Anonymity Network: Cloud-based Onion Routing

Fabian Grünbichler Klaus Krapfenbauer Thomas Hackl
Stefan Gamerith Milan Dzenovljanovic

ABSTRACT

An anonymity network should enable users to access the public Internet while at the same time hiding their real identity (e.g., IP address) from prying eyes. The main focus of this paper is a class of anonymity networks based on Onion Routing. Connections routed through such a network are strongly resistant to traffic analysis and are not limited to one specific protocol (e.g., HTTP(S)). In an Onion network, messages are encrypted and transmitted over a set of Onion Routers. Every Onion Router removes one layer of encryption (like the skins of an onion), reads the routing instructions contained within and sends the rest of the content to the next router. This process is repeated for all nodes along a so called chain, where every router is only aware of the existence of the previous and next node. Only the first node knows the true location (or identity) of the user, and only the last node knows the actual plain-text data. Today's most used Onion Routing network is Tor. Because of the ever-increasing number of users, the Tor network often suffers from poor performance. Therefore as well as for other benefits of cloud computing (broad network access, resource pooling, rapid elasticity, ensured services), moving Onion Routing services into the "cloud" seems like a logical step. This paper describes anonymous connections and their implementation using Cloud-based Onion Routing.

General Terms

Security, Privacy

Keywords

Anonymity networks, Onion Routing, Cloud computing

1. INTRODUCTION

Anonymity networks enable users to communicate over the public Internet while hiding their identity from one another. Nevertheless, is Internet communication private? One of the greatest concerns today is preventing outsiders from listening in on electronic conversations. But *traffic analysis* can also leak some sensitive information because encrypted messages can be tracked, revealing who is communicating with whom. [1]

One of today's most scrutinized as well as most popular tool for accessing the Internet anonymously is Tor. By routing the traffic through a sequence of nodes, Tor allows the traffic to appear to the target as if it were originating from the last node, instead of from the user directly. The main disadvantage of Tor is that its relays are hosted by volunteers all over the world. This volunteers mainly use private, consumer-grade ISP connections which often have limited bandwidth capacity, therefore Tor users often face performance issues. [2]

Moving Onion Routing services to the "cloud" could bring the advantage of being able to quickly rotate between a lot of different IP addresses as well as adding a large number of high-quality, high-bandwidth nodes to the network. The biggest advantage of most cloud infrastructures lays in their "elasticity". By spawning or terminating virtual machine instances in the cloud one can easily, quickly and automatically add or remove nodes/routers to or from an anonymity network. [2]

The main goal of *traffic analysis* is to discover participants of communication, whether the participants are exchanging mails or a user is just browsing the Web. Bearing this in mind, *anonymity networks* are developed to mislead the observers by making it difficult to derive such information, which is usually stored into the packet headers or in the encrypted payload, from the connection.

Thus, any information that carries identifying data must be sent through the *anonymous network*. Onion Routing is one implementation of such an *anonymous network* that stands out with comparably low latency and low overhead. Fundamentally, Tor's protocol is an extension of Onion Routing and provides two-way, real-time, connection-based communication that does not require the target to participate in the *anonymity* protocol. All aforementioned features together make anonymising communication on the Web easier. [1] [2]

2. ONION ROUTING

2.1 Overview

In Onion Routing, user connects to a target over a sequence of machines called *Onion Routers*, rather than connecting

directly to a target. This way the *originator* can remain anonymous to an *observer*. It is possible for the *originator* to remain anonymous to the target, if all informations that identifies the *originator* has been removed from the plain-text data stream before it is sent over the *Onion Routing network*.

2.2 Data transfer

In the *Onion Routing network* routers communicate using TCP connections (it could be implemented on top of other protocols as well). The set of the *Onion Routers* in a route is defined at the very setup of the connection, when the *originator* creates an *Onion*, and each router knows only the previous and next routers along the route. An *onion* is a layered data structure that encapsulates cryptographic information such as the keys or key exchange data as well as the cryptographic algorithms used during the data transfer. Along the route each *Onion Router* uses its public key do decrypt the entire *Onion*. This way the identity of the next router along the route as well as the embedded *Onion* is exposed. The *Onion Router* sends the *Onion* to the next router, after the connection is established, data can be sent in both directions. Using the keys and algorithms from the Onion, the outgoing data is encrypted for all routers before it is sent by the originator. During the data transfer along the route, each *Onion Router* removes one layer of encryption, like peeling an onion, so at the target the data arrives as plain text. Because of the layered public key cryptography the *onion* looks different to every *Onion Router* along the connection. [3]

2.3 Overhead

Because of the inherent properties of public key decryption, the most expensive calculation in the Onion Routing network is public-key cryptography, hence it is only used during the initial setup of the connection. In the data transfer phase only the symmetric cryptography is used, which is much faster and can be as fast as ordinary link encryption. Delay in the data transfer depends on the number of the *Onion Routers* which are involved. Overall, overhead in the *Onion Routing* is sufficiently small. [3]

2.4 Formal Description

Let say we have N number of routers, where each router has its own public (Su) and private (Sr) key. The public key is known to the *originator* while the private keys are only known to the router itself. [4]

As outlined before, *Onion Routing protocol* relies on public-key cryptography, therefore functions for encryption *E[key](data)* and decryption *D[key](data)* are needed. A key pair is generated in such a way that data encrypted with the public key Su can be decrypted only with the private key Sr, and vice versa, i.e. *D[Su](E[Sr](data)) = data* and *D[Sr](E[Su](data)) = data*. [4]

When the *originator* wants to send informations over the *Onion Routing network*, it sends a request to the *Onion Proxy* and gets back the set of randomly chosen *Onion Routers*, e.g. <4, 3, 5>. This means that information, which travels from the *originator* to the *target*, will be routed over this three routers respectively. The first router in the connection is called *entry node* and the last node is called *exit node*. Before information is sent to the *entry node*, the *Onion Proxy* creates an *Onion*, which in our case looks like this: *E[4u](3's IP address, E[3u](5' s IP address, E[5u](data)))*. The *Onion* is then forwarded to the *entry node* (4). The *entry node* decrypts the *Onion* with its private key, reads the IP address of the next node (3) and passes the data to it where the same process is repeated. [4]

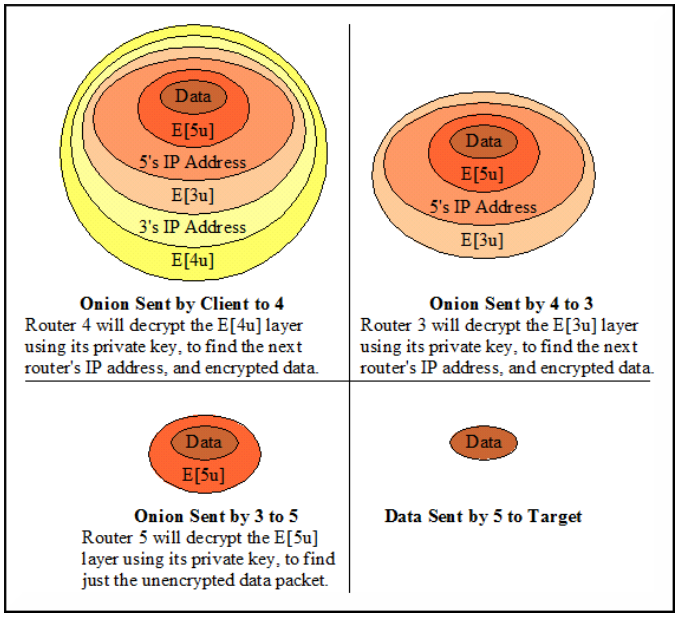

Figure 1: Onion Routing [4]

It is possible that each router has a number of TCP connection to the other routers, and multiple circuits[1] between two nodes might use the same TCP connection. Therefore, every router needs to remember the association between pairs of circuits that belong to the same Onion route. [4]

When the route <4, 3, 5> is established it allows data to travel in both ways without including routing information. In the example above, when a response message from the *target* arrives at the *exit node* (5), (5) already knows to which circuit and node (3) it should forward the information. (5) first encrypts the data it received from the *target* with its private key and sends it to (3). (3) encrypts the data with its private key and sends it to (4). (4) does the same thing and passes the data to the *onion proxy*. In order to read the original information, the *onion proxy* decrypts the data it received from (4) as: *D[4u](D[3u](D[5u](encrypted onion)))*. [4]

2.5 Advanced considerations

Thera are many ways to attack *Onion Routing* systems today: Denial of Service (DoS) attacks, Distributed Denial of Service (DDoS) attacks, passive tracking analysis, active tracking analysis.[4]

Making *Onion Routing* secure is a research area that

[1]A circuit represents the logical connection between two nodes

grows from day to day with plenty of different ideas. [4]

2.5.1 Denial of Service (Dos) and Distributed DoS

As stated in the "World Wide Web Security FAQ": *Denial of Service (DoS) is an attack designed to render a computer or network incapable of providing normal services. The most common DoS attacks will target the computer's network bandwidth or connectivity. Bandwidth attacks flood the network with such a high volume of traffic, that all available network resources are consumed and legitimate user requests can not get through. Connectivity attacks flood a computer with such a high volume of connection requests, that all available operating system resources are consumed, and the computer can no longer process legitimate user requests.* [5]

A Distributed Denial of Service (DDoS) attack uses many computers to launch a coordinated DoS attack against one or more targets. [5]

2.5.2 Traffic Analysis

The best way to protect from traffic analysis is to pay attention to the size of the onion (all onions should be the same size, regardless of the position along the route) and make sure that timing information on circuits is well hidden. [4]

3. TOR

3.1 Overview

Tor is a second generation and the most sophisticated implementation of the *Onion Routing* protocol. The first important upgrade that Tor brings is **perfect forward secrecy**: Tor now uses the *"session" key* design, where the *originator* negotiates with each node to derive a shared secret — the *session key*. The *session key* exchange protocol which Tor uses is called Diffie-Hellman[2] key exchange. The generated keys are only used for one session, i.e. for the communication over one specific route. Public-key encryption is used only to establish the connection and secure the key exchange. [4] [6]

Congestion control: Tor has the ability to detect congestion and send less data until the congestion goes down, while still maintaining anonymity. [6]

Directory servers: A small set of very reliable servers act as *directory servers*: they are able to control and monitor the chain nodes. Every active chain node has to register itself at the *directory servers*, which then spread the information to all interested clients. [6]

Leaky-pipe circuit topology: *Onion Routing* allows only the last node in the chain to be the *exit node*. In Tor's implementation of the *Onion Routing* protocol, this is changed allowing every node in the chain to be the *exit node*. [4]

End-to-end integrity checking: Tor verifies data integrity at every link of the chain in order to guarantee that no node in the chain can change the data sent from the originator. [6]

[2]W. Diffie and M.E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (1976), 644-654.

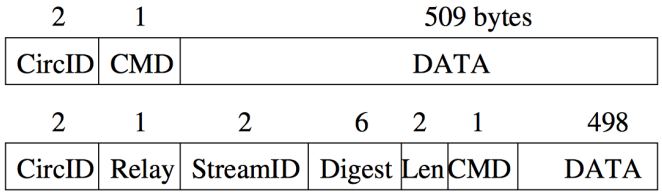

Figure 2: Cell [6]

3.2 Design

Onion Routers connect over a TLS-encrypted connection to other *Onion Routers* in the network. Each user who uses Tor runs an *onion proxy* to maintain the connections over the network, to communicate with the *directory server* and to maintain the connections with the user's applications.[6]

Each *Onion Router* has an *onion key* which is used to setup the circuit and an *identity key* which is used to register at the *directory server* and to sign TLS certificates. [6]

The main units of communication in Tor are fixed-size data structures called *cells*.[6]

3.3 Cells

In Tor's design every *cell* consists of a header and a payload and is exactly 512 bytes long. The header of the *cell* contains a circuit identification number (CircID), which is used to identify the individual logical connection, and an instruction which tells the next node how to handle the payload. We can distinguish two types of *cells*, *control cells* and *relay cells*.[6]

Control cells consist of one of the following commands: *padding*, *create* and *created* (create a new circuit), *destroy* (destroy a circuit). [6]

Relay cells are special as they tell the receiving node to forward the (encrypted) payload to the next node or a target server. The content of the *relay cells* is encrypted using AES. The commands of the *relay cells* are: *relay begin* (open a stream), *relay data*, *relay end* (close a stream), *relay connected* (successful relay begin), *relay teardown* (close a broken stream), *relay extend* (extend the circuit), *relay extended* (acknowledge of the *relay extend*), *relay truncate* (terminate only part of the *circuit*), *relay truncated* (acknowledge of the *relay truncate*), *relay sendme* (congestion control), *relay drop*. [6]

3.4 Construction of a circuit

To create a *circuit*, the originator's *onion proxy* sends a *create cell* (with a new CircID) to the first node, with a payload that contains the first half of a Diffie-Hellman handshake. The first node responds with the *created cell* containing the second half of this Diffie-Hellman handshake as well as the hash of the generated shared secret. The onion proxy can generate an identical shared secret using both halves of the Diffie-Hellman handshake. At this point the *circuit* between these two nodes is created and they can send *relay cells* to each other. [6]

To extend the circuit to the next node, the originator's *onion proxy* sends a *relay extend cell* to the first node. The

relay extend cell's encrypted payload contains the address of the second node and the first half of a Diffie-Hellman handshake. The first node decrypts the payload and wraps the handshake data in a new *create cell* (with new CircID) which it passes to the second node. The second node responds to the first node with a *created cell*, along with its half of Diffie-Hellman and a hash of the shared secret. The first node encrypts and wraps this data in a *relay extended cell* and passes it back to the originator. Now the *circuit* is extended. [6]

To extend the *circuit* further, the above mentioned protocol is repeated by wrapping further *relay* layers around the *relay extend cell*.

To completely tear down a *circuit*, the originator sends a *destroy cell* over the *circuit*. Every node which receives such a cell, closes the receiving *circuit* and passes the *destroy cell* forward. Because of the incremental nature of *circuit* building, it is possible to tear down a circuit incrementally. The originator can send a *relay truncate cell* to one node in the circuit. That node passes the *cell* forward and replies with the *relay truncated cell*. This way the originator is able to extend the circuit to different node. [6]

3.5 Opening and closing stream

When the originator wants to open a new connection to the target, he sends a request to the *onion proxy* to open a stream. By sending the *relay begin cell* to the *exit node*, the *onion proxy* opens the stream. When the *exit node* is connected to the target it sends a *relay connected cell*. Now the *onion proxy* can send data over the circuit in a (sequence of) *relay data cell(s)*. [6]

We differentiate two types of cells for closing a stream. When the stream is closed normally, one node sends a *relay end cell*, and the other side responds with its *relay end cell*. For a stream that closes abnormally (e.g., because of a connection timeout), the node next to the failed node sends a *relay teardown cell*. [6]

4. PERFORMANCE EVALUATION: TOR AND CLOUD-BASED ONION ROUTING

This evaluation considers two experiments for comparison: [2]

- downloading single files (TorPref)
- downloading entire web sites

Individual Files Download:

For performance evaluation when downloading single files, the Tor-supplied TorPref[3] measurement tool was used. The experiments used five *cloud-based Onion Routing circuits* which were built over the US and EU datacenters, and five Tor circuits, randomly built by Tor. Three files of size 50KB, 1MB and 50MB were downloaded, each download was repeated 100 times. The results are shown in the Figure 3 (left), and as you can see, the average *cloud-based Onion Routing's* performance is better then Tor's.[2]

Downloading Web pages:

For this purpose 10 **entire** home pages were downloaded using Tor, *cloud-based Onion Routing* and a direct Internet connection, and each home page was downloaded 10 times in a row. For the *cloud-based Onion Routing* downloads, two use cases were tested, one with the *cloud-based Onion Routing relay* under load (serving 50 connections at the time) and serving just one connection at the time. In the Figure 3 (center) you can see that the *cloud-based Onion Routing* is several times faster then Tor, although it is slower then direct access.[2]

Concurrent Users:

Here the number of concurrent users that one node can maintain were evaluated. Also, for the purposes of this experiment, nodes with different performances characteristics (nodes differ in CPU, memory and I/O resources) from Amazon EC2 were evaluated. It was shown that m1.large and c1.medium instances can maintain more then 100 concurrent users, while the t.micro instance has problems with handling 10 users. The results are shown in Figure 3 (right) where you can see that the two largest instances achieved speed between 50 Mbps and 120 Mbps.[2]

5. CONCLUSION

The goal of *Onion Routing* is to protect anonymity by hiding the location and identity of its users. To fulfill its purpose *Onion Routing* uses encryption to encrypt the original data and to create an onion. During its way towards the target the onion will be decrypted at each hop. When a router peels one encrypted layer of it only sees the address of the next router in the chain. Since no one can see the full path, we can say that the communication is anonymous. [4]

Because of elastic cloud infrastructure and powerful connectivity, *cloud-based Onion Routing* introduces a new, more scalable level of implementation of *Onion Routing* protocol. Also, *cloud-based Onion Routing* brings additional level of protection against blocking, leaving censors with two options: allow the traffic or block all IP prefixes used by one cloud provider. Blocking all prefixes will cause a big damage: for example, Amazon EC2, hosted over a million instances with common IP prefixes in 2010.[4] Preliminary surveys demonstrates that *cloud-based Onion Routing* can be efficient, high performing and cost effective. [2].

Onion Routing is an applied solution for real anonymity problems but yet still a big research area. [7]

[3]https://metrics.torproject.org/tools.html

[4]Recounting EC2 One Year Later: http://www.jackofallclouds.com/2010/12/recounting-ec2/

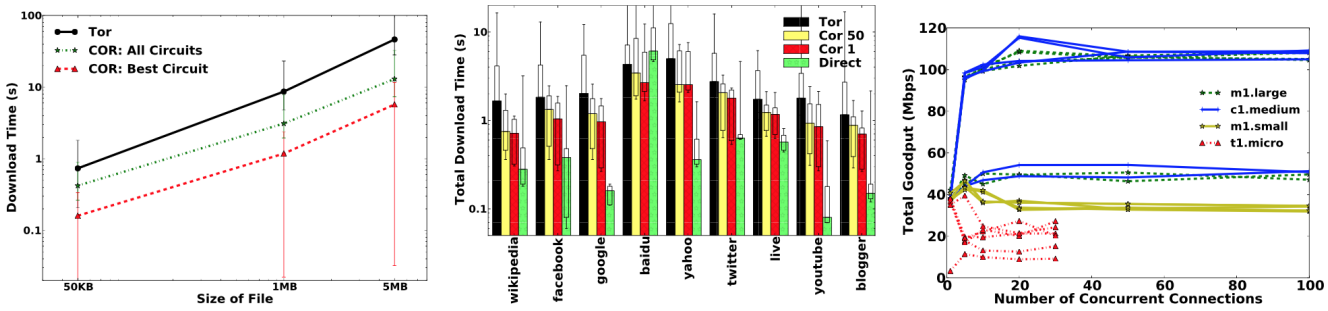

Figure 3: Evaluation [2]

6. REFERENCES

[1] Michael D. Reed, Paul F. Syverson, David M. Goldschlag, Naval Research Laboratory, *Anonymous Connections and Onion Routing*, 1998.

[2] Nicholas Jones, Matvey Arye, Jacopo Cesaero, Michael J. Freedman, *Hiding Amongst the Clouds: A Proposal for Cloud-based Onion Routing*, Princeton University, 1998.

[3] Michael D. Reed, Paul F. Syverson, David M. Goldschlag, *Onion Routing for Anonymus and Private Internet Connection*, 1998.

[4] Marc O'Morian, Vladislav Titov, Wendy Verbuggen, *Onion Routing for Anonymus Communication*, http://ntrg.cs.tcd.ie/undergrad/4ba2.05/group10/

[5] Lincoln Stein, John Stewart, *The World Wide Web Security FAQ: Securing against Denial of Service attacks*, http://www.w3.org/Security/Faq/wwwsf6.html

[6] Roger Dingledine, Nick Mathewson, Paul Syverson, *Tor: The Second-Generation Onion Router*.

[7] Paul F. Syverson, Naval Research Laboratory, *A Peel of Onion*.