The TOR Data Communication System

Ramzi A. Haraty and Bassam Zantout

Abstract: **Since the day the Internet became a common and reliable mechanism for communication and data transfer, security officers and enthusiasts rallied to enforce security standards on data transported over the globe. Whenever a user tries communicating with another recipient on the Internet, vital information is sent over different networks until the information is dropped, intercepted, or normally reaches the recipient. Critical information traversing networks is usually encrypted. In order to conceal the sender's identity, different implementations have proven successful - one of which is the invention of anonymous networks. This paper thoroughly investigates one of the most common and existing techniques used during data communication for avoiding traffic analysis as well as assuring data integrity - the onion router (TOR). The paper also scrupulously presents the benefits and drawbacks of TOR**

Index Terms: **Anonymous system, data communication, security and integrity, the onion router (TOR).**

I. INTRODUCTION

Onion routing was originally prototyped by Sun Solaris 2.5.1/2.6 with implementations for web browsing, remote login, and sanitizing user information while transmitting information through data streams. The idea and further implementation of onion routing was based on the work of David Chaum (Chaum mixes) and further continued and enhanced by Michael G. Reed, Pal F. Syverson, and David M. Goldschlag from the US Naval Research Laboratory [1].

In 1995, the US Navy Office of Naval Research (ONR) sponsored the aforementioned authors to work on an anonymous communication mechanism that allows computer users to send and receive information over the Internet while remaining anonymous, as well as, preventing against traffic analysis and eavesdropping. In 1997, the project was funded by the United States Department of Defense Advanced Research Projects Agency (DARPA) under the high confidence network program, and more work was put on the original design and components of the algorithm and implementation. In 1998, a prototype of the project was running with an average 50,000 hits per day with a peak of 84,022 simultaneous connections on the system. DARPA and other sponsors of this project were also interested in applying the same onion routing methodology not only Internet appliances, but also on cell phones and other communication devices not necessarily using the Internet in order to achieve anonymity [2].

Little work and improvements were added to onion during the past decade due to lack of funding and interest. To this date the onion router (TOR) and the onion routing project are funded by ONR and DARPA whereby it is still under development with probably one of the largest testing labs in the world, the Internet. TOR operates with almost 900 dedicated onion routers worldwide, generating and processing 960 Mb/sec of bidirectional data streams [3], [4].

This paper investigates the implementation of TOR, which is widely used today and has made a major impact on the world of networking and particularly peer-to-peer communication. The remainder of the paper is organized as follows: Section II presents background material. Section III concentrates on TOR, outlining its features, advantages as well as its drawbacks. Section IV provides a conclusion.

II. BACKGROUND

Prior to onion routing, a previous implementation based on a simple model by David Chaum of the University of California, Berkeley [5] was introduced to solve this problem of source and destination identification through traffic analysis avoidance. Chaum mixes is a simple process where the identity of the sender is hidden from the receiving entity. All traffic sent back and forth from sender to receiver goes through a proxy that is able to sanitize sender and/or receiver information if need be; however, since the sender is the focus of the problem then the receiver's identity is kept as is. The proxy in this case is the only entity that can keep track of sender and receiver identities. Chaum mixes use a series of private and public keys whereby the sender trusts a single entity with its keys to encrypt and decrypt messages and data before sending information to the receiver. The trusted entity then relays the sanitized information that can be either encrypted (or not) to the receiving party.

Once the receiver (Beta in this case) answers Alpha's request and is ready to send back information, it does not know who and what Alpha is and only sends back information to the visible entity that sent the request that exited in this case from Cathy, who in return relays what Beta sent to Alpha. Chaum mixes started as a good idea with a single trusted entity to conceal the identity of the sender or the sender and receiver if need be. However while the aim of this model is to avoid traffic analysis occurring after traffic is generated by Alpha, other types of attacks such as timing attacks can be performed to determine that Alpha is indeed talking to Beta. This, although may not compromise the integrity of the data, does not prevent against traffic analysis. Due to timing and other types of attacks, different chains of Chaum mixes were added to the network creating "Chained Chaum Mixes". Chaum Mixes was a bright idea for hiding and "*anonymizing*" the identity of the sender and receiver; however, Chaum mixes where still susceptible to end-to-end attacks on trusted entities with time based attacks to determine the sender and receiver. Add to that the overhead of using public and private

Manuscript received April 12, 2014.
This work was funded by the Lebanese American University.
Authors are with the Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon 1102 2801, email: rharaty@lau.edu.lb, bassam.zantout@lau.edu.
Digital object identifier 10.1109/JCN.2014.000071

key encryption and decryption which had computation overhead back in the mid-1980s. Although Chaum mixes was lightly implemented and tested, a new algorithm and methodology inspired by David Chaum's algorithm saw the light in 1995 called onion routing.

III. TOR

A. Onion Routing to TOR

Onion routing promised not only to protect the integrity and confidentiality of data but also against eavesdropping and traffic analysis over the network and the Internet. Goldschlag, Reed, and Syverson identified [1], as David Chaum did, that there are two entities to protect, the data and the identity of that data. They have also investigated and considered that the possibility for malicious attackers being able to eavesdrop at any part in the physical network is eminent and therefore trusted entities may no longer be trusted (the mailman cannot be trusted with the recipient and sender's addresses). As a result, the authors of the onion routing project devised a way to limit the knowledge of this information as much as possible while achieving high levels of anonymity. Onion routing protects against traffic analysis attacks mainly because the sender does not talk directly to the recipient (similar to Chaum Mixes). Instead, it initiates a connection with an application-specific router called the "onion routing proxy" that will be able to handle the TCP and Socks request of that client. Before describing the details of TOR, it is important to mention that many implementations at the time were able to achieve anonymity of the sender and receiver with some drawbacks or at a certain cost for which these implementations could, to a certain, extent prevent against traffic analysis. Anonymizer [6], JAP [7], Miximinion [8], Tarzan [9], and Morphmix [10] are examples of such solutions offered at the time TOR was being developed. However, TOR has one major advantage over the other implementations, the number of clients using TOR, which provided the project priceless information and test results since all testing was done on the Internet.

TOR is the descendant of the onion routing project whereby the project has inherited many of the design concepts introduced by onion routing. TOR is a collection of onion routers, which have different functions and roles in a network and during network communication. Each router sends information in a secure way to the next hop in a TOR network whereby if any single router in the set of onion routers is compromised, then this breach will not affect the anonymity as well the data communication sent to and from the sender and receiver. Fig. 1 sheds light on what goes on behind the scenes. The software client download contains a TOR directory fetcher, which is able to talk to the TOR root servers and acquire the latest onion routers available worldwide. These root servers are monitored and managed by official TOR personnel. TOR nodes can be managed by TOR *enthusiasts* for whom anyone can become a server acting a TOR onion router.

B. TOR: Second Generation Onion Routing

Just like Chaum mixes, TOR aims at hiding the communication between the initiator and the target host for which the initiator needs to communicate with, and just like Chaum mixes TOR utilizes a series of proxies and makes communication travel through a number of hops before it connects the initiator with the target. Given the aforementioned, one may realize that the more the number of nodes, the more secure a connection becomes since tracking communication will be difficult from sender to receiver. Moreover, the more the number of nodes the more latency is added to the connection; and for low latency connections such as Secure Shell, Telnet, and other interactive applications using a high latency connection becomes impossible to work with. Hence, there is a trade off between a secure connection that enables anonymity and that is able to use a certain number of hops while keeping connection latency bearable. After plenty of testing and research TOR was designed to route connections through three intermediate TOR nodes and a last exit node before leaving the TOR network and delivering the communication to the receiver. A total of four nodes are involved in any TOR communication. While a client is connected to the TOR network using a specially developed TOR application, data is sent through the TOR network in an encrypted format with fixed size packets called "*cells*". Cells can fit 498 bytes and are only exchanged between the TOR nodes and the client using the TOR application. The recipient is not aware nor does the recipient participate in the TOR network. The cells in a TOR network have a fixed size so that snoopers are not able to detect the type of communication being transmitted from the sender, as well as, the response returned back from the TOR nodes. Therefore, having constant packet size camouflages the type of data being exchanged. TOR cells could either contain data or TOR instructions for initiating new circuits or giving commands to TOR network components for connections and disconnections as well as exchanging other information.

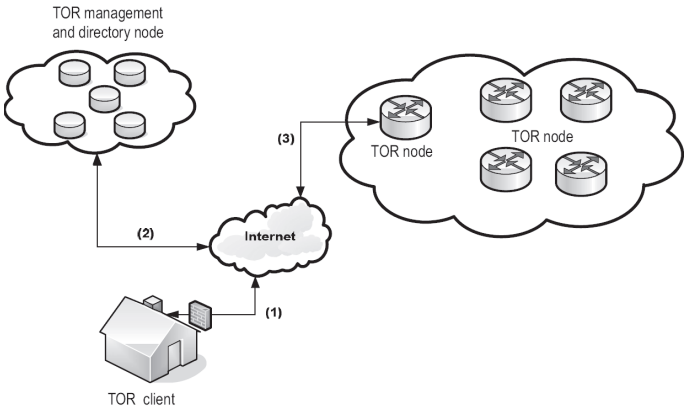

Fig. 1. A snapshot of TOR in action where multiple routes are being selected.

Upon initializing the TOR application, it starts to look for the first *bridge* (or first TOR node) that will link the user's computer to the TOR anonymous network; hence, the name *bridge*. A *bridge* is just another TOR node that accepts connections that are listed and maintained by five TOR management nodes and are secured by the TOR team. The TOR application contacts one of the five management nodes it requests a *bridge* for which a specific *handshake* occurs to get connected to the TOR network. Once connected successfully to the first bridge, the TOR software talks to the five directory services. The concept behind

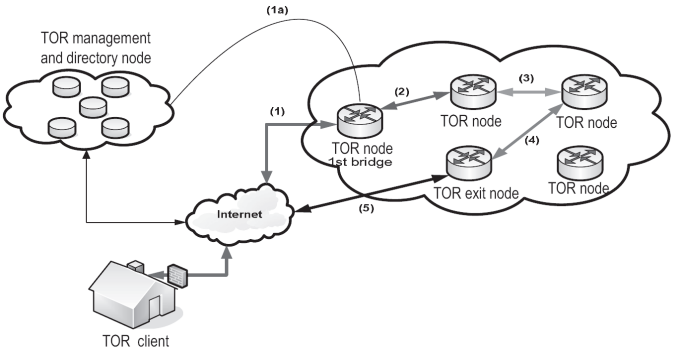

Fig. 2. A user connecting through the Internet to a TOR network.

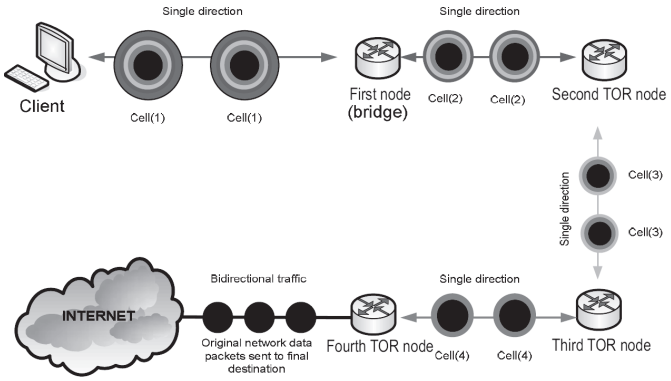

Fig. 3. A TOR Client connecting to a TOR Bridge.

having TOR nodes is to allow each node to relay *cells* from and to other TOR nodes, senders, and recipients without revealing the *cell's* content or the complete route to any of the nodes. This is achieved through cell encapsulation and multi-level encryption whereby each *cell* is encrypted/decrypted at every node and each node can only reveal a single encrypted layer in a *cell*. To better explain this, consider the following example that illustrates complete communication between a client's machine first establishing contact with a TOR *bridge* and then communicating via TOR nodes/circuit for downloading a file.

In Fig. 2, step (1) illustrates that a user must be obviously connected to the Internet in order to establish communication with the TOR network. In (2) the user's TOR software downloads a list of available *bridges* that are available to start forming the TOR circuit. Once a bridge has been reached (3), a special *handshake* that is unique to TOR occurs and then client's TOR software contacts other available TOR nodes, after securely communicating with the five directory servers, and sends a request to create circuit *cell* to all available nodes listed by the directory servers.

In Fig. 3, after connecting to the bridge and then consulting with the directory nodes to determine available TOR nodes, the software randomly selects three other nodes to form a circuit (or the user can do a selection also). The information is relayed from the directory nodes to the client's software in an encrypted format so that the bridge does not know what nodes are participating in the circuit. Hence, any TOR node only knows two segments on the network: The node preceding it that it accepts cells from, and the node it needs to forward cells to. It is also important to notice the color of each segment shown in the diagram as it has been colored for a purpose that will be explained shortly, but an explanation of how a TOR circuit is built needs to be shown first. When the TOR client determines the participating nodes it has chosen, it then needs to send a "create" cell to each of the nodes without allowing any of the nodes of the presence of each other. This is done through encryption and cell encapsulation as follows:

1. TOR client establishes a secure encrypted link with the first bridge (i.e., first TOR node) using encryption(1) with Cell(1). The segment for which Cell(1) packets are passing through are colored in red.
2. In order to establish a full TOR circuit composed of the bridge and three other nodes, the client software establishes another connection gradually, through the first bridge, to the second TOR node in the segment colored in blue. In fact, the segment colored in blue is composed of Cell(2) using encryption(2).
3. After a successful initialization using Cell(2) with the second TOR node, initialization with TOR node number three is established through the bridge then the second TOR node in order to ensure that communication with all nodes is not advertised to the public. Moreover, notice that the bridge is only aware of the existence of the client and the next TOR node it needs to speak with. However, it is not aware of the third and fourth TOR nodes. One might question the networking logic behind this. To make things clear, consider that all TOR nodes participating in a circuit are actually packet forwarders (except for the last TOR node), whereby these nodes are not aware nor do they care about the destination or shortest path to the destination the client requires. TOR nodes just relay packets from preceding nodes to destination nodes they have been instructed to relay to.
4. The circuit is kept on being built incrementally until the last and fourth node has been reached whereby the latter is called the *exit node*. The exit node is the only node capable of decrypting the content of the encrypted data or request sent by the client sent through the TOR network. The reason behind this is because the exit node is responsible for communicating with the outside world; and hence, requires the exact data and destination information. Once the *exit* node carries out the request of the client and needs to return an answer, then the exit node sends the information in an encrypted *cell* format that only the client is able to decrypt. Additionally, all the *cells* along the way back are not aware of the contents of the cell which the *exit* node has sent back to the client.
5. Throughout the above points, the data being sent to nodes has been referred to as encryption(x) and cell(x) sent to node(x). TOR utilizes private and public keys where any entity in the TOR network has both. Of course, when information needs to be sent to an entity one usually encrypts data with the public key of the second party so that the second party is the only entity capable of decrypting the data. TOR works exactly the same way the colors presented in the last diagram are now going to be explained. When the client needs to establish a secure link with the bridge, it sends a cell(1) to node(1) – the first node - using public/private key encryption method-

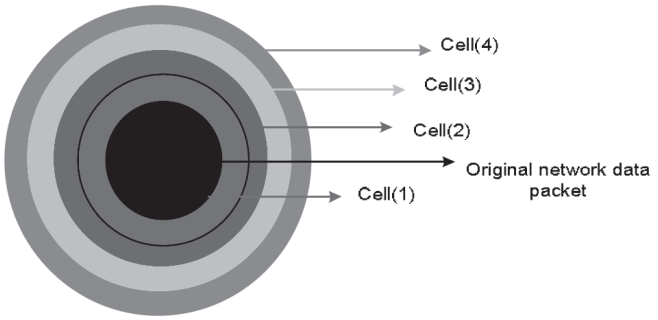

Fig. 4. Client sending and receiving cells to TOR bridge node.

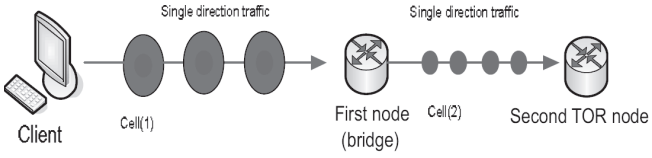

Fig. 5. First stages of encapsulated cells between nodes in TOR.

ology. Hence, any cell sent between the client and the first TOR node is encrypted. During the process of establishing a TOR circuit, *create cells* sent to the participating TOR nodes are also encrypted and relayed through already establish TOR nodes as in Fig. 4.

Communication is then established with the bridge, now the client needs to establish a connection with the second TOR node through the newly established connection with the bridge. The client acquires the public key of the second TOR node and then designs a cell in the shape of an onion. The inner part of the cell contains information encrypted with the public key of node 2 and the outer layer is encrypted with the public key of the bridge. Once the bridge receives this cell it will peal (decrypt) the outer layer and then will pass the remainder of the still encrypted cell to TOR node number 2 as illustrated in Fig. 5.

A circuit composed of four nodes hence has four different types of cells which are encapsulated in each other and only a single node understands one layer of this encapsulation (i.e., can decrypt and understand the content of the cell). When a circuit is formed it is the duty of the TOR client software to design the encapsulated cells hence called onions before sending them to the circuit. All data pertaining to the identity of the client are stripped from the cells and therefore the client becomes anonymous whereby the bridge is the only entity that knows of the client's existence (not even the *exit* node). Similarly, it is also the duty of the exit node to encapsulate and design an onion cell that can be reversely decrypted on the way back as an answer to the client's request(s). Data in an onion or encapsulated cell is illustrated in Fig. 6.

The network path for the onion in Fig. 5 that is passing through the TOR circuit via the TOR nodes is now represented in Fig. 7.

When the network packets originating from the client are sent through the fourth node they are no longer encrypted, as the fourth node has removed the last layer of encryption from the onion. The receiving entity will now be contacted by the fourth

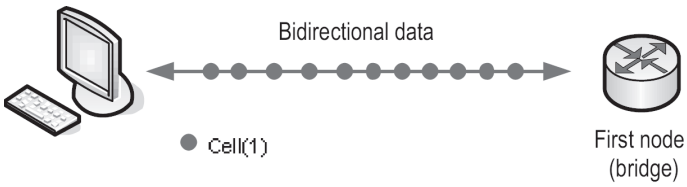

Fig. 6. A sample of multiple encapsulated cells in a TOR network.

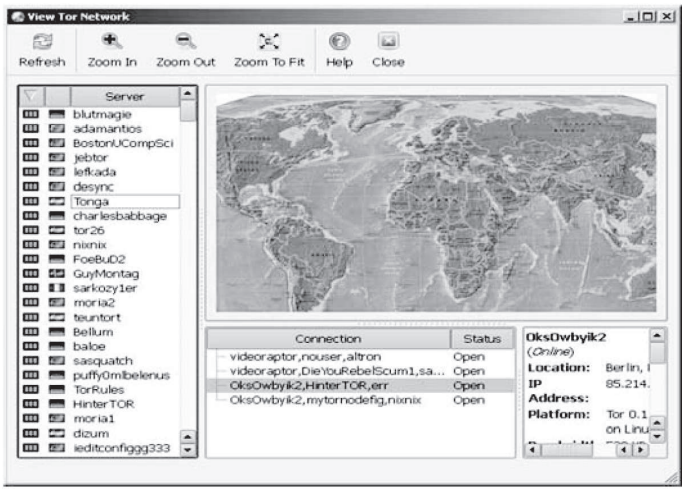

Fig. 7. A representation of the path of an encapsulated cell in a TOR tunnel.

node, hence hiding the identity of the client, and the data sent back to the client will traverse backwards along the same path data has come from. When data reaches its final destination, only the client is able to decrypt and view the data. Hence, in any TOR communication, only the bridge knows of the existence of the client in a circuit and only the *exit* node is able to reveal the data but not the identity of the client.

C. TOR Features

TOR has many features that make it attractive. For a detailed list, users are referred to [12]. These features include:

- **Ease of use through socks proxy**

TOR has been built in a way that allows users of different backgrounds to use TOR easily and anonymously. TOR also relies on applications with socks proxy features in order to redirect any application's traffic through a single tunnel to the anonymous TOR network. This allows all applications to benefit from encryption standards TOR is using. Moreover, all desktop applications are unaware of the stages of TOR and how data is encrypted/decrypted or even how cells are formed. Once a user sets his/her application to the TOR socks proxy settings, then the TOR engine is installed.

- **Protection against strong and weak attacks**

The designers of TOR admit that the anonymous network does not prevent against global adversaries that have exclusive network/resource access and are capable of monitoring traffic on all networks their users are connected to. However, TOR promises protection against strong and weak attacks from individuals and other entities with malicious attack techniques carried out on non-technical and sometimes unprotected end users. Con-

sequently, preventing against traffic analysis and assuring the integrity and confidentiality of data being transmitted over the Internet for and by users and therefore hiding the identity of recipients and senders is at the moment the concern of the TOR project. Many types of attacks have been carried out on TOR since it was introduced like basic traffic analysis, path confirmation attack, insertion attack, predecessor attack, and backtrack attack [11].

D. Critique

TOR is a unique anonymous design has that the following advantages:

- **No single TOR node is aware of the complete plan of communication**

When a circuit is formed TOR nodes participating in that circuit are only aware of the predecessor for which they have received traffic from, and the next node they need to relay traffic to (except for the last node that is able to identify the content of the cell, but not the identity of the sender). Hence, even if a single node is compromised or is acting maliciously by gathering data about traffic being relayed and analyzing such traffic, then that TOR node can only know little in order not to compromise the content of such traffic as well as identify the sender and receiver.

- **The more the number of TOR nodes the more anonymity added**

Similar to any graph model, the more the number of vertices the more the number of edges needed to create different interconnections. Hence, the more the number of TOR server nodes participating in a TOR network, and the more the number of TOR users relaying TOR traffic (through TOR clients), then the more the possible number of circuits that can be established and can therefore pass information securely along TOR paths.

- **TOR builds anonymous paths for the client based on a list of bridge nodes**

When a client is requesting to establish a circuit, then an encrypted list of all available bridges is downloaded from one of the five management nodes and then decrypted at the client level in order to establish the first hop onto the TOR circuit. Once the first hop is established with the bridge, the next TOR nodes are contacted gradually hence adding even more security to establishing circuits as opposed to contacting circuits individually.

On the other hand, TOR has several disadvantages. These are:

- **QoS considerations for dynamically assigning certain traffic to certain categories due to packet encryption**

Many users choose to abuse the TOR network unintentionally by either having large downloads being streamed to their workstations using P2P applications, or by simply downloading large files over HTTP. Once some users choose to use low latency applications like Telnet, SSH, and VoIP user experience becomes horribly unbearable due to latency and link congestions. NetCamo had solved this problem by introducing a QoS requirement for every connection that needs to traverse the anonymous system. However, connections that require a certain QoS requirement that cannot be met at the time of the request are dropped and therefore cannot be served. Since NetCamo is limited to a certain number of predefined nodes where circuits are established only once and do not frequently change, then dropping incoming connections with high QoS requirements is the only way possible to assure other connections have a reliable bandwidth to deliver their data stream [13]. Since TOR can easily shift and change network paths by reestablishing new circuits, then QoS requirements can be met based on the connection protocol.

- **Directory information servers can be blocked**

The directory information servers keep track of all participating TOR nodes, as well as, the bridges users are allowed to connect to. Moreover, the lists of participating TOR nodes that have been found reliable and participating frequently are usually posted on the TOR website. Hence, if any authority wishing to stop the usage of TOR by its users, then the lists of all available TOR servers as well as the directory servers are readily available to be simply blocked by the firewall of that organization.

- **Central servers for TOR discovery that lists TOR servers and guarantees connectivity for TOR servers**

As there are centralized directory servers for managing the TOR network and communication, this seems an advantage; however, it is not. Although TOR designers admit that TOR nodes/servers/client are all susceptible to attackers who are categorized as global adversaries, a single attacker with access to an organization's Internet gateway is able to cause a serious threat to individuals using TOR inside the organization's network. An attacker can simply fake the identity of the TOR directory servers by redirecting traffic to a local server and then generating a set of public and private keys, and then the attacker can maliciously modify a TOR client and then repackage it for users to download. The modified TOR client can simulate connections to other nodes, as in fact it's only connected to the newly introduced server that's faking and feeding false information to the user's TOR client. As a result all traffic being generated from and to the Internet is now in the clear. Of course users with a sufficient technical background may detect such types of malicious attacks; however, to the normal end user, this is undetectable [14].

- **All TOR traffic is pushed through port 9001 TCP can not only be blocked but also detected**

The ability to use TOR relies on socks proxy features found in applications, and while there are many applications that already have this feature implemented; many other applications/services do not. An example is a DNS request that requires the client to resolve against any DNS server outside the TOR network. In this scenario consider a user requesting to visit a website like google.com, once the user enters the domain in his/her browser, a DNS request will be sent to the user's ISP to perform a domain name lookup and resolve that domain name to an IP. This exact process is not anonymous and insecure and hence allows any snooper to perform time-attacks and learn that the user is, at this point in time, generating traffic and accessing google.com. To some applications that rely on DNS and do not support socks proxy makes TOR useless in some scenarios or where end-to-end attacks are possible. Therefore, TOR cannot prevent against end-to-end attacks [15].

- **Open design, Open Source Code**

The fact that TOR is an open source implementation makes TOR a widely acceptable anonymous piece of software. It presents true security to users by revealing code internals. Due to this fact also some professional attackers are capable of taking advantage

of this by introducing malicious code into some clients and then performing attacks on the TOR network. Of course this does not mean that TOR must be a closed source to add more protection to the TOR network and users, security through obscurity is not intended here; however, a thought needs to be given to this model.

- **Slow Performance**

Due to most users being end users with asymmetric connections like DSL/ADSL with limited traffic, makes users prefer choosing high bandwidth dedicated TOR nodes instead of users who have chosen to become TOR relayers with poorer bandwidth connections. This in turn not only lessens anonymity but also adds more load on TOR dedicated nodes, as well as, security risks and reliability. An attacker may simply deploy a large number of dedicated TOR servers; thus, users would willingly join these servers and risk traffic analysis being carried out on their connections.

- **Success or failure in data integrity checks**

This may render a circuit useless an attacker with enough skill can cause serious degradation in TOR's communication experience through two scenarios for which one was proven successful by Keven Bauer [16].

- **Website fingerprinting and backtrack attack**

This is due to lack of packet camouflaging, delay, and reordering [17], [18].

IV. CONCLUSION

This paper presented the TOR anonymous system and its corresponding details that have made such a system a success. Avoiding traffic analysis, and hiding the identities of users, is the aim of any anonymous system. However, since most anonymous systems rely on aging encryption technologies for which global adversaries are a capable of compromising, then the integrity of data might be at stake.

One of the key elements that worry anonymous systems researchers is QoS for the bandwidth utilized by peers on the systems and the overall network performance. Although this has been slightly commented on, more research in QoS and a bandwidth-choking approach is required while concentrating on security and functionality implications. Future work involves conducting a detailed study and comparison of the various anonymous systems.

REFERENCES

[1] S. Syverson, D. Goldschlog, and M. Reeds, "Anonymous connections and onion routing," in *Proc. IEEE SP*, Oakland, USA, 1997, pp. 482–494.

[2] (2014, June 20) *TOR - Onion routing online documentation* [Online]. Available: https://www.torproject.org/

[3] B. Choi, D. Xuan, C. Li, R. Bettati, and W. Zhao, "Efficient traffic camouflaging in mission-critical QoS guaranteed networks," in *Proc. IEEE Information Assurance and Security Workshop*, West Point, Virginia, USA, 2000, pp. 143–149.

[4] R. Dingeldine, "TOR: the second-generation onion router," in *Proc. Usenix Security Symposium*, San Diego, USA, 2004.

[5] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *ACM Commun.*, 24(2), pp. 84–88, 1981.

[6] (2014, Apr. 30) *How anonymizers work* [Online]. Available: http://www.livinginternet.com/i/is_anon_work.htm

[7] (2014, Apr. 30) *JAP anonymity and privacy* [Online]. Available: http://jap.inf.tu-dresden.de/index_en.html

[8] G. Danezis, R. Dingeldine, and N. Mathewson, "Mixminion: design of a type III anonymous remailer protocol," in *Proc. IEEE Symposium on Security and Privacy*, Berkeley, USA, 2003, pp. 2–13.

[9] M. Freedman, S. Sit, J. Cates and R. Morris, "Introducing Tarzan, a peer-to-peer anonymizing network layer," in *Proc. First Int. Workshop on Peer-to-Peer Syst.*, Cambridge, 2002.

[10] M. Rennhard and B. Plattner, "Introducing MorphMix: peer-to-peer based anonymous internet usage with collusion detection," in *Proc. ACM Workshop on Privacy in the Electronic Society,* Washington, USA, 2002, pp. 91–102.

[11] P. Syvneron, G. Tsudik, M. Reed, and C. Landwehr, "Towards Analysis of Onion Routing Security," in *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability,* Springer-Verlag, LNCS, 2009, pp. 96–114.

[12] B. Zantout and Ramzi A. Haraty. *Avoiding Traffic Analysis and Assuring Data Integrity using a Quorum-based Approach*. The Fifth International Workshop on Advanced Computations for Engineering Applications. Taif, Saudi Arabia. Mar. 2010.

[13] B. Zantout and R. Haraty, "A comparative study of Bittorrent and NetCamo data communication systems," International Journal of Computational Intelligence and Information Security, volume 1, number 2, March 2010.

[14] S. Chakravarty, "Traffic Analysis Attacks and Defenses in Low Latency Anonymous Communication," Ph.D. dissertation, Columbia Univ. 2014.

[15] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of TOR," in *Proc. IEEE Security and Privacy Symp.*, May 2006.

[16] K. Bauer, D. McCoy, D. Grunwald, S. Douglas, and K. Tadayoshi, "Low resource, routing attacks against anonymous systems," Tech. Rep. CU-CS-1025-07, University of Colorado, USA, 2007.

[17] A. Hints, "Fingerprinting websites using traffic analysis," in *Privacy Enhancing Technologies*. Springer-Verlag, LNCS 2482, 2002, pp. 171–178.

[18] B. Zantout and R. Haraty, "I2P data communication system," in *Proc. 10th Int. Conf. Netw.*, 2011, pp. 401–409.

Ramzi A. Haraty is an Associate Professor of Computer Science in the Department of Computer Science and Mathematics at the Lebanese American University in Beirut, Lebanon. He serves as the Program Administrator for the Middle East Program Initiatives (MEPI) Leaders for Democracy Fellowship program. He is also the Internship Coordinator for MEPIs Tomorrows Leader program. He received his B.S. and M.S. degrees in Computer Science from Minnesota State University - Mankato, Minnesota, and his Ph.D. degree in Computer Science from North Dakota State University - Fargo, North Dakota. His research interests include database management systems, artificial intelligence, and multilevel secure systems engineering. He has well over 110 books, book chapters, journal and conference paper publications. He supervised over 110 dissertations, theses, and capstone projects. He is a Member of the Association of Computing Machinery, Institute of Electronics, Information and Communication Engineers, and the International Society for Computers and Their Applications.

Bassam Zantout is a Mid-Tier SSE - Virtualization and Storage at EMC in Riyadh, Saudi Arabia. He foucuses on designing and administrating private cloud infrastructures, featuring multi-platform high availability environments, and encompassing a variety of vendor products. He received his B.S. and M.S. degrees in Computer Science from the Lebanese American University in Beirut, Lebanon. His research interests include cloud computing and infrastructure security.