SiegeBreaker : Multi-Client Support

Himanshu Sagar(2014046)

BTP report submitted in partial fulfillment of the requirements
for the Degree of B.Tech. in Computer Science & Engineering
on Nov 26, 2018

BTP Track: Research

BTP Advisor
Dr. Sambuddho Chakravarty

Indraprastha Institute of Information Technology
New Delhi

Student's Declaration

I hereby declare that the work presented in the report entitled **"SiegeBreaker: Multi Client Support"** submitted by me for the partial fulfillment of the requirements for the degree of *Bachelor of Technology* in *Computer Science & Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under guidance of **Dr.Sambuddho Chakravarty** . Due acknowledgements have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

..... **Place & Date: New Delhi, 7 July, 2018**
Himanshu Sagar

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

..... **Place & Date: .....**
Sambuddho Chakravarty

Abstract

Decoy Routing, the use of routers (rather than end hosts) as proxies,shows great promise as an anti-censorship mechanism. To use a Decoy Router, the user sends specially crafted packets, apparently to an uncensored website. En route, the packets encounter the Decoy Router (beyond the network boundaries of the censor), which identifies them (using a covert cryptographic handshake), decrypts their content, and proxies them to their true destination. How-ever, Decoy Routing requires routers able to perform complicated operations (detecting secret handshakes, decrypting packets, etc). This requirement is a major challenge: commercial routers are limited in flexibility, and existing Decoy Router implementations (on commodity servers) are unsuitable for carrier-grade deployments. SiegeBreaker is a practical Decoy Routing system on an SDN architecture, and divides the responsibilities for Decoy Routing among three entities the SDN switch that simply forwards packets, the SDN controller that identifies the secret handshake, and a hidden proxy server to which the switch eventually forwards the clients' request. However SiegeBreaker didn't have any support for Multiple Clients. My major contributions were towards a significant aspect of any deployable system i.e its ability to scale well. They include - support for multiple clients at proxy, module which allows simultaneous access to all websites, performance improvements, enforcing fair share policy among fellow SiegeBreaker/TCP connections.

Keywords: SiegeBreaker, Decoy Routing, SDN, Cryptography , Routers, TLS Handshakes.

Acknowledgments

I would like to express my gratitude towards my supervisor Dr. Sambuddho Chakravarty and his PhD students - Mr. Piyush Sharma and Mr. Devashish Gosain for their guidance and encouragement throughout the project. They motivated me to put in my best and responded to my queries and questions promptly. I will forever be grateful to them for what i learned with their support. Moreover, this report is derived from already submitted version of SiegeBreaker. I also acknowledge contribution of other authors of paper in writing it.

Contents

Chapter 1

1.1 Introduction

Censorious regimes block, filter, redirect, intercept, or even modify traffic between clients on their network. Circumvention tools like proxies, tor etc have faced several countermeasures from censors. We use Software Defined Networking, to implement practical Decoy Routing system - SiegeBreaker. My work was to add multi client module in SiegeBreaker on both ends - proxy and client. This enables proxy to address requests from multiple client. Yet another enhancement was made in client side, which enables him to access multiple websites in single client. These enhancements required incorporation of libpcap in entire SiegeBreaker code-base.

1.2 Background Information

1.2.1 Software Defined Networking (SDN)

Software Defined Networking, an emergent network design paradigm that decouples control plane and data plane functionalities. SDN architectures involve a centralized controller, which controls the operation of the (multiple) switches in the network (usually, by sending commands over a standard protocol named OpenFlow).In an SDN architecture, complex operations (such as computing network-wide control policies and implementing policy decisions for the network) are the responsibility of the controller. The policies,in the form of flow tables, are installed on the network switches simple, dedicated devices, which forward traffic based on these tables.

1.2.2 Decoy Routing

Decoy Routing is an anti-censorship mechanism, that uses friendly routers (Decoy Routers or DRs), in the Internet, as proxy servers.Like any other proxy servers, Decoy Routers allow users in censorious countries to access filtered network destinations; but being routers, they are very difficult to block

Chapter 2

Motivation and Research Problem

The objective of this project is to add multi client support to already existing system of SiegeBreaker, thereby increasing the scalabity of system. Challenge is to

1. Ensure clients should also have the ability to browse multiple websites simultaneously.
2. Simulate TCP fair share policy, hence properly share bandwidth among fellow SiegeBreaker/TCP connections.
3. Exploit good fraction of available bandwidth, thereby achieving TCP-like transfer rates.
4. Ensure portability and usability of system, while selecting modules and software as dependencies.(We selected libpcap).

Motivation behind this research problem is to pitch in, an significant aspect of any deployable system i.e it's ability to scale well while avoiding any unrealistic hit on performance.

Chapter 3

Work Done

3.1 SEMESTER 1

3.1.1 Central Manager

Idea: It is to replace Hidden Proxy with Central Manager/Proxy which which spawns new threads for each connection. Raw sockets tend to listen to entire traffic associated with system, we needed raw sockets for crafting packets. This creates havoc because each raw socket is listening to lost of irrelevant traffic and Hidden Proxy server was running on fixed port(say 443). The need was to channel traffic of all connections, to their respective threads and NOT let all raw sockets read entire traffic. Hence, we decided to keep a raw socket per connection and divert traffic associated with a connection to it's corresponding another(random) port, Now raw socket need to sniff packets of this port only as all traffic meant for it, is coming to this port.

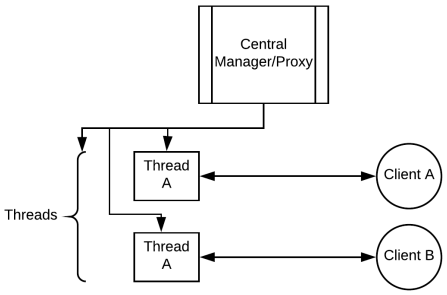


Figure 3.1: Basic Threading Architecture for Central Manager.

I already found prototype of SiegeBreaker working with single connection. We built an entity called Central Manger which does following jobs:

3.1.2 Challenges & Code Subtlety

Challenges: First, Enable serial connections from same client i.e Given an instance of proxy is running, client code can executed multiple time without resetting proxy. Second is to, enable parallel connections from same client i.e Given an instance of proxy is running, multiple client systems can be run simultaneously.

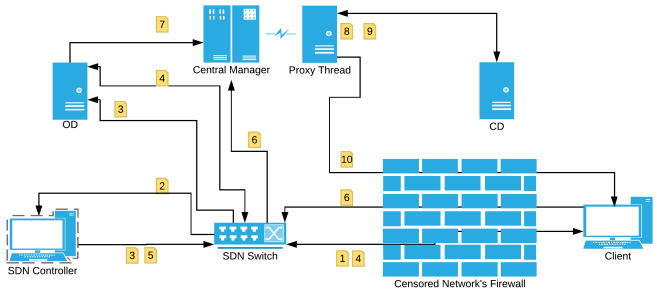
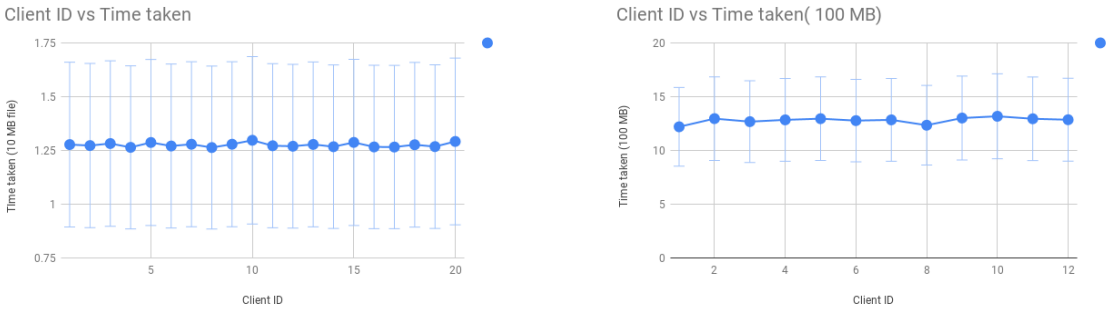


Figure 3.2: Subtle Change in SiegeBreker Protocol Manager.

3.1.3 Deter : Experimental Setup

I learned about DETER, it's rules system and how it gives out required topology. I generated following topology on DETER test-bed, in order to perform experiments related to Multi-Client aspect of SiegeBreaker.

Some Comments: Overhead incurred because of thread management and Central Manager should not increase in consecutive runs. This is showed by minimalistic difference in time taken to download files serially. On the other hand, larger sized files(say 100MB) show more variation in time taken, compared to smaller sized files(10 MB).



(a) 10 MB file

(b) 100 MB file

Figure 3.3: Deter Experiments

3.2 SEMESTER 2

3.2.1 Problem A : Raw Sockets

Since underlining code used 'raw' sockets, we experienced unoptimal performance from Siege-Breaker downloads compared to wget. This was happening because raw sockets serves all packets inbound to system. Thus we have to apply if-else filters to throw irrelevant packets out. These if else checks were performed for all inbound packet, including lot of irrelevant ones. We realized a typical SDN controller processes several packets per second, will experience more cross-traffic. We wanted a library which can give us filtered traffic i.e user space process only sees 'Siege-Breaker' packets.

3.2.2 Solution A : Libpcap

We used libpcap, a portable C/C++ library for network traffic capture. This increased NOT only packet processing speed of system, but portability too. Unlike raw sockets which only work on Linux based distros, libpcap works on windows too. This was good enough reason for us to move to libpcap. However, libpcap has many bad examples on internet, which simply don't work. After spending quite a lot of time, in figuring out how libpcap works. We moved proxy code to libpcap.

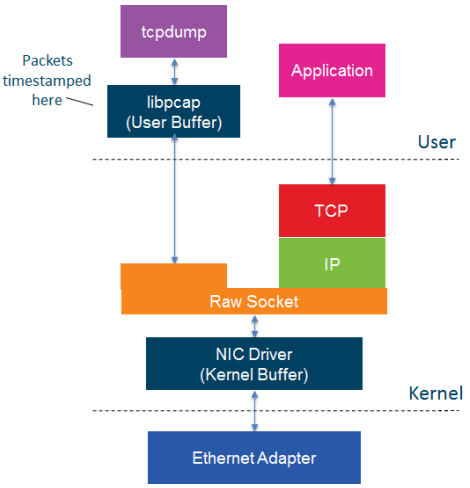


Figure 3.4: Comparision of Libpcap and Raw socket

3.2.3 Problem B : Simultaneous Downloads

After moving proxy code to libpcap, we saw significant improvements in performance.(and in our joy!). Soon, we felt the need to move even client code to libpcap, as clients systems too, have many internet-connected applications. They generated a lot of cross-traffic for SiegeBreaker client, who bore the burden of filtering them via if-else statements. Moreover, it was quite cpu-intensive, lot of cpu cycles were wasted ONLY in retreiving packets of interest.

3.2.4 Solution B : Again Libpcap

Porting client side code, turned out to be more challenging. Client side code had some issues with the way client processes incoming packets from Covert Destination. After spending few weeks, we were able to achieve desired feat. This enabled a single client to access multiple websites simultaneously, just like the case with typical TCP connections. We observed little to no improvement in performance. However, using pcap ensured that bandwidth will be shared

fairly by fellow TCP/SiegeBreaker connection. This "fairness", is essential as a SiegeBreaker connection, shouldn't hog all resources by itself.

3.2.5 Experiments Results

My work dictates performance(improvement) and transport layer fairness of system , and same should be tested and verified by experiments given below. Moreover, I also helped in conducting other irrelevant(to this project) experiments.

1. **Multi Client :** We did a sample run of 9 clients simultaneously downloading a 100 MB file. As evident from the results, the clients flows received an equal share of the link capacity of 11% (i.e. equal division among the nine client). Similar results were observed when this experiment was repeated for larger files (such as 200 MB).

2. **Fairness I:** Previous experiment's equal division depicts fair share policy enforced among SiegeBreaker clients.

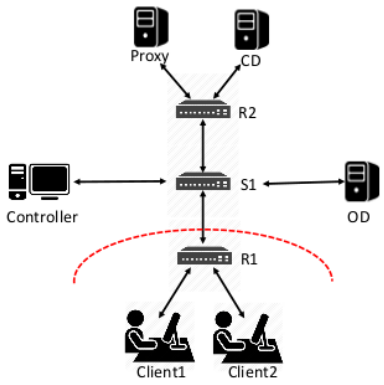


Figure 3.5: SiegeBreaker's experimental topology

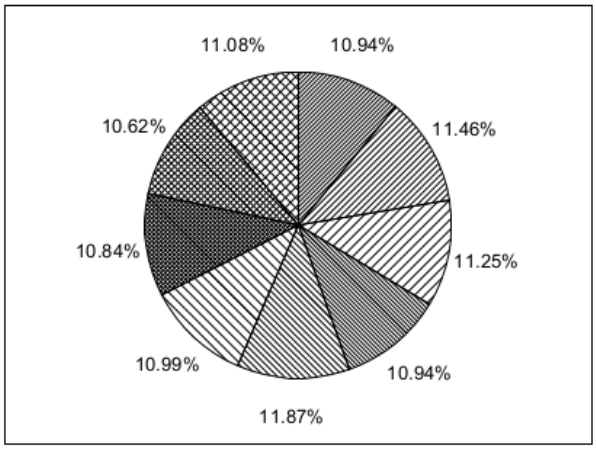
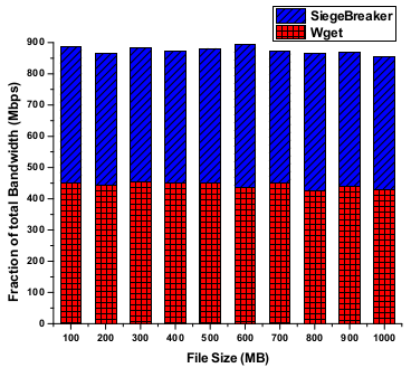


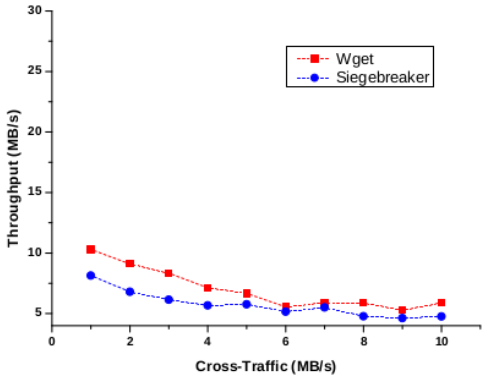
Figure 3.6: Percentage share of bandwidth when SP handles multiple SiegeBreaker clients.

3. **Congestion and Flow Control:** Both SiegeBreaker and wget clients simultaneously download large files (varying between 100 MB and 1GB) from different CDs, over 1 Gbps shared network link (between s1 and s2, see Fig. 3). The throughput achieved over this shared link, for both SiegeBreaker and wget.

4. **Fairness II:** To check fairness in presence of cross-traffic, we used one client (Client1 in the topology) to download a file of size 100 MB using Siege- Breaker, while simultaneously, on a shared 100 Mbps link (between R1 and S1), another client (Client2) downloads a file directly from CD. The second client provides a variable cross-traffic load ranging from 0 to 10 MB/s.



(a) TCP's throughput on SiegeBreaker vs wget

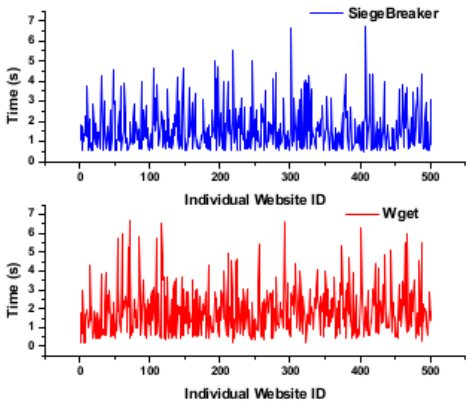


(b) SiegeBreaker vs wget in presence of varying cross-traffic

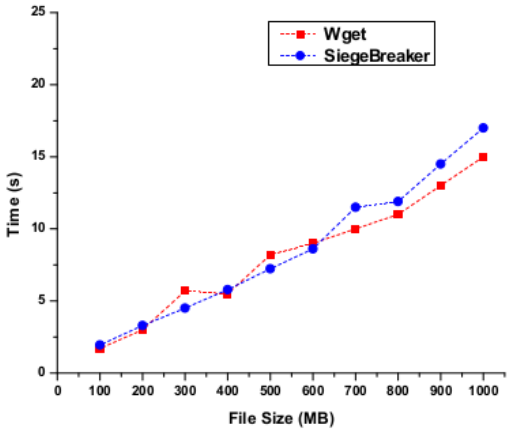
Figure 3.7: Two Real World Experiments depticting Fairneess and Flow Control respectively

5. **Web Browsing Performance:** We recorded the download times for the the home pages of Alex top-500 websites, which were blocked by our Universitys perimeter firewall. Refer to figure We ran 500 parallel SiegeBreaker and wget(without censorship) clients individually.

6. **Bulk Download Performance** : We instantiated our own web servers on six geographically distributed cloud hosting machines, each serving files of various sizes. We compere SiegeBreaker and wget's performance in downloading censored and uncensored files of size up to 1GB, when Overt Destination and Covert Destination are hosted on Internet.



(a) SiegeBreaker vs wget for browsing alexa top-500 websites

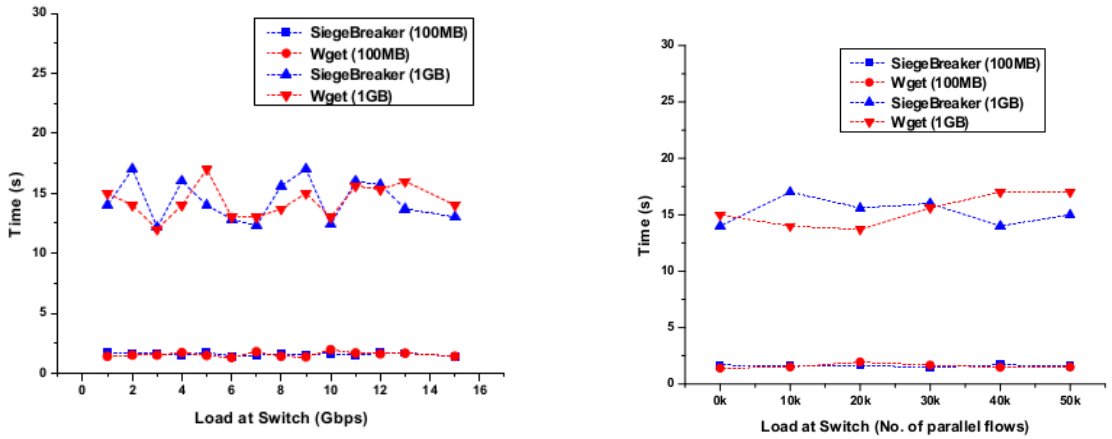


(b) SiegeBreaker vs wget for file download upto 1 GB

Figure 3.8: Two Real World Experiments depicting browsing times and bulk downloads respectively

7. **Heavy stress - Cross-traffic($\approx$ 15 Gbps) and Parallel flows($\approx$ 50 K)**: We wanted to gauge the performance of SiegeBreaker under heavy stress - cross-traffic loads and many parallel flows. We connected 15 hosts to the switch, who established P2P connections with one another and exchanged large volume of traffic (15 Gbps) to generate the cross-traffic.

For second part of experiment, the hosts, instead of the P2P data exchanges, ran Apache Benchmark Version 2.4 and connected to a web server and spawned many concurrent web connections (as many as 50k). We again downloaded large files (100 MB and 1 GB) from the cloud servers, for both parts of experiment.



(a) SiegeBreaker vs wget while increasing cross-traffic at SDN switch

(b) SiegeBreaker vs wget while increasing number of parallel flows

Figure 3.9: Experiments depicting effect of cross traffic and parallel flows respectively

3.2.6 Problem C : ICMP Signaling

The use of ICMP echo messages for signaling is a critical design decision for SiegeBreaker. As already described, it reduces the burden on the system to identify DR requests, while also preserving privacy of other flows. It may be argued that hypothetical adversaries (extremely powerful censorious regimes) may block all ICMP messages. Such scenarios may prevent the functioning of Siege- Breaker. Hence we have decided to relinquish our dependency at another protocol - ICMP and try some other signalling mechanism.

3.2.7 Solution C : Email for Signaling

Existing research has shown email to be successful way to bypass censorship in several forms - SWEET, MailMyWeb etc. We aim to use email as a covert channel for signaling, because:

1. If censor is incapable of observing recipient email address, then, it is unable to block email destined to a particular client, thereby inviting us to use it as covert and all-time-available channel.

2. If censor can indeed see recipient email and it's contents. we can hide our "signal" inside email attachments via steganography.

As of now, I am exploring several email providers in censorious regimes, how and whether they support SMTP/IMAP/POP3. Moreover, email-introduced latency will also be worked upon.

Chapter 4

Future Work

In our future work, we aim to engineer a better covert channel, that maintains stealthy signaling, ideally without the need to inspect all packets. Some possible options could be SIP protocol, VoIP, Video Streaming etc, each providing different guarantees due to the way they're designed.