cluster size, attack volume, Hadoop splits or block size, and threshold for counter-based algorithm, and measured their impact on the performance of HADEC. The admin can change these parameters on the fly in the configuration settings to adjust any errors or bias present in the results. These parameters are briefly described as follows.

- *File size*: We evaluate the performance of our framework with different sizes of log files. We consider log file size of 10 MB, 50 MB, 100 MB, 200 MB, 400 MB, 600 MB, 800 MB, and 1 GB and evaluate how well the framework performs with varying log size.
- *Cluster size*: Hadoop distributes the processing among the available cluster nodes (datanodes). This means that increase in the cluster size will effect detection phase performance. We used varying cluster size between 2, 4, 6, 8, and 10 datanodes to observe the performance enhancement.
- *Attack volume*: For our evaluations, we used two with different attack volumes, i.e., 80–20 (80% attack traffic and 20% legitimate traffic) and 60–40 (60% attack traffic and 40% legitimate traffic). 80–20 traffic volume is used to emulate flooding behavior where attack traffic surpasses the legitimate one, whereas 60–40 traffic volume is used to emulate low volume attacks where legitimate and attack traffic volume is relatively close.
- *Block size*: Hadoop's NameNode splits the log file data into same size large blocks and distributes them among the cluster nodes (data nodes). We evaluated HADEC with different block sizes since they directly influence the degree of parallelism.

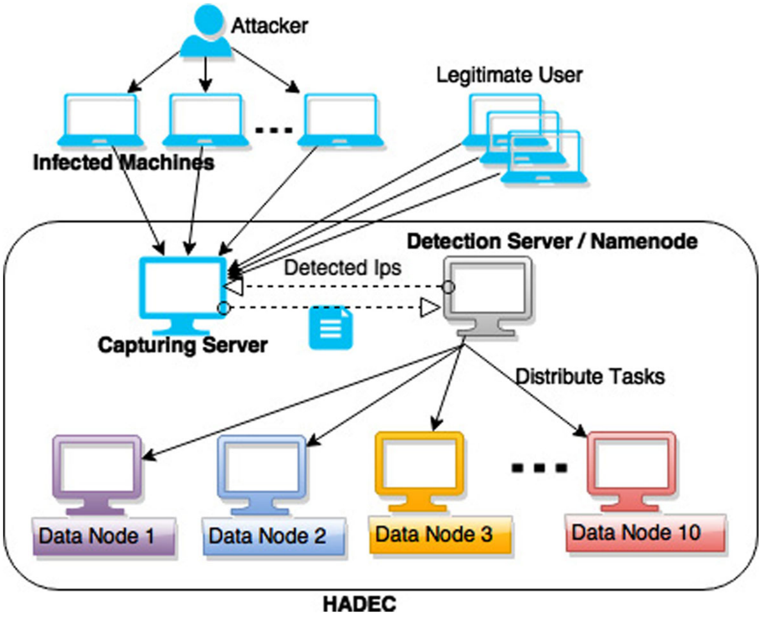


Fig. 7 HADEC testbed

– *Threshold*: This marks the cutoff frequency after which any particular IP is marked as attacker. We have used two threshold values of 500 and 1000 packets per second for our evaluations.

5.1 Traffic capturing and file transfer

The capturing server works on two major tasks simultaneously. First, it captures huge amount of network traffic (913 Mbps in our testbed) and transforms it into log file(s). Second, it transfers the log file(s) to the detection server for further processing. This simultaneous execution of capture and transfer operations are important for live analysis of DDoS flooding attack, but on the other hand, both the operations compete for resources.

Figure 8 shows the capturing and transfer time taken by the capturing server for log files of different sizes. The capturing time is almost linear to the increase in file size. It takes approximately 2 s to log a file of 10 MB and extends to 142 s for 1 GB file. File transfer takes 14 s to transfer 10 MB file and approx. 35 s for 1GB file. This shows a clear improvement in throughput with the increase in file size. Here, it is also interesting to note that the transfer operation has to compete for bandwidth, and during peak time, more than 90% of the bandwidth is being consumed by the

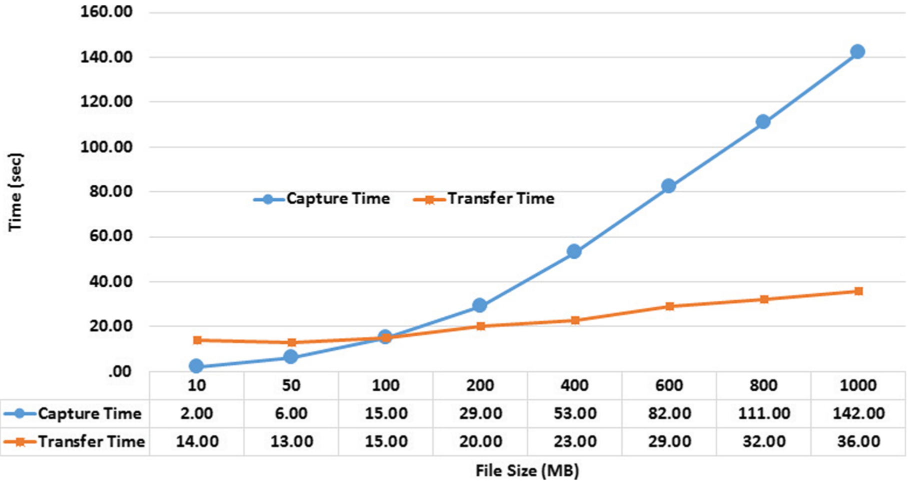


Fig. 8 Capture and transfer time of a log file (file size is in MBs and time is in seconds)

attack traffic. We already discussed above that all the evaluation results are an average of five readings. However, we rounded the average to whole number in Fig. 8. This is because the unit of evaluation is in seconds and we had clear difference in results for different parameters. For all the remaining results, we kept the value till two decimal places because at time it is hard to observe any difference between different experimental settings.

5.2 Number of attackers and attack volume

HADEC use counter-based algorithm to detect attackers. This means that during the DDoS flooding attack, any particular attacker has to cross certain volume threshold to be detected. It is imperative to study the number of attackers one file can capture in it. Table 1 presents the relationship between the size of log file with the total number of attackers and the aggregate traffic volume. According to Table 1, the capturing server has to analyze approx. 0.24 GBs of network traffic to generate a log file of 10 MB and it could represent 100 plus attackers that cross the flooding frequency threshold of 500–1000 packet. By increasing the log file size, the capability to capture accurate information related to attackers also increases. There is a trade-off between the log file size and overall detection rate; therefore, the admin will have to adjust the framework parameters that will best fit in different attack scenarios.

5.3 DDoS detection on Hadoop cluster

In this section, we evaluate the performance of DDoS detection on the Hadoop cluster. For our evaluations, we used different sizes of the log files (10, 50, 100, 200, 400, 600, 800, and 1000 MBs), different threshold values (500 and 1000) for counter-based detection algorithm, and attack volume of 80–20 and 60–40. For all the evaluations in this section, we used fix Hadoop data block of 128 MB.

Figure 9 shows the detection time on Hadoop cluster with 500 threshold. With the increase in file size, the number of attack traffic also increases, which affects the mapper and reducer operation frequency and time. In short, with the increase in file size, the detection time increases and it will also increase the detection rate or the number of attackers IPs, which is a plus point.

Table 1 Relationship of log file size with no. of attackers and traffic volume

File size (MB)	No. of attackers	Traffic vol.
10	100	0.24 GB
50	500	0.67 GB
100	1500	1.67 GB
200	2000	3.23 GB
400	4000	5.91 GB
600	6000	9.14 GB
800	8000	12.37 GB
1000	10,000	15.83 GB

Increase in cluster size hardly affects the detection time for files less than 400 MB in size; on the contrary, in some cases, it might increase a little due to added management cost. Hadoop enables parallelism by splitting the files into different blocks of specified size. Files smaller than the Hadoop block size are not split over multiple nodes for execution. Therefore, the overall detection time remains the same over different cluster node.

Starting from the file size of 400 MB, the detection time improves with the increase of cluster size. For bigger files like 800 MB and 1000 MB, Hadoop works more efficiently. We can see that the detection time reduces around 27 to 37 s for 800 and 1000 MB files respectively, when the cluster size is increased from 2 to 10 nodes. This is because with 1000 MB file, there are nine blocks, and with the increase in cluster size, Hadoop will assign the task to different nodes in parallel.

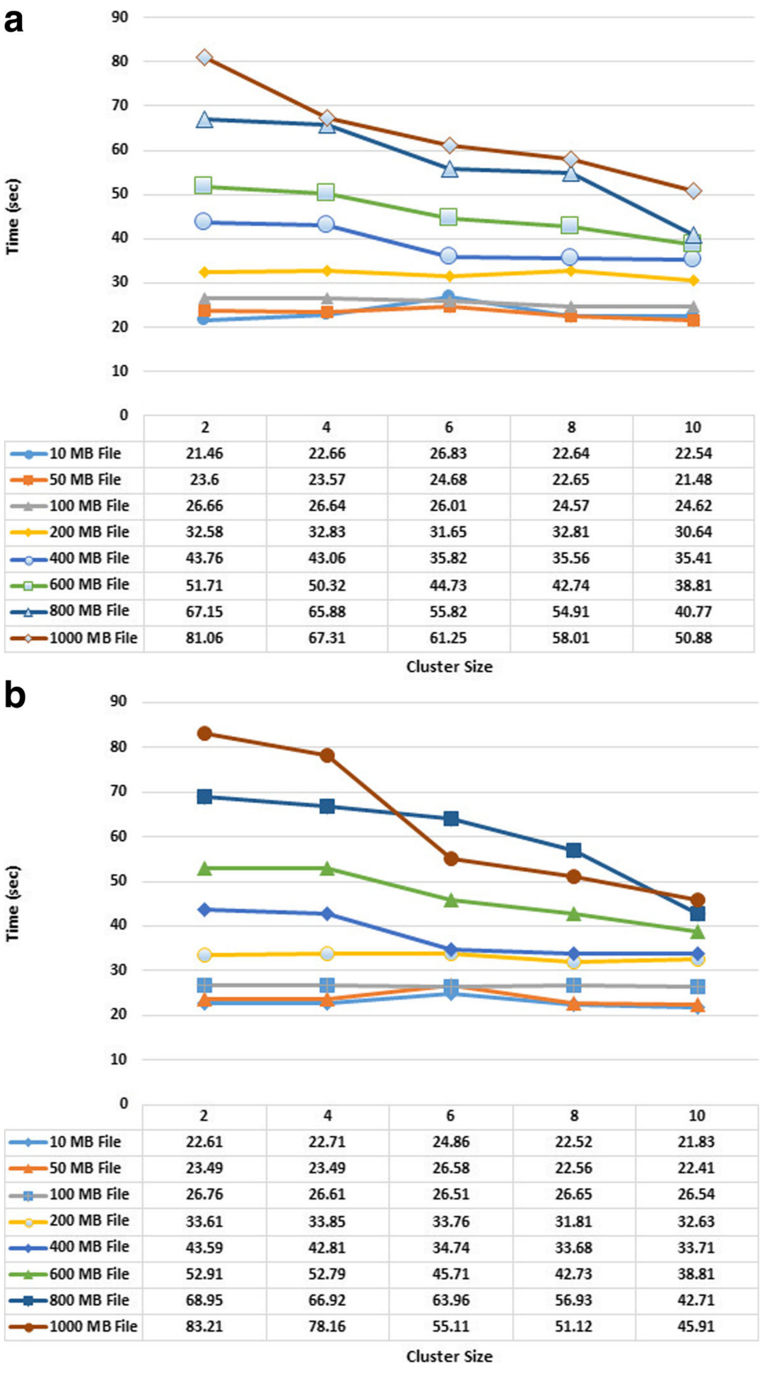
We have evaluated our detection phase on the basis of two different attack volumes: (1) 60–40 attack volume and (2) 80–20 attack volume. The main difference between these two volumes is that in 80% attack volume, the total time for detection slightly increases due to increase in the number of reducer operations. In 80% attack volume, the number of unique IPs is more than 60% attack volume which slightly increases (couple of seconds only) the reducer time for 80% attack volume file size. Again, this difference is only visible for larger log files.

Figure 10 shows the detection time on Hadoop cluster with a threshold value of 1000. In this experiment, we only change the threshold value and all the remaining settings are similar to the Fig. 9. With the increase in threshold value, the total number of inputs for reducers also increases and this will increase the reducer time. This is the reason why *majority* of the results shown in Fig. 10 has couple of second higher detection time as compared to the results in Fig. 9.

5.4 Effect of different block sizes

Figure 11 show the effect of varying block sizes on the detection time for 1 GB file. In this experiment, we use fix threshold of 500, 80–20 attack volume, and three different blocks of size 32, 64, and 128 MB. For 1 GB file the block size of 128 MB gives the maximum performance gains in terms of detection time with the increase in cluster nodes. With smaller block size, there are more splits, resulting in multiple tasks being schedule on a mapper and adds management overhead.

The effect of cluster size is prominent on large files. This is because with large files, Hadoop can effectively split the files in multiple blocks and distribute on the available



a

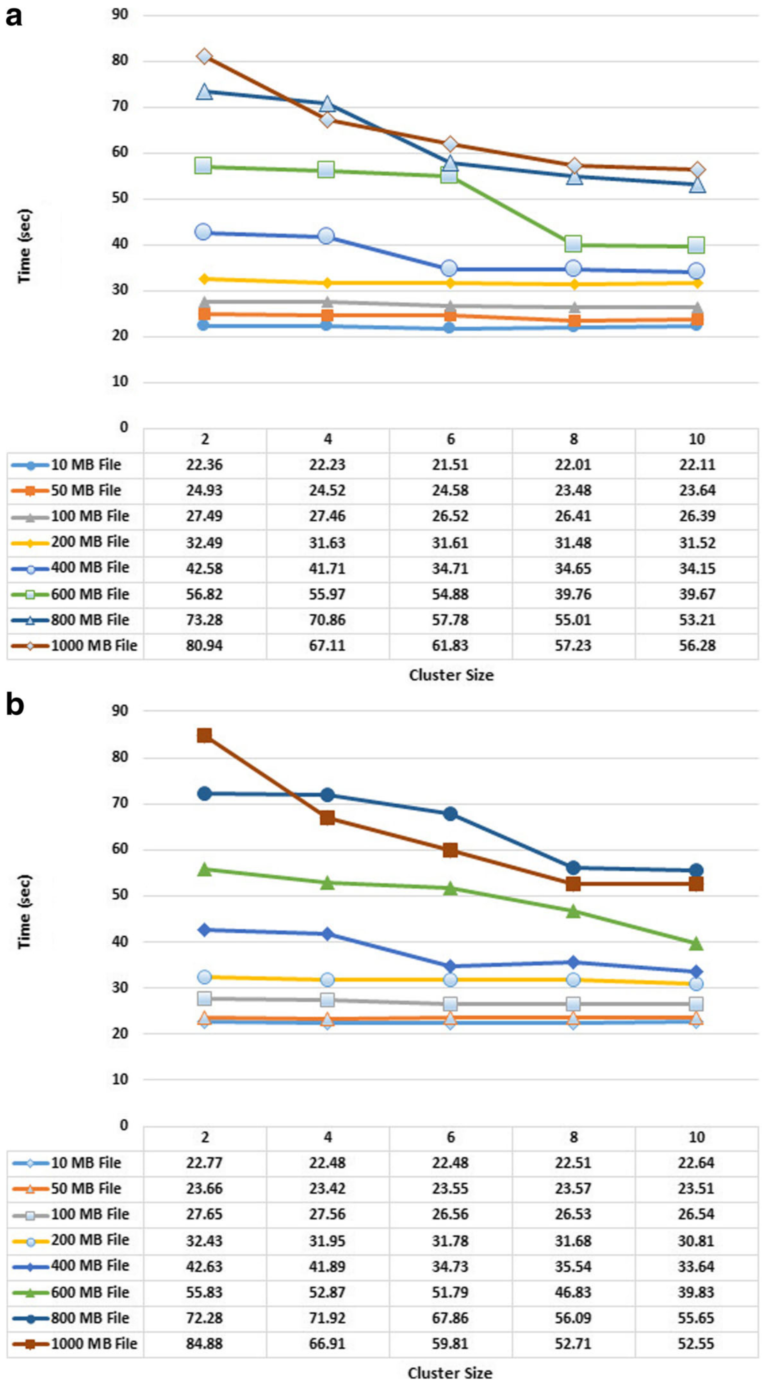
	2	4	6	8	10
10 MB File	21.46	22.66	26.83	22.64	22.54
50 MB File	23.6	23.57	24.68	22.65	21.48
100 MB File	26.66	26.64	26.01	24.57	24.62
200 MB File	32.58	32.83	31.65	32.81	30.64
400 MB File	43.76	43.06	35.82	35.56	35.41
600 MB File	51.71	50.32	44.73	42.74	38.81
800 MB File	67.15	65.88	55.82	54.91	40.77
1000 MB File	81.06	67.31	61.25	58.01	50.88

b

	2	4	6	8	10
10 MB File	22.61	22.71	24.86	22.52	21.83
50 MB File	23.49	23.49	26.58	22.56	22.41
100 MB File	26.76	26.61	26.51	26.65	26.54
200 MB File	33.61	33.85	33.76	31.81	32.63
400 MB File	43.59	42.81	34.74	33.68	33.71
600 MB File	52.91	52.79	45.71	42.73	38.81
800 MB File	68.95	66.92	63.96	56.93	42.71
1000 MB File	83.21	78.16	55.11	51.12	45.91

Fig. 9 Detection time at Hadoop cluster with 500 threshold (detection time is measured in seconds and cluster size varies from 2 to 10 nodes). **a** 80–20 attack volume .**b** 60–40 attack volume

cluster nodes. Figure 12a and 12b show the effect of different block size and cluster node on the detection time for 10 GB and 20 GB respectively, with a fix threshold of 500 and 80–20 attack volume. One hundred twenty-eight megabyte block size gives the most efficient results; this is because when the number of blocks increases, the resource manager in Hadoop needs to manage each of the blocks and its result. Thus, it will take more time to manage each task. For larger block size, there is only one map task to process the whole large block. On a 10 GB file with a block size of 128 MB, Hadoop finished the detection task in approx. 7.5 min with a cluster size of 2 nodes. The detection time goes down to approx 4.5 mins when the cluster size is increased to 10 nodes. For 20 GB



	2	4	6	8	10
10 MB File	22.36	22.23	21.51	22.01	22.11
50 MB File	24.93	24.52	24.58	23.48	23.64
100 MB File	27.49	27.46	26.52	26.41	26.39
200 MB File	32.49	31.63	31.61	31.48	31.52
400 MB File	42.58	41.71	34.71	34.65	34.15
600 MB File	56.82	55.97	54.88	39.76	39.67
800 MB File	73.28	70.86	57.78	55.01	53.21
1000 MB File	80.94	67.11	61.83	57.23	56.28

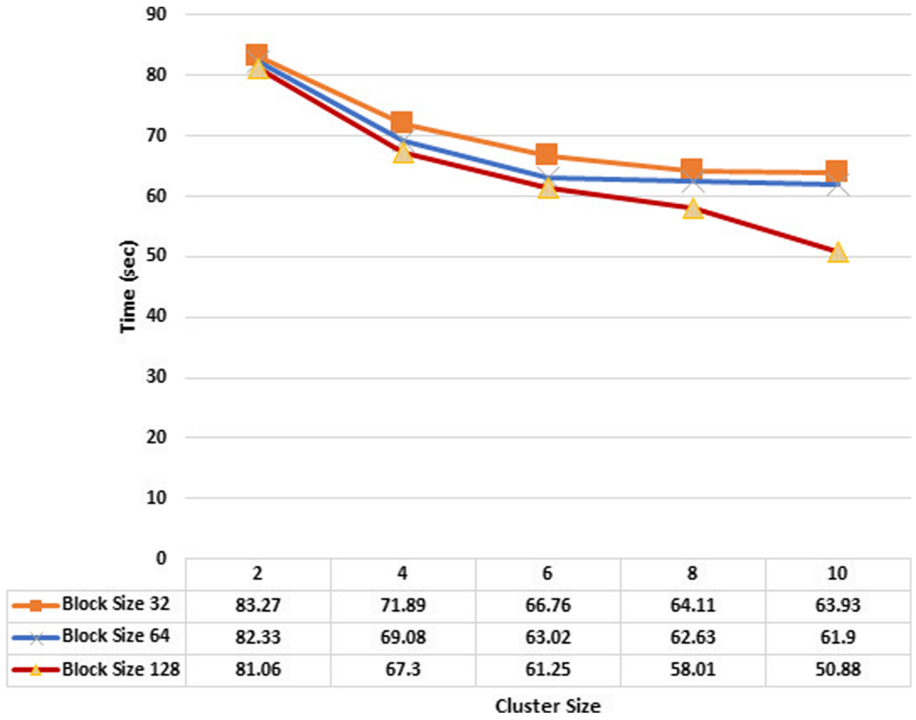
	2	4	6	8	10
10 MB File	22.77	22.48	22.48	22.51	22.64
50 MB File	23.66	23.42	23.55	23.57	23.51
100 MB File	27.65	27.56	26.56	26.53	26.54
200 MB File	32.43	31.95	31.78	31.68	30.81
400 MB File	42.63	41.89	34.73	35.54	33.64
600 MB File	55.83	52.87	51.79	46.83	39.83
800 MB File	72.28	71.92	67.86	56.09	55.65
1000 MB File	84.88	66.91	59.81	52.71	52.55

Fig. 10 Detection time at Hadoop cluster with 1000 threshold (detection time is measured in seconds and cluster size varies from 2 to 10 nodes). **a** 80–20 attack volume .**b** 60–40 attack volume

file with a block size of 128 MB, the time to finish the detection task is 14.6 min and 8.3 mins on a cluster of 2 and 10 nodes respectively. If we approximate the numbers in Table 1, HADEC can effectively resolve 100K attackers for an aggregate traffic volume of 159 GBs with 10 GB of log file in just 4.5 min. These numbers are doubled for 20 GB.

5.5 Overall framework performance

Figures 13 and 14 show the overall performance of our proposed framework to detect the DDoS attacks. These numbers present the total time required for capturing, processing, transferring, and detection with different file sizes. For the experiments in Fig. 13a, we have used 80–20 attack volume, 128 MB block size, and 500 threshold.



	2	4	6	8	10
■ Block Size 32	83.27	71.89	66.76	64.11	63.93
× Block Size 64	82.33	69.08	63.02	62.63	61.9
▲ Block Size 128	81.06	67.3	61.25	58.01	50.88

Fig. 11 Detection time with different block sizes on 1 GB file (detection time is measured in seconds and cluster size varies from 2 to 10 nodes)

For the experiments in Fig. 14a, we have only changed the threshold to 1000. The experiments in Fig. 13b and Fig. 14b used 60–40 attack volume with 500 and 1000 threshold respectively.

In Fig. 13a, we can observe that with the increase in the file size, the overall overhead of capturing and transferring phase increases. A 10 MB file takes approx. 16 s (42%) in capturing/transferring phase and 21 s in detection phase. The best case of 1 GB file (10 node cluster) takes 178 s (77%) in capturing/transferring phase and just 50 s in detection phase. On the whole, it takes somewhere between 4.3 and 3.82 min to analyze 1 GB of log file that can resolve 10K attackers and generated from an aggregate attack volume of 15.83 GBs. The variation of threshold and attack volume in Fig. 14a, Fig. 13b, and Fig. 14b slightly effects the overall detection time with couple of seconds.

5.6 System benchmarks

In this section, we evaluated the system benchmarks (CPU and memory usage) for the capturing, transferring, and detection phases in HADEC and log the readings every second. For the experiments, we used fix Hadoop block of 128 MB, threshold of 500, and varied file sizes (10, 50, 100, 200, 400, 600, 800, and 1000 MB) with different number of cluster nodes. For the benchmark evaluations, we ran all the phases (capturing, transfer, and detection) in isolation to capture the performance of each phase without any external influence. Each phase shows a different CPU and memory usage pattern (see Table 2 for average usage). However, we observed that varying the experimental parameters like file size made no clear difference on CPU and memory usage within each phase (capturing, transfer, and detection). Thus, plotting CPU and memory usage for all the different file sizes results in overlapping plots that are illegible, confusing, and hard to analyze. The following sections discuss average system benchmarks during each phase of the proposed framework.

5.6.1 *Capture phase benchmarks*

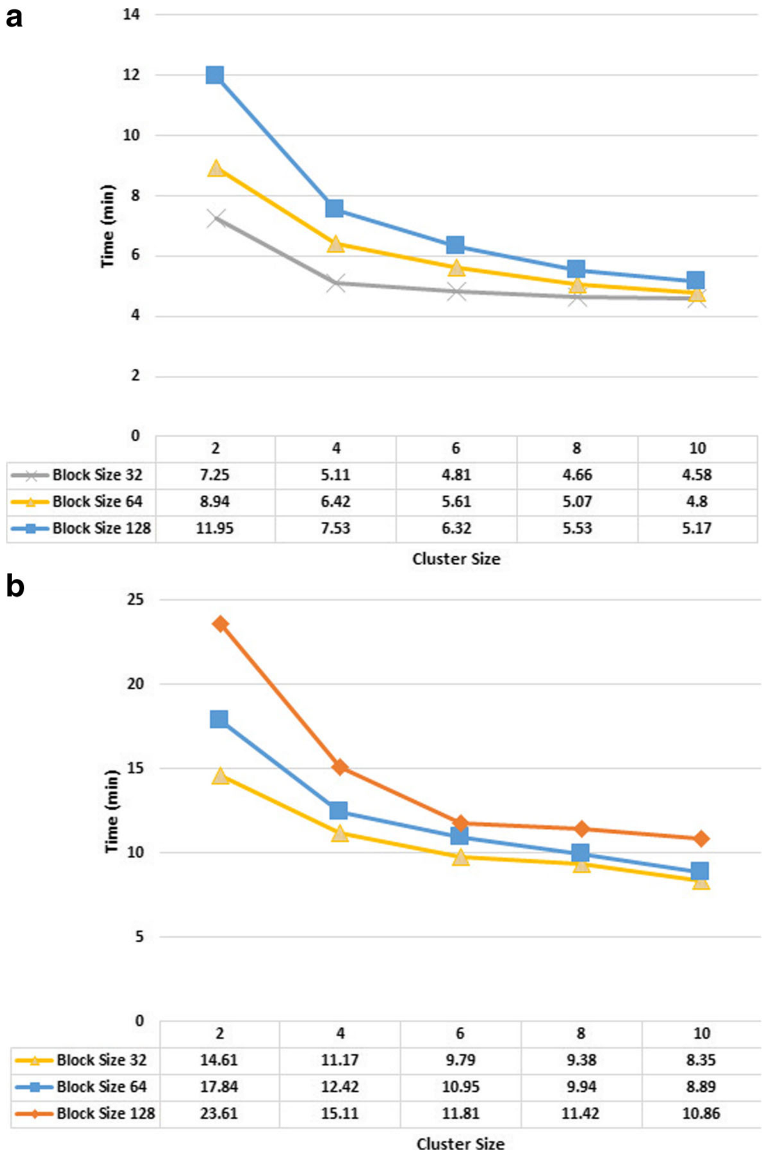
The CPU measurement results show that there is no clear difference of varying file size on the CPU usage. The CPU usage largely remains between the range of 20 to 80% with an average of 50%.

The memory usage remained constant at around 1000 MB mark which is approximately 12–13% of the available memory.

The length of the experiments depends on the size of the file being captured. Once the required size is achieved, the capturing process stops and the CPU and memory usage smooths out within couple of seconds.

5.6.2 *Transfer phase benchmarks*

During the file transfer, the CPU usage is much lower than the capturing phase. The CPU usage mainly remains



	2	4	6	8	10
Block Size 32	7.25	5.11	4.81	4.66	4.58
Block Size 64	8.94	6.42	5.61	5.07	4.8
Block Size 128	11.95	7.53	6.32	5.53	5.17

	2	4	6	8	10
Block Size 32	14.61	11.17	9.79	9.38	8.35
Block Size 64	17.84	12.42	10.95	9.94	8.89
Block Size 128	23.61	15.11	11.81	11.42	10.86

Fig. 12 Effect of block size on huge files (detection time is measured in minutes and cluster size varies from 2 to 10 nodes). **a** 10 GB file. **b** 20 GB file

between the range of 10 to 25% with an average of 15%. When it comes to memory usage, the transfer phase consumes an average of 7600 MBs which is approximately 94% of the available memory.

The length of the experiments depends on the size of the file being transferred to the detection server. Once the file is completely transferred, the CPU and memory usage smooths out within couple of seconds.

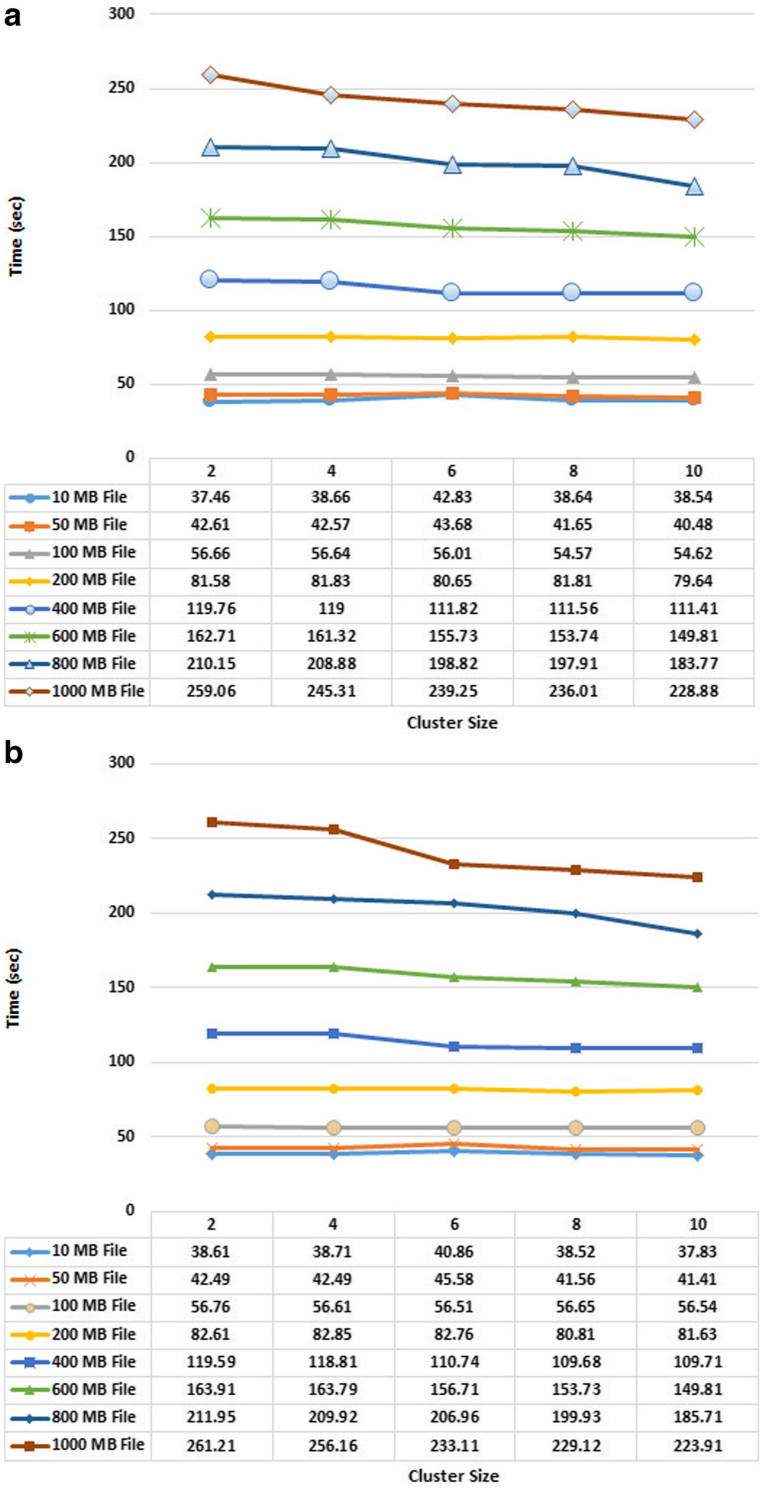
5.6.3 Detection phase benchmarks at the Hadoop NameNode

For the benchmarks at the detection phase, we varied the number of Hadoop cluster nodes for different file sizes and observed the CPU and memory usage at the NameNode.

The memory usage remained the same in overall detection phase, and on average, the memory usage remained between 85 and 95%. The CPU usage on average remained below 40%. In NameNode, we observed that with the increase in cluster size, the CPU usage decreases. Further, we also observed that when parallelism occurs in Hadoop due to bigger file size, the NameNode CPU usage become lower.

6 Comparison with existing work and optimization recommendations

This papers uses destination-based approach due to easy and cheaper DDoS detection with high accuracy. We leverage Hadoop to solve the scalability issue to detect multiple attacks from a huge volume of traffic by parallel

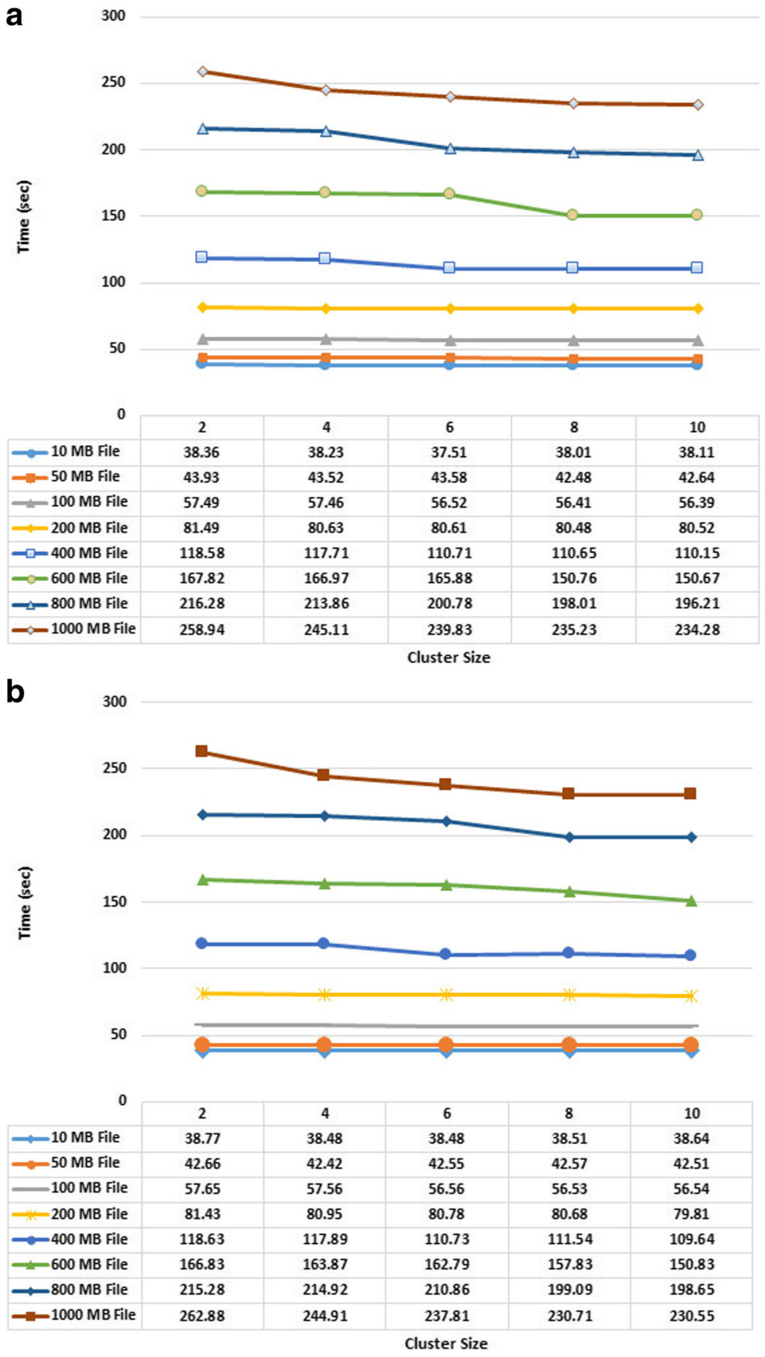


	2	4	6	8	10
10 MB File	37.46	38.66	42.83	38.64	38.54
50 MB File	42.61	42.57	43.68	41.65	40.48
100 MB File	56.66	56.64	56.01	54.57	54.62
200 MB File	81.58	81.83	80.65	81.81	79.64
400 MB File	119.76	119	111.82	111.56	111.41
600 MB File	162.71	161.32	155.73	153.74	149.81
800 MB File	210.15	208.88	198.82	197.91	183.77
1000 MB File	259.06	245.31	239.25	236.01	228.88

	2	4	6	8	10
10 MB File	38.61	38.71	40.86	38.52	37.83
50 MB File	42.49	42.49	45.58	41.56	41.41
100 MB File	56.76	56.61	56.51	56.65	56.54
200 MB File	82.61	82.85	82.76	80.81	81.63
400 MB File	119.59	118.81	110.74	109.68	109.71
600 MB File	163.91	163.79	156.71	153.73	149.81
800 MB File	211.95	209.92	206.96	199.93	185.71
1000 MB File	261.21	256.16	233.11	229.12	223.91

Fig. 13 Total time to detect DDoS attack in HADEC with 500 threshold (detection time is measured in seconds and cluster size varies from 2 to 10 nodes). **a** 80–20 attack volume and 500 threshold. **b** 60–40 attack volume and 500 threshold

data processing. Traditional computational architectures mainly deploy single host-based approaches and enhance memory efficiency or customize process complexity. Initially, Lee and Lee [27] proposed to use Hadoop for DDoS detection in a position paper. However, they only performed offline analysis on the traffic logs and their



	2	4	6	8	10
10 MB File	38.36	38.23	37.51	38.01	38.11
50 MB File	43.93	43.52	43.58	42.48	42.64
100 MB File	57.49	57.46	56.52	56.41	56.39
200 MB File	81.49	80.63	80.61	80.48	80.52
400 MB File	118.58	117.71	110.71	110.65	110.15
600 MB File	167.82	166.97	165.88	150.76	150.67
800 MB File	216.28	213.86	200.78	198.01	196.21
1000 MB File	258.94	245.11	239.83	235.23	234.28

	2	4	6	8	10
10 MB File	38.77	38.48	38.48	38.51	38.64
50 MB File	42.66	42.42	42.55	42.57	42.51
100 MB File	57.65	57.56	56.56	56.53	56.54
200 MB File	81.43	80.95	80.78	80.68	79.81
400 MB File	118.63	117.89	110.73	111.54	109.64
600 MB File	166.83	163.87	162.79	157.83	150.83
800 MB File	215.28	214.92	210.86	199.09	198.65
1000 MB File	262.88	244.91	237.81	230.71	230.55

Fig. 14 Total time to detect DDoS attack in HADEC with 1000 threshold (detection time is measured in seconds and cluster size varies from 2 to 10 nodes). **a** 80–20 attack volume and 1000 threshold. **b** 60–40 attack volume and 1000 threshold

framework was also discussed like a blackbox. We covered all the design and implementation aspects and thoroughly evaluated live network to measure the efficacy of using Hadoop against DDoS. This is the first effort that prototyped a complete framework against flooding DDoS using Hadoop, and we extensively evaluated it as well.

The primary motivation behind this work was to present a framework based on distributed architecture that can handle high volume flooding attacks using low-end commodity hardware. This will ease the burden on small/medium enterprises and financial institute to deploy in-house cheap defenses and essentially save huge costs incurred from the third party DDoS mitigation

Table 2 System benchmarks (average CPU and memory usage) during capture, transfer, and detection phases

Phase	CPU usage %	Memory usage (%)
Capture phase	50	13
Transfer phase	15	94
Detection phase at NameNode	40	90

service providers. The testbed of our framework is divided into capturing and detection (Hadoop) server. This is a design choice to deploy HADEC on low-end machines (just core i5 with 8 GB ram) for real efficacy. For evaluations, we used desktop machines (core i5) with low computational power. Unlike [27], we extensively evaluated our frame by varying different parameters like log file size, attack volume, Hadoop cluster size, Hadoop splits or block size, and threshold for counter-based algorithm, and measured their impact on the performance of HADEC (like execution time and system benchmarks). These evaluation gave us insights on the strength and bottlenecks of the proposed framework and using this information system admin can customize their deployment strategy for optimal results.

Based on the results presented in this paper, we can conclude that HADEC is capable of analyzing high volume of DDoS flooding attacks in scalable manner. Several gigabytes of attack traffic can be analyzed in couple of minutes (from capturing to detecting). This also involves the bootstrapping time, i.e., initially when traffic was being captured the detection was idle. For an ongoing attack, both phases (capturing and detection) will be executing simultaneously and detection would be more effective. By using small size for log file, the overall detection time can be reduced to couple of seconds (30–40 s). But small log files also have an inherent limitation to the number of attacker's they can track. There is no single recommended setting; the admin will have to tweak the framework configuration that best match their requirement. We also noticed that with smaller files, Hadoop does not provide parallelism. This means that if any admin configures HADEC to work on small files of under 400 MB, there will be no point in setting up multiple node cluster. A single or two node clusters of Hadoop will do the job within few minutes (2–3) with the hardware settings we used in our testbed.

In our evaluations of HADEC, capturing and transferring phase showed the performance overhead and majority of the framework time was spent in these phases. This problem could be easily resolved by using reasonable to high-end server optimized for traffic operations, instead of mid-level core i5 desktop that are used in our testbed. A better approach, for banks and financial institutions, would be to deploy a *network tap* (custom monitoring and recording devices to capture and mirror network traffic) instead of a desktop system to capture and transfer the logs to detection server. A network tap with traffic mirroring capacity of 40–60 Gbps would range between 3000 and 4000 $ (from different vendors like Gigamon, Data-Com, Dualcom, etc.), and it would help transfer the log files representing 60 Gbps (60 Gb or 7.5 GBs will generate a log of approx. 550 MBs) of attack to the detection server in just matter of seconds and the attack logs can be further analyzed at the detection server within a minute. This cost effective (under 4K–6K $) solution would serve the defense needs of medium enterprises and financial institutions and help them save subscription cost to third party DDoS Mitigation services like Akamai Prolexic. Thus, a more effective and cost efficient deployment would be to use only couple of cluster nodes and configure a Network Tap as per traffic requirement for packet capturing and processing.

7 Conclusions

In this paper, we present HADEC, a scalable Hadoop-based live DDoS detection framework that is capable of analyzing DDoS attacks in affordable time. HADEC captures live network traffic, processess it to log relevant information in brief form, and uses MapReduce and HDFS to run detection algorithm for DDoS flooding attacks. HADEC solves the scalability, memory inefficiency, and process complexity issues of conventional solution by utilizing parallel data processing promised by Hadoop. The evaluation results showed that HADEC would need less than 5 min to process (from capturing to detecting) 1 GB of log file, generated from approx. 15.83 GBs of live network traffic. With small log file, the overall detection time can be further reduced to couple of seconds.

Based on the system benchmarks, we conclude that packet capturing is more CPU intensive as compared to log transfer phase. On the other hand, data log transfer consumes way more memory than the data capture phase. Further, the increase in cluster size and parallelism ease out the CPU usage at the Hadoop NameNode. We have observed that capturing of live network traffic incurs the real performance overhead for HADEC in terms of time. In worse case, the capturing phase consumes 77% of the overall detection time. As a future work, HADEC framework may allow potential optimizations to improve the capturing efficiency.

Endnote

[1] There is no silver bullet when it comes to DDoS defense. There are other forms of sophisticated DDoS attacks in the wild like reflected attacks, drive by attacks, resource attacks, flash-crowds, etc. However, this paper focus on designing and developing a Hadoop-based live

DDoS detection framework with flooding attacks as a proof of concept example. This framework will help to build similar systems for other attack types.

Acknowledgements
We would like to thank the CS department of NUCES for providing us the testbed infrastructure. Special thanks to Mr. Talha Shamim Jafri (network manager) for his continuous help and support to keep the lab up and running.

Authors' contributions
Both authors have made substantial intellectual contribution from design, architecture, testbed deployment, evaluations, and writeup of this manuscript. Both authors read and approved the final manuscript.

Authors' information
Sufian Hameed received MSc in Media Information from RWTH-Aachen, Germany, in 2008 and received PHD in Information Security from George August University Goettingen, Germany in 2012. He currently hold a position of Assistant Professor in Department of Computer Science, National University of Computer and Emerging Sciences, Pakistan. His research interests revolve around all aspects of applied Computer Security, with an emphasis on network security, web security, mobile security, and secure architectures and protocols for Cloud and IoTs.
Usman Ali started his education in Computer Science from COMSATS University, Pakistan, from where he received his BS(CS) in the 2012. He did his MS-CS from National University of Computer and Emerging Sciences in 2015. His area of research interest is Machine Learning and Big Data Analysis. He is currently working as Software Engineer in Journey Xp.

Competing interests
There are no non-financial competing interests (political, personal, religious, ideological, academic, intellectual, commercial or any other) to declare in relation to this manuscript.

Publisher's Note

Received: 27 September 2017 Accepted: 12 July 2018
Published online: 31 July 2018

References

1. Dyn cyberattack. www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet. Accessed 09 July 2018
2. Github ddos attack. www.wired.com/story/github-ddos-memcached/. Accessed 09 July 2018
3. Hadoop. https://hadoop.apache.org/. Accessed 09 July 2018
4. Hadoop yarn. http://hortonworks.com/hadoop/yarn/. Accessed 09 July 2018
5. Iperf: network performance measurement tool. https://iperf.fr/. Accessed 09 July 2018
6. Loic: a network stress testing application. http://sourceforge.net/projects/loic/. Accessed 09 July 2018
7. Mapreduce. http://wiki.apache.org/hadoop/MapReduce. Accessed 09 July 2018
8. Mausezahn. https://github.com/uweber/mausezahn. Accessed 09 July 2018
9. Operation Payback cripples MasterCard site in revenge for WikiLeaks ban, dec. 8, 2010. http://www.guardian.co.uk. Accessed 09 July 2018
10. Powerful attack cripples internet, oct. 23, 2002. http://www.greenspun.com/. Accessed 09 July 2018
11. Scapy. http://www.secdev.org/projects/scapy/. Accessed 09 July 2018
12. Secure copy. http://linux.die.net/man/1/scp. Accessed 09 July 2018
13. Tshark: network analyzer. www.wireshark.org/docs/man-pages/tshark.html. Accessed 09 July 2018
14. Yahoo on trail of site hackers, wired.com, Feb. 8, 2000. http://www.wired.com/. Accessed 09 July 2018
15. AA Cárdenas, PK Manadhata, SP Rajan, Big Data Analytics for Security. IEEE Security & Privacy. **11**(6), 74–76 (2013). https://doi.org/10.1109/MSP.2013.138
16. B Claise, Cisco systems netflow services export version 9, rfc 3954 (informational) (2004)
17. PJ Criscuolo, Distributed denial of service: Trin00, tribe flood network, tribe flood network 2000, and stacheldraht ciac-2319 (2000). Technical report, DTIC Document
18. R Fontugne, J Mazel, K Fukuda, in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Hashdoop: A MapReduce framework for network anomaly detection, (Toronto, 2014), pp. 494–499. https://doi.org/10.1109/INFCOMW.2014.6849281
19. J Francois, S Wang, W Bronzi, R State, T Engel, in *2011 IEEE International Workshop on Information Forensics and Security*. BotCloud: Detecting botnets using MapReduce, (Iguacu Falls, 2011), pp. 1–6. https://doi.org/10.1109/WIFS.2011.6123125
20. J Francois, S Wang, R State, T Engel, ed. by J Domingo-Pascual, P Manzoni, S Palazzo, A Pont, and C Scoglio. NETWORKING 2011, volume 6640 of Lecture Notes in Computer Science (Springer, Berlin Heidelberg, 2011), pp. 1–14
21. S Hameed, U Ali, in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. Efficacy of Live DDoS Detection with Hadoop, (Istanbul, 2016), pp. 488–494. https://doi.org/10.1109/NOMS.2016.7502848
22. S Hameed, HA Khan, in *International Conference on Networked Systems (NetSys)*. Leveraging SDN for collaborative DDoS mitigation, (Gottingen, 2017), pp. 1–6. https://doi.org/10.1109/NetSys.2017.7903962
23. S Hameed, UM Ali, On the efficacy of live ddos detection with hadoop. CoRR (2015). abs/1506.08953, arxiv.org/abs/1506.08953
24. S Hameed, HA Khan, SDN based collaborative scheme for mitigation of DDoS attacks. Futur. Internet. **23**(2018)
25. C Kolias, G Kambourakis, A Stavrou, J Voas, DDoA in the IoT: Mirai and other botnets. Computer. **50**(7), 80–84 (2017)
26. Y Lee, W Kang, Y Lee, in *Traffic Monitoring and Analysis, volume 6613 of Lecture Notes in Computer Science*, ed. by J Domingo-Pascual, Y Shavitt, and S Uhlig. A hadoop-based packet trace processing tool (Springer, Berlin Heidelberg, 2011), pp. 51–63
27. Y Lee, Y Lee, in *In Proceedings of The ACM CoNEXT Student Workshop (CoNEXT '11 Student)*. Detecting DDoS attacks with Hadoop (ACM, New York, 2011). Article 7 , 2 pages. https://doi.org/10.1145/2079327.2079334
28. J Mirkovic, G Prier, P Reiher, in *Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP)*. Attacking DDoS at the source, (2002), pp. 312-321. https://doi.org/10.1109/ICNP.2002.1181418
29. J Mirkovic, G Prier, P Reiher, in *Second IEEE International Symposium on Network Computing and Applications (NCA)*. Source-end DDoS defense, (2003), pp. 171–178. https://doi.org/10.1109/NCA.2003.1201153
30. K Park, H Lee, in *Proceedings IEEE Conference on Computer Communications (INFOCOM)*. On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack, vol. 1, (Anchorage, 2001), pp. 338–347. https://doi.org/10.1109/INFCOM.2001.916716
31. K Park, H Lee, On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. ACM SIGCOMM computer communication review. **31**(4) (2001)
32. V Paxson, Bro: A system for detecting network intruders in real-time. Comput. Netw. **31**(23–24), 2435–2463 (1999)
33. A Pras, JJ Santanna, J Steinberger, A Sperotto, in *International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*. Ddos 3.0-how terrorists bring down the internet (Springer, 2016), pp. 1–4
34. S Ranjan, R Swaminathan, M Uysal, A Nucci, E Knightly, DDos-shield: DDos-resilient scheduling to counter application layer attacks. IEEE/ACM Trans. Networking (TON). **17**(1), 26–39 (2009)
35. M Roesch, in *Proceedings of the 13th USENIX Conference on System Administration,* LISA '99. Snort - lightweight intrusion detection for networks (USENIX Association, Berkeley, 1999), pp. 229–238
36. JJ Santanna, et al., in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. Booters – An analysis of DDoS-as-a-service attacks, (Ottawa, 2015), pp. 243–251. https://doi.org/10.1109/INM.2015.7140298
37. H Wang, C Jin, KG Shin.Defense against spoofed IP traffic using hop-count filtering. IEEE/ACM Trans. Networking (ToN). **15**(1), 40–53 (2007)
38. A Yaar, A Perrig, D Song, in *Proceedings of IEEE Symposium on Security and Privacy (IEEE S&P)*. Pi: a path identification mechanism to defend against

DDoS attacks, (2003), pp. 93–107. https://doi.org/10.1109/SECPRI.2003.1199330

39. X Yang, D Wetherall, T Anderson, A DoS-limiting network architecture. SIGCOMM Computer Communication Review. **35**(4), 241–252 (2005). https://doi.org/10.1145/1090191.1080120
40. X Yang, D Wetherall, T Anderson, TVA: a DoS-limiting network architecture. IEEE/ACM Trans. Networking. **16**(6), 1267–1280 (2008)
41. ST Zargar, J Joshi, D Tipper, A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. Commun. Surv. Tutorials IEEE. **15**(4), 2046–2069 (2013)