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 Web while, at the same time, hiding their identity on the Internet. Onion routing's anonymous connections are strongly resistant to traffic analysis and can be used through a socket connection. In an onion network, messages are encrypted and transmitted over a onion routers. Every onion router removes the layer of encryption, reads the routing instructions and sends the message to the next router in the chain where the same process is repeated. In the onion network chain every router is aware just about existence of the previous end next node, and only the last node knows the content of a data. Todays most used global onion routing network is Tor. Because of using a large number of volunteers, Tor faces a poor performance often. Therefore as well as for 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 Web while hiding their identity from one another. Nevertheless, is Internet communication private? Greatest concerns today is preventing outsiders listening in on electronic conversation. But *traffic analysis* can also leak some sensitive information because encrypted messages can be tracked, revealing who is talking to whom. [1]

Todays best and most popular tool for accessing the Internet anonymously is Tor. By taking the traffic through a sequence of nodes, Tor allows the traffic to appear to target from its last node, not from the user directly. Main disadvantage of Tor is that its relays are hosted by a volunteers all over the world. This volunteers mainly use private, consumer-grade ISP connections which often have limited bandwidth capacity, therefore Tor faces performance drops. [2]

Moving onion routing services to the "cloud" will benefit with possibility to rotate between numbers of IP addresses as well as with large number of high-quality, high-bandwidth nodes. Great advantage of the "cloud" infrastructure lays in its "elasticity". By spawning new virtual machine instance in the "cloud" one can easily add or remove a new node to a "cloud". [2]

Main goal of *traffic analysis* is to discover participants of conversation, weather the participants are exchanging mails or user is just browsing the Web. Bearing this in mind, the *anonymity networks* are developed to obfuscate the observers by making them difficult to identify informations, that are stored into the packet headers or in a track of encrypted payload, from the connection. Thus, any information that carries identifying data must be sent through the *anonymus network*. Onion routing is implementation of *anonymous network* that stands out with low latency and overhead. Fundamentally, onion routing relays on Tor's protocol and provides two-way, real-time, connection-based communication that does not require the target to participate in the *anonymity* protocol. All afore mentioned 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 connect directly to a target. This way the *originator* and the target remain anonymous to *observer*. It is possible for the *originator* to remain anonymous to the target. In this case all informations that identifies the *originator* have to be removed from the data stream before being sent over the *onion routing network*.

2.2 Data transfer

In the *onion routing network* routers communicates over a TCP (it can be implemented on top of other protocols). The set of the *onion routers* in a route is defined at very setup of the connection, when *originator* creates an *onion*, and each router knows only previous and next router along the route. An *onion* is a layered data structure that encapsulates cryptographic informations such as the keys and cryptographic algorithms used during the data transfer. Along the route each *onion router* uses its public key do decript the entire *onion*. This way the identity of the next router along the route as well as embedded *onion* is exposed. The *onion router* ends the data to the next router and after the connection is established, the data can be sent in both directions. Using the keys and algorithms from the onion, the data is encrypted before it is sent from 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 a plain text. Because of the layered public key cryptography an *onion* looks different to every *onion router* along the connection. [3]

2.3 Overhead

Because of the public key decryption, the most expensive calculation in the onion routing network is public-key cryptography, and it only takes place at the connection setup. In the data movement phase only the symmetric-key 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*. Overall, overhead in the *onion routing* is sufficiently small. [3]

2.4 Formal Description

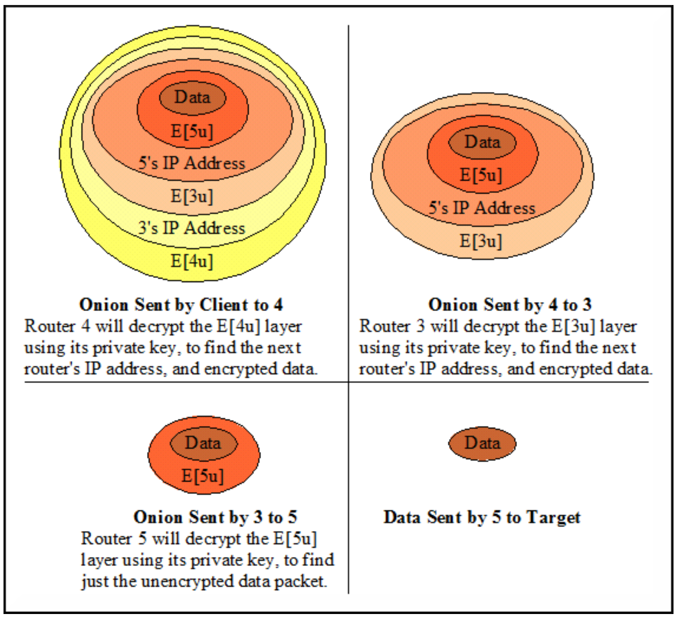
Let say we have N number of routers, where each router have its own public (Su) and private (Sr) key. Public key is known to the *originator* and private keys are known to the router. [4]

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

When *originator* wants to send informations over the *onion routing network*, he 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 circuit[1] is called *entry node* and the last node is called *exit node*. Before information is sent to the *entry node*, *onion proxy* creates an *onion*, which in our case looks like: *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 the data to it where the same process is repeated. [4]

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

[1]Circuit represents the logical connection between two nodes



Data
E[5u]
5's IP Address
E[3u]
3's IP Address
E[4u]
Onion Sent by Client to 4
Router 4 will decrypt the E[4u] layer using its private key, to find the next router's IP address, and encrypted data.
Data
E[5u]
5's IP Address
E[3u]
Onion Sent by 4 to 3
Router 3 will decrypt the E[3u] layer using its private key, to find the next router's IP address, and encrypted data.
Data
E[5u]
Onion Sent by 3 to 5
Router 5 will decrypt the E[5u] layer using its private key, to find just the unencrypted data packet.
Data
Data Sent by 5 to Target

Figure 1: Onion routing [4]

When the circuit <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 know to which circuit and node (3) it should forward the information. (5) first encrypt 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 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 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]

The best way to protect from the DoS attacks is to use digital currency like Hashcash[2].

2.5.2 Traffic Analysis

The best way to protect from traffic analysis is to pay attention on the the size of the onion (all onions should be the same size) and make sure that information about timing 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*. First important upgrade that Tor brings is **perfect forward secrecy**: Tor now uses the *"session" key* design, where the *originator* negotiates with every node upon a *"session" key* (shared secret). *"Session" key* exchange protocol which Tor uses is called Diffie-Hellman key exchange [W. Diffie and M.E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (1976), 644-654.] and keys that are generated during this exchange are valid only during the communication (depend on the circuit's lifetime). Public-key encryption is used only to establish connection. [4] [6]

Congestion control: Tor have an ability to detect congestion and sends less data until congestion goes down while maintaining anonymity. [6]

Directory servers: Most reliable nodes act as *directory servers*: they are able to control and monitor the chain nodes. Only *directory servers* know the public key and IP address of the chain nodes, and every activ chain node has to register itself to the *directory servers.* [6]

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

End-to-end integrity checking: Tor verifies data integrity until it leaves the chain so that not a single node in the chain can change the contents of data cells. [6]

3.2 Design

Tor's every *onion router* connects over a TLS to every other *onion router* in the network. Each user ted to use Tor runs *onion proxy* to maintain a circuits over the network, to communicate with *directory server* and to maintain tho connections with user's applications. [6]

Each *onion router* have a *onion key* which is used to setup the circuit and *identity key* which is used to register to the *directory server* and to sign TLS certificates. [6]

Main unit of communication in Tor are fixed-size *cells.* [6]

[2]http://www.hashcash.org

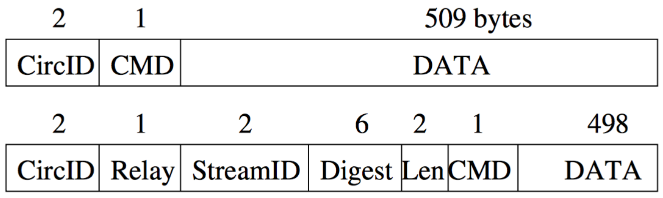


Figure 2: Cell [6]

3.3 Cells

In Tor's design every *cell* is consists of header and payload and is fixed-size of 512 bytes. In the header of the *cell* is a circuit identification number (CircID), which is used to identify the individual logical connection and instruction which tells what to do with the payload. We can distinguish two types of *cells*, *control cells* and *relay cells.* [6]

Control cells consist of a 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 target. Content of the *relay cells* is encrypted using AES. The commands of the *relay cells* are: *relay begin* (open a stream), *relay data*, *relay nd* (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*, originator's *onion proxy* sends a *create cell* (with new CircID), which payload contains the first half of the Diffie-Hellman handshake, to the first node. First node responds with the *created cell* containing its half of Diffie-Hellman and hash of their shared secret. At this point *circuit* between this two nodes is created and they can send *relay cells* to each other. [6]

To extend the circuit to the next node, originator's *onion proxy* sends, along with the address of the second node and first half of the Diffie-Hellman handshake, to the first node a *relay extend cell.* First node wraps the data in a *create cell* (with new CircID) and passes it to the second node. Second node responds to the first node with the *created cell*, along with its half of Diffie-Hellman and hash to their shared secret, which wraps the data in a *relay extended cell* and passes to the originator. Now the *circuit* is extended. [6]

To extend the *circuit* further, above mentioned protocol is repeated.

To tear down a *circuit*, originator sends a *destroy cell* which every node receives, closes that *circuit* and passes the *destroy cell* forward. Because of the incrementally nature of the *circuit* building, originator can send a *relay truncate cell* to one node in the circuit. That node passes the *cell* forward and replays with the *relay truncated cell.* This way

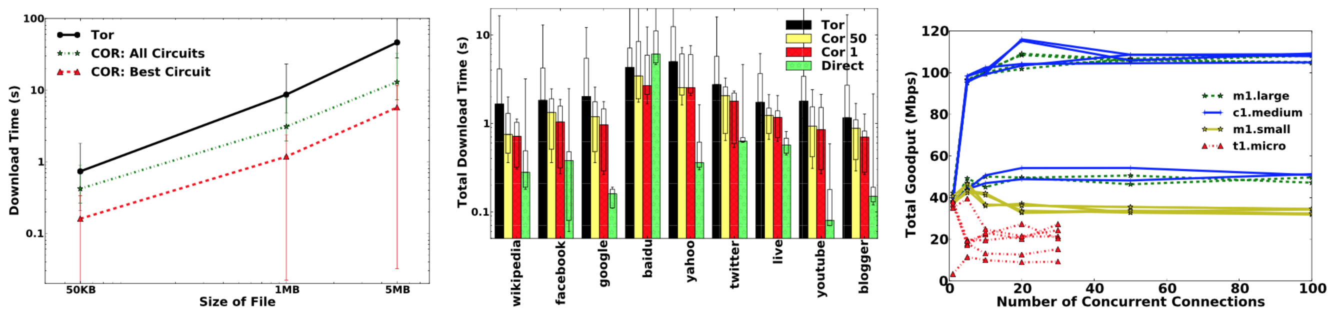


Figure 3: Evaluation [2]

originator is able to extend the circuit to different node. [6]

3.5 Opening and closing stream

When 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*, *onion proxy* opens the stream. When the *exit node* is connected to the target it sends the *rally connected cell.* Now the *onion proxy* can send data over the circuit in a *relay data cell.* [6]

We differentiate two types of cells for closing stream. When the stream is closed normally, one node sends a the *relay end cell*, and the other side responds with its *relay end cell.* For the stream that closes abnormally, 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, Tor-supplied TorPref[3] measurement tool is used. For experiments are 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 repeating each download 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 purposes 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 is evaluated the number of concurrent users that one node can maintain. Also, for the purposes of this experiment, nodes of the different size (nodes differ in CPU, memory and I/O resources) from Amazon EC2 are evaluated. You can see that m1.large and c1.medium can maintain more then 100 concurrent users, while the t.micro has problem 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

TODO

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

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.*