acceleraTOR: Tor in the Fast Lane

Spring15 : 18-731 Network Security - Final Report

Antonio Rodrigues
antonior@andrew.cmu.edu

Saurabh Shintre
sshintre@andrew.cmu.edu

Soo-Jin Moon
soojinm@andrew.cmu.edu

Zijie Lin
zjlin@cmu.edu

***Abstract*—The abstract goes here.**

I. Introduction

Anonymity refers to the unlinkability of the source or/and destination of a communication activity, or the unlinkability of different communication activities belonging to the entity [1]. Anonymity has become an important property in the presence of censorship, privacy breaches by intelligence agencies, and pestering personalized content/advertisement delivery by web-based companies. Simultaneously, anonymous communication is being used for organized crime or buying/selling of contraband substances, such as the popular online marketplace SilkRoad. With increasing applications and user-base, a number of anonymous networks have been developed such as Mixminion[1], Crowds [2], etc. For low-latency applications, such as web-browsing, Tor [3] remains the most popular and widely-based anonymous network.

Tor's design includes a Tor client, 'Onion' routers (ORs), and directory servers. All available ORs maintain live TLS connections between each other. When a Tor client wishes to connect to a server such as a website, it randomly picks three ORs from the list of available ORs and initiates Diffie-Hellman Key Exchange with each OR sequentially. Data packets are divided into cells of fixed size (called 'onions') and each onion is encrypted in a layered way with the keys exchanged with each selected OR. As each OR receives the onion, it strips down the outermost layer, which is encrypted with its shared key, and forwards the remaining onion to the next OR. If an OR is the last one on the path – known as the 'exit node' – it reconstructs the packet and forwards it to the website. Data from the website follows the reverse path. While this process only provides data confidentiality, prevention from traffic analysis is provided due to the fact that each OR forwards packets belonging to different traffic streams and therefore, the adversary cannot perform traffic analysis by observing few links/ORs in the Tor network. Additionally, Tor provides hidden services to enable a server to provide service without revealing its IP address or identity.

Despite its popularity, Tor has certain weaknesses which either have been used to breach anonymity or limits its effectiveness in other relevant applications. Unlike mix networks [4], Tor does not perform any explicit buffering, delaying, or reordering of onions. This means that in the absence of sufficient traffic the traffic patterns created at source travel deep into the tor network without significant modifications and can be identified there. Murdoch and Danezis [5] used this to launch a side channel attack that reveals the ORs used by a certain anonymous traffic stream. Since, Tor uses the same path for multiple traffic streams belonging to the same client in order to avoid path setup delay, this attack can be used to correlate two communication activities of the same client breaching its anonymity. In spite of optimization done to minimize packet delay, cryptographic processing of numerous packets leads still induces packet delay of the order some milliseconds which makes Tor unsuitable to other applications that have stringent delay requirements, such as VoIP and streaming.

Certain solutions to reduce the packet or path setup delay in Tor have been proposed. LASTor [6] is a modification in the client side of Tor, which ensures that OR chosen for a path are not separated demographically so ensure low transmission delay. Christin et al. [7] proposed to reuse the path chosen by a hidden service provider to reach hidden service's directory server for the rendezvous point for the server and the client. While these solutions indeed provide reduction in packet/path setup delay they require fundamental change in Tor's design and have implications on the anonymity guarantees of Tor. In this work, we intend to reduce overall delay of Tor with the use of Intel's DPDK packet processing framework. (...) DPDK uses optimized functions of handling batches of data and uses specialized data structures, such as ring buffers, to achieve significant improvement in packet processing delay. This project explores benefits of DPDK translate to better performance for Tor and the impact of this approach on Tor's anonymity guarantees.

II. acceleraTor Design and Implementation

In this section we describe acceleraTor's design in some detail and briefly explain relevant implementation issues.

A. Design Overview

acceleraTor's overall design is depicted in Figure 1: we compare it to the design of 'vanilla' Tor to clearly emphasize their differences.

At the extremes of Figure 1 b) we have acceleraTor (top), a TCP-based network application which implements Tor OR logic; and DPDK's user-level packet I/O libraries (bottom), which read/write packets directly from DPDK-compatible Network Interface Cards (NICs). The main difference with respect to 'vanilla' Tor is that all levels in acceleraTor's design run in user space. In-between, we recur to a user-level TCP stack – mTCP [8] – to interconnect acceleraTor and DPDK, and ultimately reserve the 'services' of a single DPDK-compatible NIC to the application. Although not part of

[1] http://mixminion.net/

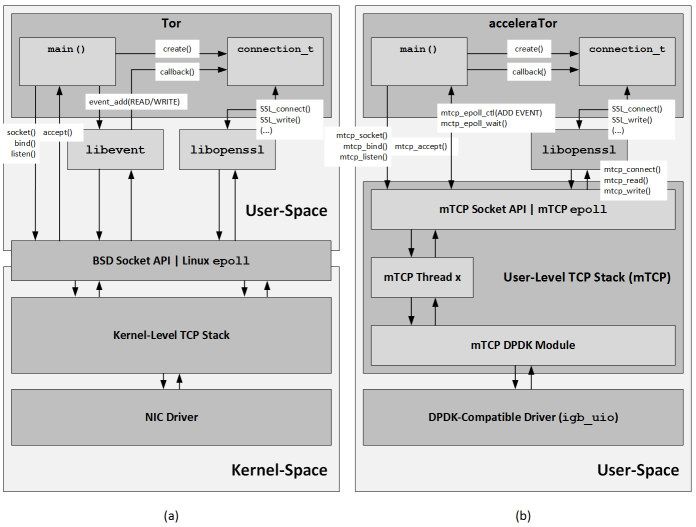

Fig. 1: 'Vanilla' Tor (a) vs. acceleraTor's (b) design overview.

acceleraTor's design per se, this application-NIC 'exclusivity' led us to run acceleraTor in nodes equipped with two NICs, one to be handled by a DPDK-compatible driver and reserved for acceleraTor, another handled by a Linux kernel driver, which allows for the parallel operation of 'usual' networking applications.

In the next subsections, we discuss design and implementation aspects for each one of the levels in more detail.

B. Tor's Integration with mTCP

Some of the benefits brought about by DPDK come from a bypass of the kernel networking stack, allowing applications to process 'raw' packets at the user-level and thus overcome kernel-level inefficiencies such as heavy system call overhead [8], [9]. Nevertheless, such a 'bypass' also involves challenges. Without kernel-level support of TCP, we either:

1) Modify Tor to become independent of TCP, having it directly process packets 'fresh' out of the NICs;
2) Develop/use an user-level TCP stack, integrated with DPDK, on top of which Tor can be adapted.

Option 1 would heavily modify the way Tor works – e.g. Tor makes use of TLS, which depends on TCP's reliable transport – not only making acceleraTor ORs incompatible with 'vanilla' Tor ORs, but also making modifications to Tor unnecessarily intrusive and harder. Option 2 is more desirable: ideally, it should rely on an intermediate module which (1) interfaces with DPDK to read/write 'raw' packets from/to NICs, (2) implements "enough of" TCP to have TCP-based applications and TLS work on top of it; and (3) exposes a 'BSD-like' socket API, allowing for easy integration of TCP-based applications and libraries (e.g. OpenSSL[2] for TLS).

mTCP [8] is a user-level TCP stack which – while specifically designed to solve TCP performance issues in multicore systems – fulfills the points of the 'ideal' option 2: as of version 3[3], mTCP comes directly integrated with DPDK (v1.8 and v2.0). Other choices have been quickly surveyed – e.g. drv-netif-dpdk[4], a DPDK interface for Rump Kernels [10]; the solution proposed by Kantee et al. in [11]; or NUSE [12] – but dismissed since these could not beat mTCP's suitability to the aforementioned requirements.

Despite its convenience, the use mTCP to integrate Tor and DPDK poses the following set of implementation challenges, some of which are briefly addressed in the following subsections:

- mTCP's API slightly differs from the 'BSD' socket API used in Tor, and does not provide all its functionalities. acceleraTor must accommodate these limitations without major loss of functionality.
- In its 'vanilla' version, Tor makes use of `libevent`[5] to handle asynchronous input/output (IO). mTCP provides its own `epoll`-like event system to monitor mTCP sockets, not compatible with `libevent`[6]. Therefore, Tor's asynchronous IO logic must be changed.
- Tor makes heavy use of Transport Layer Security (TLS) connections, through OpenSSL, which must now be modified to comply with mTCP's API.

1) Porting Tor to mTCP: Since mTCP provides a 'BSD-like' socket API, most of the porting effort consisted in identifying the code using 'BSD-like' system calls and replacing these mTCP's equivalent calls. Such changes have been kept as contained and isolated as possible: we allow for the selective compilation of either 'vanilla' Tor or acceleraTor via a new Tor configuration option[7].

Besides 'syscall-translation', the porting effort also included the initialization of a single mTCP thread from within Tor's main thread, which ultimately 'binds' to an available DPDK-compatible NIC. Tor's main loop was changed to continuously wait for mTCP events (e.g. new connections, read/write events on existing connections) using mTCP's user-level event system : in Tor's original design, a similar approach is used to listen for new events, using `libevent`'s API instead. More details are given in section II-B2.

2) Event Handling Logic: Tor uses asynchronous input/output (IO) to monitor sockets associated with inter-OR TCP connections and invoke appropriate connection-handling callbacks whenever read/write events are triggered. mTCP provides its own `epoll`-like event system to monitor mTCP connections, not compatible with `libevent`, used by 'vanilla' Tor.

When adding creating a new OR/client connection, Tor registers new read/write events to monitor the file descriptors (i.e. sockets) associated with the connection. `libevent`'s API allows for a direct association between the registered event

[2]https://openssl.org/

[3]Released in April 2, 2015, https://github.com/eunyoung14/mtcp.

[4]https://github.com/rumpkernel/drv-netif-dpdk

[5]http://libevent.org/

[6]I.e. `libevent` does not recognize the event-checking 'method' made available by mTCP's event system.

[7]Source code and setup scripts available at https://bitbucket.org/clockWatchers/accelerator

and a specific connection descriptor. The same is not possible in mTCP: mTCP's API associates events with a socket file descriptor. We therefore need some 'external' way of mapping socket file descriptors to the respective connection descriptor objects, and then re-use already existing read/write callbacks. We accomplish this via a global hash table, implemented using `uthash`[8].

3) Integration with OpenSSL: Last – but definitely not least – there is the issue of TLS: Tor makes heavy use of TLS to secure OR/client connections, and in turn uses OpenSSL's API to get TLS support. OpenSSL must then be also ported to mTCP. As OpenSSL's implementation relies on 'BSD'-like system calls, the porting effort mostly consists in tracking down the code which uses such system calls and replace them with those exposed by mTCP API. Besides the 'translation' effort, we introduce the changes in OpenSSL's API:

a) Added a new attribute in SSL structures (i.e. representations of a TLS connection), `mctx_t mctx`, which represents the mTCP thread context to which the SSL structure should be associated with.

b) Added a new call to OpenSSL's API

```
int SSL_set_mtcp_ctx(SSL * s, mctx_t mctx)
```

which lets accelaraTor (or other mTCP-based applications willing to use OpenSSL) set the mTCP thread context attribute on some SSL structure `s`. These changes are convenient, as other calls exposed by OpenSSL's API – e.g. `SSL_connect(SSL * s)` – which originally only take a single SSL structure as argument, do not have to be changed to include an additional reference to the respective mTCP context as argument. This eases the integration effort on Tor's own source code ('wrapper' functions for OpenSSL's API, defined in `tortls.c`).

C. mTCP and DPDK

As of version 3, mTCP comes directly integrated with DPDK (v1.8 and v2.0), and so the relevant issues during development were implementation-related, mostly consisting in finding compromises with (OS- and hardware-level) constraints imposed by both mTCP and DPDK.

- DPDK-compatible mTCP (v3) could only work with the Linux-3.13.0 kernel (we have used 3.13.0-46-lowlatency, which comes with the `uio` kernel module installed by default);
- We have tested setups in both 32- and 64-bit architectures: in the latter case, DPDK's libraries had to be compiled as shared;
- DPDK is compatible with a limited set of NICs: we have successfully tested DPDK with Intel 82545 EM[9]and Intel 82599[10] Ethernet adapters.

[8]http://troydhanson.github.io/uthash/

[9]Oracle Virtual Box: https://www.virtualbox.org/.

[10]AWS EC2 `c4.large` instances.

III. Evaluation

IV. Related Work

Tor is a widely used anonymous network on the Internet. There is a large body of work which have attempted to improve Tor's performance. Some approaches include the refinement of Tor's relay selection strategy [6], [13], [14]. Akhoondi et al. [6] have proposed an approach that brings performance gains by trading off between the anonymity and latency. Such approaches are orthogonal to our approach and can be used in conjunction with acceleraTor as we do not modify the path selection algorithm. Jansen et al. [15] propose a socket management algorithm that uses real-time kernal information to compute the amount of information to write to each socket to avoid congestion bottleneck which is claimed to occur in egress kernel socket. Torchestra [16] uses TCP connections to carry bulk traffic in order to isolate effects of congestions betwen different traffic classes and their proposal is based on the analysis that bulk downloads cause delays for interactive traffic. AlSabah et al. [17] proposed DefenestraTor which modifies traffic management in Tor relays to reduce queuing delays. Panchenko et al. [18] proposed path selection algorithms to choose the path inferring from available bandwidth on each relay. As described, many works have focused on the choosing of path selection algorithms but authors [19] have also studied the tradeoff between improved performance and anonymity. However, our approach does not deal with path selection algorithms, thus will not hurt anonimity due to having low diversity in path selection. Other approaches to improve performance of Tor have proposed incentive-based means to users to relay Tor circuits [20], [21]. Nowlan et al. [22] proposed the use of uTCP and uTLS [23] to tackle the head-of-line blocking problem between Tor cirtuis sharing TCP connections to relax the in-order delivery assumptions of TCP.

V. Conclusions and Future Work

References

[1] A. Pfitzmann and M. Köhntopp, "Anonymity, unobservability, and pseudonymity — a proposal for terminology," in *Designing Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, H. Federrath, Ed. Springer Berlin Heidelberg, 2001, vol. 2009, pp. 1–9. [Online]. Available: http://dx.doi.org/10.1007/3-540-44702-4_1

[2] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM Trans. Inf. Syst. Secur.*, vol. 1, no. 1, pp. 66–92, Nov. 1998. [Online]. Available: http://doi.acm.org/10.1145/290163.290168

[3] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-generation Onion Router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, p. 21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251375.1251396

[4] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–90, Feb. 1981. [Online]. Available: http://doi.acm.org/10.1145/358549.358563

[5] S. J. Murdoch and G. Danezis, "Low-Cost Traffic Analysis of Tor," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, ser. SP '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 183–195. [Online]. Available: http://dx.doi.org/10.1109/SP.2005.12

[6] M. Akhoondi, C. Yu, and H. Madhyastha, "Lastor: A low-latency as-aware tor client," *Networking, IEEE/ACM Transactions on*, vol. 22, no. 6, pp. 1742–1755, Dec 2014.

[7] N. Christin, "Traveling the Silk Road : A Measurement Analysis of a Large Anonymous Online Marketplace," 2013.

[8] E. Jeong, S. Wood, M. Jamshed, H. Jeong, S. Ihm, D. Han, and K. Park, "mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, 2014, pp. 489–502. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/jeong

[9] J. D. Brouer, H. F. Sowa, D. Borkmann, and F. Westphal, "Challenge 10Gbit/s," pp. 1–33, 2014. [Online]. Available: http://people.netfilter.org/hawk/presentations/nfws2014/dp-accel-10G-challenge.pdf

[10] R. K. Project, "Rump Kernels," p. 1, 2014. [Online]. Available: http://rumpkernel.org/

[11] A. Kantee, "Environmental Independence : BSD Kernel TCP / IP in Userspace," in *Asia BSDCON 2009*, 2009, pp. 1–10. [Online]. Available: https://2009.asiabsdcon.org/papers/abc2009-P5A-paper.pdf

[12] S. Han, "NUSE: Networking Stack in Userspace?" p. 1, 2014. [Online]. Available: https://www.eecs.berkeley.edu/~sangjin/2013/01/14/NUSE.html

[13] M. Sherr, A. Mao, W. R. Marczak, W. Zhou, B. T. Loo, M. Sherr, A. Mao, W. R. Marczak, W. Zhou, and B. Th, "A3: An extensible platform for application-aware anonymity," in *17th Annual Network & Distributed System Security Symposium (NDSS*, 2010.

[14] R. Snader and et al., "A tune-up for tor: Improving security and performance in the tor network," 2008.

[15] R. Jansen, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "Never been kist: Tor's congestion management blossoms with kernel-informed socket transport," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 127–142. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/jansen

[16] D. Gopal and N. Heninger, "Torchestra: Reducing interactive traffic delays over tor," in *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*, ser. WPES '12. New York, NY, USA: ACM, 2012, pp. 31–42. [Online]. Available: http://doi.acm.org/10.1145/2381966.2381972

[17] M. AlSabah, K. Bauer, I. Goldberg, D. Grunwald, D. McCoy, S. Savage, and G. M. Voelker, "Defenestrator: Throwing out windows in tor," in *Proceedings of the 11th International Conference on Privacy Enhancing Technologies*, ser. PETS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 134–154. [Online]. Available: http://dl.acm.org/citation.cfm?id=2032162.2032170

[18] A. Panchenko and J. Renner, "Path selection metrics for performance-improved onion routing," in *Applications and the Internet, 2009. SAINT '09. Ninth Annual International Symposium on*, July 2009, pp. 114–120.

[19] A. Panchenko, L. Pimenidis, and J. Renner, "Performance analysis of anonymous communication channels provided by tor," in *Proceedings of the 2008 Third International Conference on Availability, Reliability and Security*, ser. ARES '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 221–228. [Online]. Available: http://dx.doi.org/10.1109/ARES.2008.63

[20] R. Jansen, N. Hopper, and Y. Kim, "Recruiting new tor relays with braids," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 319–328. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866344

[21] R. Jansen, A. Johnson, and P. Syverson, "Lira: Lightweight incentivized routing for anonymity."

[22] M. F. Nowlan, D. I. Wolinsky, and B. Ford, "Reducing latency in tor circuits with unordered delivery," in *Presented as part of the 3rd USENIX Workshop on Free and Open Communications on the Internet*. Berkeley, CA: USENIX, 2013. [Online]. Available: https://www.usenix.org/conference/foci13/workshop-program/presentation/Nowlan

[23] M. F. Nowlan, N. Tiwari, J. Iyengar, S. O. Amin, and B. Ford, "Fitting square pegs through round pipes: Unordered delivery wire-compatible with tcp and tls," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, pp. 383–398. [Online]. Available: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/nowlan