Distributed Systems

X_400130

Big Lab Exercise

Team Members

Chao Zhang	2619800	yellywong7@gmail.com
Ahmad Yasir Moosavi	2592048	a.y.moosavi@student.vu.nl
Ali Nikouei	2557346	a.nikouei@student.vu.nl
Hsu Hsu	2614921	deanton.ccns@gmail.com
Ibrahim Kanj	2615072	ibrahimkanj@outlook.com

Submitted on
15 December 2017

Table of Contents

Abstract

Many governments put limits on what citizens can and cannot do online. Such practice is known as internet censorship. The focus of this paper is on the distributed version of the Biband[1] system; the system which is capable of inspecting 10,000,000 different websites as well as internet-based services (e.g Skype, Signal, Google Apps) and determine whether they are blocked. The system also displays some distributed features of availability, consistency and fault tolerance which are vital for accurate results. In fact, by analyzing the data in Iran, one could see that 25% of all domains are actually blocked where cities like Tehran and Shiraz get the lion's share of the blockage. By comparing Biband's results with similar existing software out there (OONI[2]), Biband showed more diversity and accuracy. In conclusion, VPN seems to be the only viable option for most Iranian people to access something on the internet[3].

1. Introduction

Access to information is the key to any fundamental social or political change in a given country and internet, as the largest source of information, plays a very crucial role in achieving that goal. However, many countries block information from reaching the public by implementing what is known as "Internet Censorship." Thus, in this paper, we will be trying to address this problem taking the stance that censorship is in contrast with the basic design principle of the internet in being an open and distributed system available for everyone.

In practice, there are similar systems out there that do what our system achieves. Beside Biband one of the remarkable tools would be OONI (Open Observatory of Network Interference) which is a free software capable of detecting censorship, surveillance and traffic manipulation on the internet. It started monitoring internet censorship in 2012 and didn't stop ever since. Another example is ONI (OpenNet Initiative). ONI used a software system (based on a client-server architecture) developed to identify blocked URLs in order to do technical testing of internet filtering.

The system that we will be implementing in this paper is also based on a client-server architecture where we have a group of probes responsible for requesting a list of domains from the servers to check their blockage statuses. The system has also another interface where users (authenticated) and visitors (guests) could send a request to investigate a domain's status. The following sections will describe the system in more detail.

The rest of the paper is structured as follows. **Section 2** introduces the system's background. **Section 3** presents the system design which includes the system's features. **Section 4** presents the experimental results which specify the setup details and procedure. **Section 5** presents a

[1] https://biband.com/
[2] https://ooni.torproject.org/
[3] https://goo.gl/BJYhvS

discussion of our system's trade-offs as well as the main findings of our work. Finally, we summarize our contributions in **Section 6**.

2. Background

The system could be operated to monitor internet censorship globally, but for the sake of this paper, we decided to limit the scope and focus solely on Iran as to have decisive results in the end. As stated before, the system is made up of probes (workers) who are located in different cities having different ISPs and servers including a coordinator server. The servers communicate with the probes by sending them a list of domains, and then the probes send back the status of the domains in order to be saved in a database. The coordinator server is responsible for scaling the system. It turns on a server when the workload gets high, and it terminates another when this workload is low. Moreover, It can balance the workload among available servers. The system also incorporates a backup server for an added security and reliability.

The system incorporates the following distributed features:

Scalability: The system is designed to be scalable. For this purpose, the number of servers in the system is made dynamic. The coordinator server receives continuously other servers' workload, and it can decide on whether a server needs to be added, an existing server needs to be terminated, or the workload needs to be balanced. There are three optional methods that could be used to select a coordinator, Ring, Token, and Bully method. The "Bully"[4] method is best suited for our system, and this will be explained in more details in the design section.
Fault-tolerance: The system can handle requests at all times. In case one server crashes, the other servers will handle the requests and the system will continue to operate. In the design section, we provide some solutions by which the system recovers in case any other component of the system fails.
Security: To protect the system against the security risks, some measures will be applied. These measures include data sanitization to protect against SQL injection, firewall, antivirus to detect and block malware, encryption, authentication, etc.
Consistency: To apply consistency, the method read-one-write-all (ROWA) is used. Protocols based on ROWA quorums maintain consistency by reading one copy of the data and writing all copies.
Replication: For data replication we use MySQL. We chose this solution over the others because MySQL does not need a plugin or rely on using triggers to collect changes. Thus, ridding our system from the write amplification involved in trigger-based solutions.

[4] Maarten van Steen and Andrew S. Tanenbaum, Distributed Systems (2017), 330-331.

3. System Design

3.1. System Overview

In this part of the paper, we tend to familiarize the reader with the different components of our system. The system is composed of four main components: the servers, probes, APIs, and clients (user & visitors).

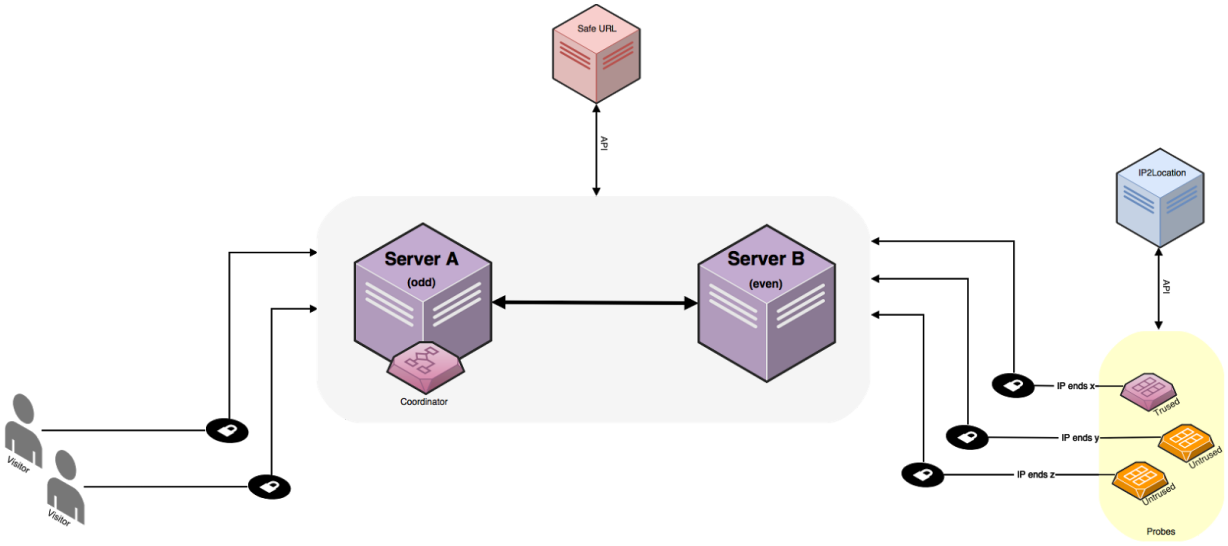


Figure 1 - System Architecture

The servers are responsible for preparing the domains, sending them to the probes and inserting domain's results into the database. As mentioned earlier, one server is the coordinator. In addition to the initial role of the coordinator in balancing the workload of the different servers, the coordinator server is responsible for preparing the protocol (configuration file) between the server and probe. Primarily deciding what server should a probe connect to and how many domains should be tested.

Probes are pieces of code that run on any machine (Windows, Mac, and Linux) and connects to the servers in order to check a list of domains. We recognize two sets of probes: trusted and untrusted probes. Trusted probes are machines that we have access to, while untrusted probes are regular users joining the system (running the piece of code).

Our system incorporates two types of APIs. One API checks the probe's location and makes sure it's in our designated search region or whether a VPN is being used. The other API checks the safety of the domains as we are interested in whether malicious domains are blocked or not.

Before describing who our clients are, It's better to explain how our servers and probes communicate. When a probe goes online, it sends a "POST" request to the location API. In case the location is different than what we expect, or if the probe is using a VPN, an error message will be shown, and the probe sends another request after 30 seconds. This loop will keep running until the probe terminates or gets it location figured. In case the location API was successful, the probe

will send a request to get a server based on the last digit of its IP. If it is an odd number, an odd server (server A) will be responsible for this probe. Otherwise, an even server would be selected (server B). This will be better explained in the scalability section (3.2.2) below in case more than 2 servers are available. After a server is selected, the server will register the probe by adding its unique ID and opening an SSL connection to it.
Then the server sends the probe a list of domains which gets checked for category and blockage status and sent back the result to the server. At this stage, the server will do validation checks (security checks) as a precautionary step before saving to the database. In case of any suspicious results, a flag will be risen, and a trusted probe repeats the test.
The clients in the system are users (authenticated) and visitors (guests). They send a request concerning a domain or a list of domains of a certain category. Similar to the server-probe communication, a server will be selected to answer the client's request. In case the query was already in the database, the result will be sent back to the client otherwise an email will be sent to the client when data becomes available. In the latter case, the coordinator will choose a server (a server which is under the lowest workload) to prepare the domain/list of domains and add it to the database of unfinished tasks.

3.2. System Features

3.2.1. Fault-tolerance

For each component available in the system, some fault-tolerance measures are applied.
Server: If a server crashes, then another server will be responsible for handling the requests. To find if a server is down, all servers, every couple of minutes, check the availability of all predefined servers and if the coordinator is down, then they hold an election using the bully algorithm. The (new) coordinator will update the list of online servers and define a server for each probe. The method by which a server gets selected will be explained in the scalability section.
Coordinator: The coordinator is selected using the Bully Algorithm. There is a main coordinator. If it fails, the next server will be assigned as the new coordinator. When the main coordinator comes online again, it gets its role back by sending a broadcast message to all the servers including the current coordinator.
API: If one of the location APIs does not respond, the other API (SafeURL) can take over its job. In case that none of the servers work, the probe will wait for 30 seconds and repeat the request. On the other hand, if the API for the SafeURL is not responding, the system will not be affected.
Probe: In case a probe disappears for any reason, the system will not be affected, because the server registers the result coming from the probe as soon as it receives it. When the probe actually becomes available again, it can get a new list from the server and check it normally.

3.2.2. Scalability

The system can be scaled up or down based on the workload on the system. The coordinator can add or remove servers to distribute the workload on the system. Also, each server has a queue which can keep up to 250 tasks and pass it to each worker. In total, there are 5 workers for each

core of CPU of a server which means that each core can keep 1250 tasks every moment and pass it one by one to the worker. This stack helps to manage the scalability.
On the other hand, for balancing the workload, the last digit of the integer IP will be taken into account. There will be a default list of the distributed workload as it is mentioned below.

```
{'0':'S1', '1':'S1', '2':'S1', '3':'S1', '4':'S1', '5':'S1', '6':'S1', '7':'S1', '8':'S1', '9':'S1'}
```

The coordinator based on the resource usage and specification of servers decide on how to distribute the workload among servers. For example, if there are only two servers -say S1 and S2- with same specifications, the first distribution will be something like the following.

```
{'0':'S1', '1':'S2', '2':'S1', '3':'S2', '4':'S1', '5':'S2', '6':'S1', '7':'S2', '8':'S1', '9':'S2'}
```

If the workload on S1 is more than the workload of S2, then, the workload on S2 should be more than S1 to balance the workload. One of the possible distribution will be the following.

```
{'0':'S1', '1':'S2', '2':'S2', '3':'S1', '4':'S2', '5':'S2', '6':'S1', '7':'S2', '8':'S2', '9':'S1'}
```

If the workload is not balanced yet, the coordinator will send more workload to S2.
The same strategy applies when the number of servers is more than two. In Appendix 1, there is a code snippet which is responsible for the decision making in balancing the workload by either adding a server or sending the workload to the next available server by which the following formula is derived:

```
# Calculate the workload of a server
load = ((
       server_status[key]['cpu'] +
       (2 * server_status[key]['mem'])) /
       (3 * HARDWARE_FACTOR[key]))
```

To justify the formula, besides the assumptions[5], we checked all the servers we have and found out that the memory is more engage rather than CPU, then we gave it the coefficient of 2 and 1 to the CPU (these two are percentages) and add them together. To have a rationale percentage as the result of this calculation, we needed to divide it by 3, but there was another metric, HARDWARE_FACTOR which was required to affect the formula. We divide the equation by this metric to have the final result. The latest metric is a float number between [1,9], the better hardware specification and the less workload, the greater number. Then if the server has 4GB RAM and 4x3GHz CPU with 100% free workload in comparison with the same specification but workload of 60% the HARDWARE_FACTOR of the latter will be smaller than the first one, as the result of the equation for the first one will be greater than the second one.

3.2.3. Security

The following is the list of security measures applied in the system:

[5] We tried to find a standard formula for calculating the performance of a system but we didn't find anything. However, we found 3 engaged metrics in it: memory, CPU and the specification of the system plus the workload on the server besided Biband's workload. The coordinator gets the resource usage of each system including CPU and memory; we defined a static variable called HARDWARE_FACTOR as the metric of the system specification and the workload apart Biband.

Encryption: All the connections between the system components are encrypted to ensure secure connections.
Authentication: Making sure that the login process is secure and not susceptible to being bypassed.
Auditing the system frequently: Continuously supplying the system with testing domains with previously known results and checking that the servers are working perfectly and not storing faulty data in the database.
Firewall and Antivirus: The system actually has a Firewall and antivirus to secure and quarantine the system against attacks.

3.2.4. Replication

The system replication is managed through MySQL[6] which is an open-source relational database management system (RDBMS). There are other options like Slony which requires a trigger and actually writes the data twice. However, Slony is difficult to manage and, operate and that is why it did not qualify for this project. Bi-Directional Replication was our first choice, but MYSQL does not need any plugin for replication. Also, MySQL can replicate all changes which is the safest way for replication. Another advantage of MySQL is that of fewer data getting written to log files.

3.2.5. Consistency

To apply consistency, ROWA (Read-one-write-all)[7] method is used for this project. It simply means that reading can be done from any replica, and writing must be done on all replicas. In case, there is write-write conflict, ROWA can handle it by using a Vector Clock. Therefore, if there are two or more concurrent writes, the one which has been executed first, will be applied and committed to the servers, and the others will be aborted.

3.3. Additional Features

3.3.1. Benchmarking

To manage benchmarking in our system, we decided to use the Citizenlab's[8] list which is flagged as censored. It is also being frequently checked by OONI (mentioned in Section 1). Citizenslab's list includes hundreds of domains which are randomly selected and checked in different cities. By putting our system to this test and having it run in two cities (Tehran and Shiraz), we wanted to see whether our system could provide more accurate data than OONI. The results came astounding where our system showed that 72% and 69% of domains were blocked and 28% and 31% were not actually blocked respectively. As for OONI, 100% of the domains were actually HTTP blocked in both cities. There is a result sheet of domains in appendix 2 that might give a clearer view.

[6] https://www.mysql.com
[7] https://digital.lib.washington.edu/researchworks/handle/1773/6905
[8] https://github.com/citizenlab/test-lists/blob/master/lists/ir.csv

4. Experimental Results

4.1. Experimental Setup

To setup the system, it is necessary to prepare several servers, probes, and APIs. To do so, a total of 5 servers have been configured. One of the servers is located in the United States, and the other four servers are in Lithuania. One of the servers was selected as the main coordinator. As mentioned before, there are two types of probes; trusted, and untrusted. The trusted probes are already registered in the system. The probes are mostly located on systems which use Windows and Linux operating systems. Finally, the location and SafeURL APIs are integrated into all the servers.

The system is designed to be able to handle a workload of 200 users per hour. However, the actual workload on the system will be much less than the mentioned amount. According to 3.2.2. Scalability, each core of a server can handle up to 1250 tasks simultaneously. Also, the workload plan is explained in details in (3.2.2. Scalability) which shows how the system manages the workload.

In order to conduct the experiment, a piece of code (Appendix 3) has been generated to simulate the number of probes which communicate with the servers. The piece of code can work with the modified version of Biband which had a predefined set of IPs to simulate the last digit of IPs between 0 and 9. This way, it can simulate the workload to show if the system can handle it or not.

To add a new server, the latest version of services, configurations, and scripts must be cloned. For this purpose, the backup server is used to copy all the aforementioned components to a new server. This procedure is done automatically and takes 10 to 15 minutes. Then, the database on the new server needs to be updated which takes 2 to 3 minutes. Therefore, the whole procedure will take less than 20 minutes. To save time, there are servers used for other purposes such as web and database services. However, they are pre-configured to be added to the system when needed. In this case, we can save 10 to 15 minutes and the whole procedure will not take more than 3 to 5 minutes.

4.2. Experiments

To implement the experiments, a total of 10264 domains have been checked since 2 December till 7 December 2017. Table 1 shows the statistics for the mentioned period. Also, the resource usage graph is included in Appendix 4.

Total Number of Domains	**10264**
DNS Blocked	2.41%

HTML Blocked	24.93%
TCP Blocked	0.93%

Table 1 - System statistics for the period 2 December till 7 December 2017

It shows that there was no significant change in the resource consumption before, during, and after the test. It is worth mentioning that this specified server was the mail server, as well as the webserver beside being the coordinator. The peak in the CPU usage is related to the scalability test by simulation that we ran, and it was mitigated by the coordinator by adding a new server.

4.2.1. Scalability

To test the scalability feature of the system, we have tested it for 2 servers and 300 requests, and it was able to handle it easily. A piece of code (Appendix 3) has been generated to simulate two types of workload on the system:

Load increase on a single server: It was achieved by increasing the number of users who have the same number at the last digit of their IP. In this case, the system worked properly and could handle the workload when IPs of the probes were not distributed normally.

Load increase on the whole system: It was achieved by increasing the number of users through adding more number of probes (value of the variable “n” in the code). In this case, the system could handle the overall workload by adding new servers. Also, the system utilized the queue system which has been explained in details in 3.2.2. Scalability.

4.2.2. Fault-tolerance

Three scenarios were designed to test the fault-tolerance of the system:

Scenario 1 (Server shutdown): In this scenario, server 2 was shut down while the system was working. In less than 5 minutes, the coordinator found out that the server was not running and changed the configuration file to update and send it to the probes. If the broken server was the coordinator, after holding an election, the new coordinator would update the configuration file and send it to all probes.

Scenario 2 (The responsible server did not receive the probe’s message): In this scenario, when the probe received the list of the domain, we ran the firewall and blocked the address of all the servers. The probe tried to send the result to the server, but, it was unreachable. It tried 5 times with the delay of 90 seconds between each try. Afterwards, it dropped sending the result and contacted the coordinator for the new list of domains.

Scenario 3 (Probe shut down): The probe was disconnected from the Internet. The server did not react, because it did not wait for the response from the client.

4.2.3. Security

System security was tested and ensured using the following measures:

SQL Injection: A probe tried to inject SQL command into the database. As the inputs are sanitized and validated[9] before adding to the database, the SQL injection could not happen, and input was not added to the database.
Firewall, IP filtering, and Antivirus: All servers use a firewall, IP filtering, and antivirus to ban malicious access to the server and detect them in case of being successful.
DDoS attack: In this case, a probe sent 6 requests in less than a minute. As a result, the probe was blocked temporarily. The probe tried to send 6 requests in less than a minute for several times. Each time, the probe was blocked and the ban time was increased.

4.2.4. Consistency and Replication

For consistency, we relied on MYSQL and its built-in features for replication. Therefore, we just inserted something into the database of one of the servers, and after an atomic period of time, we saw the update on other servers.

5. Discussion

Due to the time constraints of this paper and the low number of Iranian probes that are configured to work with the system, the expected findings behind distributed Biband would hopefully see daylight in the near future. This could give hints on the types of censorship being implemented in Iran and in which cities. Future results could also help in figuring out the big questions like whether the censorship in Iran is a centralized or a decentralized system. Are news websites, e.g., CNN, BBC, and social media websites, e.g., Youtube, Facebook, and Twitter, blocked due to their popularity and massive role in shaping the public opinion or is the blockage system generally targeting the categories of these websites? With the small list of domains tested, the system has shown that the rate of censorship in Iran is around 26%. Actually, the system is proving to be more accurate than the other existing software out there. By checking 100 websites that OONI claims to be blocked (HTTP blocked), the results came shocking as 30% of the websites were not actually blocked, 50% HTTP blocked, 9% TCP blocked and 11% DNS blocked.
The design of a distributed Biband systems was not free of some limitations in the Biband system itself. A major trade-off would be the internet speed. As in Iran, the highest download/upload speeds one could have is 128 Kbps[10]. Many activists during President Ahmadinejad's mandate expressed their frustration with such low speeds as they considered that such speeds stand as a hurdle in the flow of information. It was only recently that the Iranian government increased the speed of the internet to 50 Mbps[11] which increased the average bandwidth to 1.5 Mbps[12]. Therefore, the design of the probe came with bandwidth in mind as to minimize the amount of upload/download needed to reduce the costs. The other trade-off in the system might be the blockage. If the Iranian government decides to block the Biband's servers, then the system would

[9] http://download.oracle.com/oll/tutorials/SQLInjection/html/lesson1/les01_tm_ovw3.htm
[10] https://goo.gl/JJQhQF
[11] https://goo.gl/o3fHps
[12] https://goo.gl/95DbYg

be out of work. However, a countermeasure to such action is probably moving the service to the cloud.
Having these trade-offs aside, The system showed huge improvements upon the addition of up to five servers. For more than a year starting on August 2016, Biband was running on a single server. It checked approximately 1,400,000 websites with around 50 connected probes. The main problem was not that of communication between probes and server but that of data on servers. The huge amount of data that needs to be processed, validated and saved to the database raises a red flag. If the number of probes increases, the amount of data will increase as well. Therefore changes, like migrating to the cloud database, need to be added to the system to handle the excess data.

6. Conclusion

In conclusion, the paper had shed light on the different distributed components (consistency, replication, security, fault-tolerance, and scalability) that are important for the availability and performance of the Biband system as an internet censorship monitoring tool in Iran. The results from such a system would help in determining the basis on which censorship is being applied and would hopefully play as a pressure card toward an internet free from such blockage. As for the future works that might be part of the Biband system, perhaps a mobile client is what is next in line. Everyone these days has a mobile phone; thus if Iranians really want to reject the censorship, then they could start by turning their mobile phones into probes as to let circumvention tool developers have a better insight about the censorship technology in Iran. Otherwise, VPN would seem to be their only option against this censorship keeping in mind that the Iranian government pursues VPN blocking as well.

Appendixes

Appendix 1

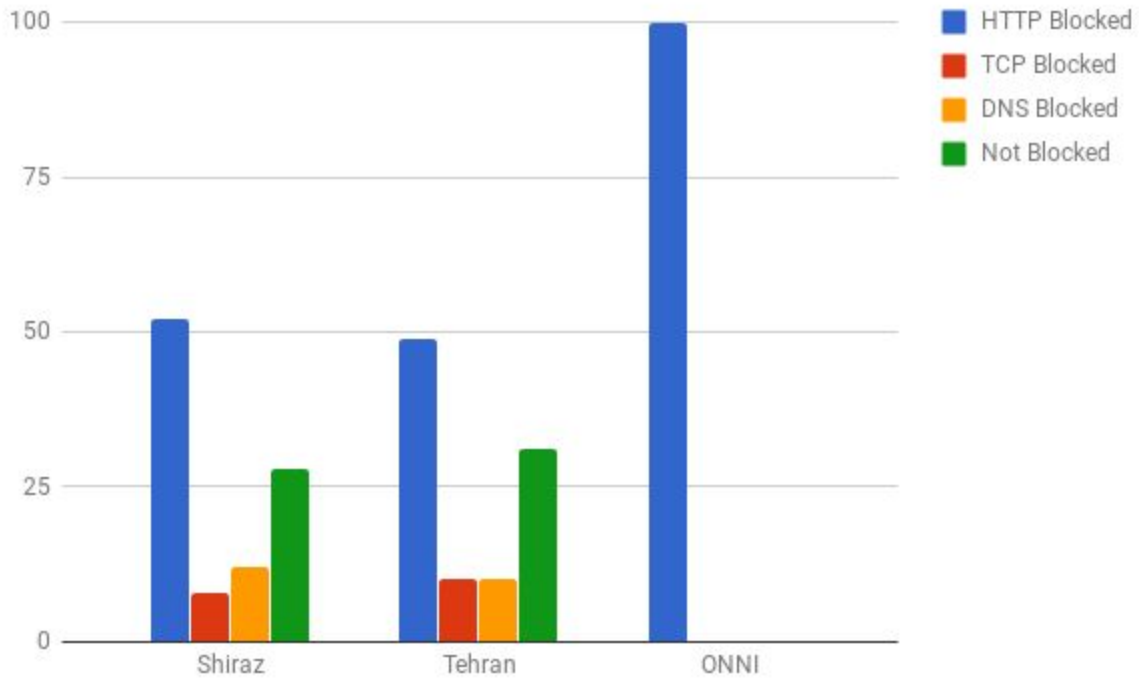
Part of the coordination code which calculates the workload on a server and if it is more than the limitation, balances the workload among available servers.

```
# Calculate the workload of a server
load = ((
        server_status[key]['cpu'] +
        (2 * server_status[key]['mem'])) /
        (3 * HARDWARE_FACTOR[key]))

# Balances the workload
if load <= LIMIT:
        online_servers.append(key)
        online_servers.append(key)
elif load >= LIMIT:
        online_servers.append(key)
```

Appendix 2

Comparison of the blocked sites checked by OONI and the check by Biband in two different cities in Iran, Shiraz and Tehran.

City	HTTP Blocked	TCP Blocked	DNS Blocked	Not Blocked
Shiraz	52	8	12	28
Tehran	49	10	10	31
ONNI	100	0	0	0



Row	Domain	Shiraz	Tehran	ONNI
241	mohajerani.maktuob.net/	blocked	blocked	blocked
647	www.iranrights.org/	blocked	blocked	blocked
880	www.zard-aloo.blogspot.com	blocked	blocked	blocked
391	www.al-madina.com/	blocked	blocked	blocked
674	www.kayhanlondon.com	blocked	blocked	blocked
263	openasia.org/	blocked	blocked	blocked
52	bahai-education.org	blocked	blocked	blocked
968	www.linay.com	blocked	blocked	blocked
548	www.ghoghnoos.org	blocked	blocked	blocked

942	www.mp3.com	blocked	blocked	blocked
943	www.multiproxy.org	blocked	blocked	blocked
176	ir.mondediplo.com/	blocked	blocked	blocked
216	linkestane-ggbg.blogspot.com	blocked	blocked	blocked
457	www.bbc.co.uk	blocked	blocked	blocked
397	www.alburhan.com	blocked	blocked	blocked
681	www.khosoof.com/	blocked	blocked	blocked
730	www.niabati.tk	blocked	blocked	blocked
484	www.darsahn.org/	blocked	blocked	blocked
344	tehranreview.net/	blocked	blocked	blocked
426	www.anbacom.com/	blocked	blocked	blocked
958	www.zoomshare.com	blocked	blocked	blocked
669	www.kaafaraaneh.blogspot.com	blocked	blocked	blocked
6	aadaamaak.blogfa.com	blocked	blocked	blocked
259	nurizad.info/	blocked	blocked	blocked
36	aqueeerdiaries.blogspot.com	blocked	blocked	blocked
889	zane-azad.blogspot.com	blocked	blocked	blocked
938	www.lesbian.org	blocked	blocked	blocked
525	www.femirani.com	blocked	blocked	blocked
914	spankwire.com	blocked	blocked	blocked
941	www.meetic.com	blocked	blocked	blocked
799	www.sazegara.net/persian/	blocked	blocked	blocked
471	www.cbs.gov.il	blocked	blocked	blocked
887	zahrahb.tumblr.com	blocked	blocked	blocked
382	www.aghaghi.com	blocked	TCP blocked	blocked
282	pjhouse.blogspot.com	blocked	blocked	blocked
222	mangoosaleam.blogspot.com	blocked	blocked	blocked
178	iran.mojahedin.org	blocked	blocked	blocked
716	www.muslm.net/vb	blocked	blocked	blocked
962	masexualite.ca	blocked	blocked	blocked
787	www.rferl.org/featuresarchive/country/iran.html	blocked	blocked	blocked
83	cafeneveshteh.com/	blocked	TCP blocked	blocked
726	www.newsagency-ina.com	blocked	blocked	blocked

81	brick.blogfa.com/	blocked	blocked	blocked
338	sverigesradio.se/sida/default.aspx?programid=2493	blocked	blocked	blocked
86	ch1.cc/	blocked	blocked	blocked
820	www.sigarchi.net	blocked	blocked	blocked
17	alarabiya.net	blocked	blocked	blocked
247	nesvan1.wordpress.com/	blocked	blocked	blocked
834	www.sunni-news.net	blocked	blocked	blocked
280	photomohammad.persianblog.com	blocked	blocked	blocked
218	lunashadzi.wordpress.com/	blocked	blocked	blocked
146	gistela.blogspot.com/	blocked	not blocked	blocked
762	www.phoenix-dg.com	DNS blocked	DNS blocked	blocked
367	wordpress.com	DNS blocked	DNS blocked	blocked
253	nikpress.com/	DNS blocked	DNS blocked	blocked
133	farzan.envy.nu	DNS blocked	DNS blocked	blocked
298	realm.io/	DNS blocked	DNS blocked	blocked
847	www.theworld.org/2011/09/the-consequences-of-irans-lake-orumiyeh-drying-up/	DNS blocked	DNS blocked	blocked
549	www.glwiz.com/homepage.aspx	DNS blocked	DNS blocked	blocked
69	bintray.com/	DNS blocked	DNS blocked	blocked
742	www.omilani.netfirms.com	DNS blocked	DNS blocked	blocked
511	www.emc.com/en-ca/index.htm	DNS blocked	DNS blocked	blocked
629	www.ifttv.com/	DNS blocked	not blocked	blocked
100	developer.atlassian.com/	DNS blocked	not blocked	blocked
191	islamabadtimes.ir/	not blocked	not blocked	blocked
43	audiojungle.net/	not blocked	not blocked	blocked
91	codecanyon.net/	not blocked	not blocked	blocked
765	www.polymer-project.org/	not blocked	not blocked	blocked
507	www.eghtesadnews.com/	not blocked	not blocked	blocked
33	anidalton.blogfa.com	not blocked	not blocked	blocked
65	berooztarinha.com/	not blocked	not blocked	blocked
170	honarkhan.com/mag/	not blocked	not blocked	blocked
722	www.nazarnews.com/	not blocked	not blocked	blocked
763	www.playmapscube.com/	not blocked	not blocked	blocked

193	java.com	not blocked	not blocked	blocked
585	www.google.com/search?hl=fa&q=%D8%AC%D9%84%D9%88%DA%AF%DB%8C%D8%B1%DB%8C+%D8%A8%D8%A7%D8%B1%D8%AF%D8%A7%D8%B1%DB%8C&btnG=%D8%A8%D9%8A%D8%A7%D8%A8	not blocked	not blocked	blocked
482	www.d-alsonah.com/vb	not blocked	not blocked	blocked
542	www.geocities.com/ahangark/	not blocked	not blocked	blocked
417	www.alsonnhway.com	not blocked	not blocked	blocked
431	www.apple.com/itunes/	not blocked	not blocked	blocked
266	p30download.com	not blocked	not blocked	blocked
700	www.mizanonline.ir/	not blocked	not blocked	blocked
158	groups.yahoo.com/group/room_cekaf_chat/	not blocked	not blocked	blocked
274	persianbasket.com/	not blocked	not blocked	blocked
576	www.google.com/search?hl=fa&q=%D8%A7%D8%B3%D9%84%D8%AD%D9%87&btnG=%D8%A8%D9%8A%D8%A7%D8%A8	not blocked	not blocked	blocked
80	boyernews.com/	not blocked	not blocked	blocked
566	www.google.com/search?hl=ar&q=Khafaghan&btnG=%D8%A5%D8%A8%D8%AD%D8%AB%21&lr=	not blocked	not blocked	blocked
754	www.pcparsi.com/	not blocked	not blocked	blocked
550	www.godaddy.com/	not blocked	not blocked	blocked
575	www.google.com/search?hl=fa&q=%D8%A7%D8%B3%D8%AA%D9%82%D9%84%D8%A7%D9%84&btnG=%D8%AC%D8%B3%D8%AA%D8%AC%D9%88%D9%8A+Google	not blocked	not blocked	blocked
92	coreos.com/	not blocked	not blocked	blocked
807	www.shamela.ws	not blocked	not blocked	blocked
954	www.weedfarmer.com	TCP blocked	TCP blocked	blocked
808	www.sharareha.com	TCP blocked	TCP blocked	blocked
910	pridetube.com	TCP blocked	TCP blocked	blocked
210	kodoiin.blogspot.com	TCP blocked	TCP blocked	blocked
411	www.alqadisiyya3.com	TCP blocked	TCP blocked	blocked

911	proxpn.com	TCP blocked	TCP blocked	blocked
671	www.kaleme.com	TCP blocked	TCP blocked	blocked
641	www.iranhumanrights.org	TCP blocked	TCP blocked	blocked

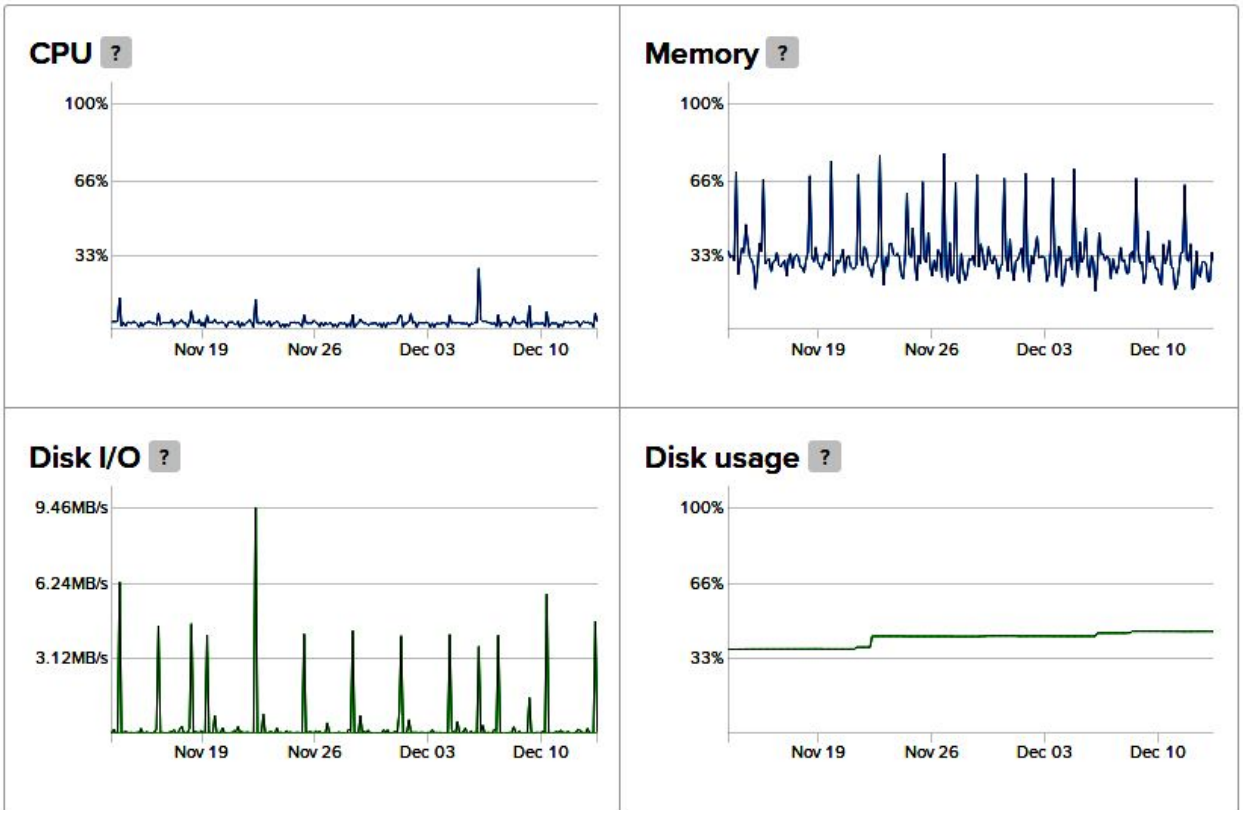
Appendix 3

Part of the scalability code produced the workload on the system.

```
#!/usr/bin/python

import subprocess
import time

try:
	n = input("how many clients do you want to simulate");

	p_list = []

	for i in range(0,n):
	print "create probe" + str(i)
	j = i % 10
	p = subprocess.Popen(["python3","biband-client-dev/app{}.py".format(j)])
	p_list.append(p)

	while True:
	is_break = True
		for p in p_list:
			if p.poll() is None:
			is_break = False
		time.sleep(1)
		if is_break:
			break;

except KeyboardInterrupt:
	print "the program is terminated"
```

Appendix 4

Resource usage from 12 November till 12 December 2017.



Appendix 5

Distributed Biband	Biband	Total
Total_Time = 57 h Think_Time = 10 h Dev_Time = 15 h XP_Time = 1.5 h Analysis_Time = 0.5 h Write_Time = 10 h Waste_Time = 30 h	Total_Time = 190 h Think_Time = 30 h Dev_Time = 50 h XP_Time = 10 h Analysis_Time = 100 h Write_Time = 0 h Waste_Time = 0 h	Total_Time = 247 h Think_Time = 30 h Dev_Time = 65 h XP_Time = 11.5 h Analysis_Time = 100.5 h Write_Time = 10 h Waste_Time = 30 h

Appendix 6

The links to the project (MIT Licence) on Github and the explanation about each.
https://github.com/baaroo/biband
https://github.com/baaroo/biband-coordinator
https://github.com/baaroo/biband-client

Details about the piece of code in Probe.

```
.
├── app.py
├── CHANGELOG.md
├── conf.json
├── LICENSE
├── modules
│   ├── coordinator.py
│   ├── dns.py
│   ├── domains.py
│   ├── category.py
│   ├── getip.py
│   ├── getloc.py
│   ├── html.py
│   ├── __init__.py
│   ├── log.py
│   ├── result.py
│   └── tcp.py
├── README.md
└── requirements.txt
```

The app.py is the application which consists of several modules. It connects to IP2Location (getip.py and getloc.py) to get the location of the user. If the user is not in Iran, then it will send a GET request to the server to get the list of 10 domains. If the user uses VPN or it is not located inside Iran, it will send an error message.
To get the list of domains, it firstly checks who is the coordinator (coordinator.py). It sends its request to one of the predefined servers in the configuration file (conf.json). If one of the servers is down while it is sending its requests, it goes for the next one. When it finds the available server, that server sends the protocol of connection which is based on the integer IP of the probe. This protocol is defined by the coordinator. Coordinator.py sends a GET request to the coordinator and get the available servers as well as the protocol.
After knowing to which server it should connect to, it sends another GET request to that server to get the list of the domains and start checking them (domains.py). It will check if the blockage type is DNS blocking (dns.py), if it is TCP blockage (tcp.py) or if it is the manipulation of the page (html.py). The last one will be the check for the categories (category.py) if it is asked by the coordinator to check it.
After doing all tests, it will send the result to the server (result.py) and waits for the next round of the domains to do the test.
If the server gets down during this process (will not take more than 3 minutes), the probe waits for 90 seconds, and will resend the result. If it happens 5 times, it will drop it and start from the beginning which means asking the coordinator about the online servers and the server it should connect to.

Code of the Coordinator

```
.
├── coordinator.py
```

```
├── README.md
├── requirements.txt
└── settings.py
```

The coordinator does 2 jobs (coordinator.py); firstly it should defines which servers are online, and secondly, it should balance the load of the system among available servers.
Firstly, it gets a token from the server for the sake of security; it is done by sending a POST request to one of the servers. It helps to insert the list of the server into the database which is replicated then the latest list of the servers will be available on all servers. It helps the system when the coordinator is down for any reason.
At the beginning all the system are the same, to find the coordinator they will hold an election by the bully method and the winner will be the coordinator. Every couple of minutes (say 10 minutes) they will do the test to see if the coordinator is available or not, in this case they will hold another election to find the coordinator.
From now, all parts are done in the coordinator:
To find the available servers it requests each of them an SSH request (port 22) to see if it is open or not. If it is open, it means that the server is running. It will update the list of the available servers, and flag the server as 1 and if the server is not available it will be flagged as 0. After getting the list of available servers it starts asking the resource usage from each and submit the average of it in the list.
The other important factor is the specification of each server which is defined in settings.py. There is a variable which says what the configuration of the server (HARDWARE_FACTOR) is.
By having this variable and the resource usage, the coordinator can make decision on how to balance the load of the system by calculating the workload:

```
( Average of CPU usage + 2 x Average of memory usage ) / ( 3 x HARDWARE_FACTOR )
```

It helps coordinator to define the lost of available servers and the balanced workload. The output will be something like this which shows the available servers and a protocol to balance the workload based on the last digit of the IP.

```
{'0':'S1', '1':'S2', '2':'S2', '3':'S1', '4':'S2', '5':'S2', '6':'S1', '7':'S2', '8':'S2',
'9':'S1'}
```

This test will be done every couple of minutes (say 10 min)

Code of the servers

This part is based on flask and contains the website as well.

```
.
├── __init__.py
├── models
│   ├── categories.py
│   ├── domains.py
│   ├── ip2location.py
│   └── results.py
├── resources
│   ├── domains.py
```

```
│   ├── measurements.py
│   └── results.py
├── validations.py
└── views.py
```

The main activity of the server is related to the API folder. It is responsible for categorizing the domains. Validating the results received from the probes. Inserting the results after validation, checking the security metrics of each domain, and prioritizing the domains to be sent to probe based on different flags (category, priority, number of the times it is checked, etc.).