Poster: Bento: **Bringing Network Function Virtualization to Tor**

Michael Reininger
University of Maryland

Arushi Arora
Purdue University

Stephen Herwig
University of Maryland

Nicholas Francino
University of Maryland

Christina Garman
Purdue University

Dave Levin
University of Maryland

ABSTRACT

Tor is a powerful and important tool for providing anonymity and censorship resistance to users around the world. Yet it is surprisingly difficult to deploy new services in Tor—it is largely relegated to proxies and hidden services—or to nimbly react to new forms of attack. Conversely, "non-anonymous" Internet services are thriving like never before because of recent advances in programmable networks, such as Network Function Virtualization (NFV) which provides programmable in-network middleboxes.

This work seeks to close this gap by introducing programmable middleboxes into the Tor network. In this architecture, users can install and run sophisticated "functions" on willing Tor routers, further improving anonymity, resilience to attack, performance of hidden services, and more. We present the design of an architecture, Bento, that protects middlebox nodes from the functions they run—and protects the functions from the middleboxes they run on. Bento does not require modifications to Tor, and can run on the live Tor network. Additionally, we give an overview of how we can significantly extend the capabilities of Tor to meet users' anonymity needs and nimbly react to new threats.

ACM Reference Format:
Michael Reininger, Arushi Arora, Stephen Herwig, Nicholas Francino, Christina Garman, and Dave Levin. 2020. Poster: Bento: Bringing Network Function Virtualization to Tor. In *2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3372297.3420020

1 INTRODUCTION

Anonymity systems are critical in achieving free, open communication on today's Internet. In particular, Tor [4] has become a staple in resisting online censorship by rogue nations and allowing journalists to safely communicate with sources world-wide [10]. However, there is a surprisingly narrow set of services that Tor is able to support in a robust fashion. Today, the use of Tor is largely relegated to web proxies and hidden services [13], and, unfortunately, neither of these applications has the ability to scale to handle dynamic workloads or attacks by automated bots [9]. Moreover, Tor has had to wage multiple concurrent arms races to combat a wide slew of attacks like website fingerprinting and bridge node detection—unfortunately, rolling out new defenses can be slow and cumbersome. Conversely, services on the standard, "non-anonymous" Internet are thriving like never before. Impressive innovations in network function virtualization (NFV) [12] in particular have resulted in more robust, scalable, and resilient network services. The present and future Internet is comprised of *programmable networks*, but there do not exist the basic primitives to achieve such features in anonymous networks.

We introduce Bento, a new architecture that augments the Tor anonymity network with user-programmable "middleboxes." Briefly, Bento allows clients to write sophisticated middlebox "functions" in a high-level language (Python, in our implementation) and run them on willing Tor routers. We propose a wide diversity of functions that significantly improve various aspects of Tor, including: a `Browse` function that offloads a client's web browser to avoid website fingerprinting attacks, a `LoadBalance` function that automatically scales hidden service replicas up and down to handle varying load, and a `Dropbox` function that allows Tor to be used as an anonymous file store. Through these many functions, we show that making Tor more programmable carries significant promise.

Naturally, a more programmable Tor network also introduces several important challenges regarding how to achieve it *safely*. In particular, we must be careful to protect middlebox nodes from the functions they run on others' behalf—and we must protect functions from the middleboxes who run them. To address these concerns, Bento's design will employ sandboxes as well as recent advances in deploying legacy software in trusted secure enclaves [5].

Bento is fundamentally an architecture: its goal is to provide the core abstractions and mechanisms upon which sophisticated, anonymous systems can be built. One of our contributions is identifying and presenting a design for the critical components that collectively support programmable middleboxes: **Composable functions**: Bento provides mechanisms for installing user-defined functions on Tor nodes, and chaining functions together to perform more complex tasks; **Safe execution environments**: Bento employs an execution environment that is *general-purpose* enough to enable a wide range of middlebox functions, but also *safe* in the presence of rogue developers or malicious middlebox nodes; **Middlebox node policies**: Tor node operators should be able to express what sorts of things they are willing to do on behalf of functions. We introduce *middlebox node policies* that represent what a user is willing to allow functions to do, and enforcement mechanisms to ensure them.

Threat model Our architecture is deployed on top of the existing Tor network (therefore, it is incrementally and immediately deployable), and as a result we adopt the same network-level threat model as in traditional Tor. This can vary by user and application, but a

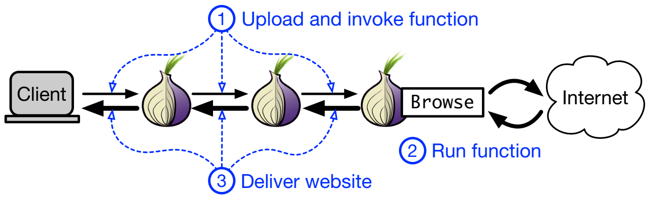

Figure 1: Overview of installing, and executing a `Browser` "function" that runs on another Tor node, downloads a given URL, and delivers it, padded to some threshold number of bytes. Note that, from the perspective of an attacker sniffing the client's link, the client uploads a small amount and then downloads a large amount.

common assumption is that of a powerful routing-capable adversary [11], such as a nation-state. Such an adversary often controls a large network—and can even influence nearby routes to go through its network—but cannot have a *global* view of Internet traffic. In addition to these routing-capable network-based adversaries, our architecture requires us to consider the threats that can arise from an altogether new mode of interaction: loading and running code on other users' machines. We assume that users naturally have physical access to their machines, and can thus introspect on running processes on their machines. However, we also assume that some Bento middleboxes will have secure enclaves, namely, Intel SGX. We explicitly assume that these environments are safe; that is, for any code or data being executed or stored inside of a secure enclave, we assume that the attacker cannot introspect on either, despite having physical access to the machine.

Attacks In showing one of the potential benefits of Bento, we consider two broad classes of attack and new ways for how to prevent them. First, *deanonymization attacks* [1] seek to infer the two endpoints of a Tor circuit through passive or active traffic analysis. Routing-capable adversaries are very well-suited for these kinds of attacks, as they can influence traffic on the entry leg (between source and entry node) and the exit leg (between exit node and destination) to go through networks they control—at that point, they can perform straightforward traffic correlation attacks [6]. The second broad class of attacks we consider are *fingerprinting attacks* [8]. Typical defenses involve reordering or batching requests and sending junk control packets to make websites appear indistinguishable from traffic patterns alone.

2 OVERVIEW OF BENTO

In this poster, we introduce the first architecture to bring the power of programmable middleboxes to the Tor anonymity network. This section presents a high-level overview of how Bento enables users to extend Tor with programmable functions.

Motivating Example Consider a user, Alice, who wishes to anonymously browse a website over Tor, but who also fears that an adversary who knows her identity has the ability to observe traffic entering and leaving her machine. Such an adversary could launch, for instance, a website fingerprinting attack [2] by correlating traffic patterns with known websites, thereby potentially violating Tor's unlinkability property.

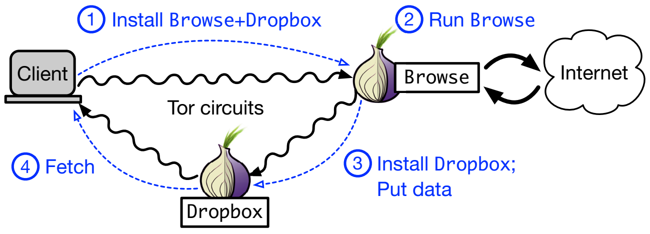

Figure 2: In this motivating example, the user composes two functions: `Browser` which runs a web client to download a website, and `Dropbox` which stores a piece of data to later be fetched.

Approach Typical solutions to this problem would have Alice alter her traffic patterns while visiting the website, in the hopes of confusing the adversary's attempts at fingerprinting. Instead, in Bento, Alice can *offload* processing that would have typically happened on her own machine to *another* node in the Tor network. Figure 1 depicts this; we next describe each step at a high level.

Writing a Function First, Alice writes (or, more likely, downloads) what we call a *function*: a (typically small) piece of code in a high-level, powerful language that is intended to be run on other Tor nodes. These functions can be powerful, but they are constrained to a limited API. Also, critically, they run outside of unmodified Tor—in essence, they are like small servlets running on Tor routers. In our example, Alice's function, `Browser`, is a program that takes as input a URL to download. Upon being invoked with its input, the function starts a web client, autonomously downloads her chosen URL, saves it to a single digest file (e.g., a tarball), and returns the file, padded to the nearest 1MB.

Deploying a Function In Bento, some Tor nodes opt into acting as *middlebox nodes*, who are willing to run (some) functions on behalf of other users. Much like exit nodes, middlebox nodes publicly specify a policy of what function properties they are and are not willing to support. Alice chooses a middlebox node who would be willing to run her function, opens a circuit to it, and, with its permission, installs the function on it.

Executing a Function Once the function is deployed, Alice sends a message over the Tor circuit to the middlebox node to invoke the function on the URL of the site she wishes to visit. Upon receiving this message, the middlebox node executes the function: it downloads the website, packages it into a padded archive, and sends that back to Alice, over the circuit. The attacker observing Alice's communication sees two small uploads from Alice (when she installs the function and when she invokes it), followed by a large download (the padded website). Thus, because Alice is not actively involved during the download of the website, the attacker cannot gain *any* of the informative traffic dynamics that prior fingerprinting techniques require in order to work.

Composing Functions To further thwart the attacker, Alice decides to *go offline completely* during the website download. To achieve this, she *composes* two functions together, as shown in Figure 2. In addition to `Browser`, she also instructs the `Browser` function to deploy, on a separate node, a simple `Dropbox` function

Function	Description	Problem Solved
`Browser`	Runs an application-layer web proxy at the exit node, and delivers padded contents to the client.	Website fingerprinting
`Cover`	Instruct desired relay to create a number of circuits; send cover traffic through each new circuit.	Circuit fingerprinting
`LoadBalancer`	Balance incoming connections among hidden service hosts.	Increased capacity, reliability
`Dropbox`	Allows a node to deposit data; only the node with a matching token can retrieve the data.	Traffic correlation
`Shard`	Takes in a file, breaks it into shards, and encrypts each shard.	Anonymity, low storage

Table 1: Example middlebox functions.

that supports two invocations: a "put" of a data file and a subsequent "get" to retrieve it. Alice then instructs the `Browser` function to deliver the (padded) file to `Dropbox` rather than directly to her. Some time later, she visits the `Dropbox` node to fetch the data. From the perspective of an attacker who can sniff Alice's link, not only would she not provide activity that could be fingerprinted: she need not be online at all while the website was being downloaded!

Why This Helps These motivating use cases show that, with just simple programs, a user could significantly extend the capabilities of the Tor anonymity network. We give examples in Table 1 of a wider variety of functions, which make hidden services more robust, provide cover traffic when needed, and more. However, to *responsibly* achieve this vision, we must adhere to a set of safety and reliability goals, which we outline next.

3 BENTO GOALS

We have four main goals and one non-goal when designing Bento to ensure safe and secure deployment of our programmable functions:

Expressiveness We wish to empower users to write (or use) sophisticated, composable functions. We make use of a high-level programming language with no inherent limitations.

Protect functions from middlebox nodes We must protect users' functions against confidentiality and integrity attacks on untrusted third-party middleboxes. This is similar to the large body of work on making safe use of untrusted third-party compute resources like cloud computing [3] or even Tor itself [7]. To achieve these, we employ recent advances in deploying legacy software in trusted secure enclaves [5].

Protect middlebox nodes from functions We must also protect the users who run the middlebox nodes. Much like how Tor relays can express the destinations for which they wish to serve as exit nodes, middlebox nodes should be able to express policies over the actions they do and do not wish to perform on behalf of other users. Our solution is *middlebox node policies*, which allow middlebox operators to specify which system calls to permit, and how many resources to provide to functions. We aim to enforce these policies by mediating access to all resources.

No Harm to Underlying Tor Deploying Bento should cause no degradation to the existing anonymity properties of Tor. Our functions run on top of Tor, and interface with it via the Stem library.

No Extensions to Tor We aim to sit on top of Tor, and to require no additional user privileges, so as to support more robust applications.

4 CONCLUSIONS

In this work, we have argued that programmable anonymity networks are useful, possible, and challenging, yet attainable. We have sketched an example application and various challenges, as well as some possible avenues to achieve them. Our proposed applications suggest that even a small amount of programmability would significantly improve the speed at which new techniques can be rolled out into the Tor network; for instance, load balancing at introduction points has been a proposal for years, but would be a trivial snippet of code in a more programmable Tor network.

There remain many interesting and important problems that must be solved in order to achieve programmable anonymity networks. We view this work as merely the first step, and we hope that it engenders a lively discussion among the anonymity community as well as application developers who wish to expand the offerings possible on anonymity networks.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants CNS-1816422, CNS-1816802, and CNS-1943240.

REFERENCES

[1] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. 2013. LASTor: A Low-Latency AS-Aware Tor Client. In *IEEE Symposium on Security and Privacy.*

[2] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *ACM Conference on Computer and Communications Security (CCS).*

[3] Michael Coughlin, Eric Keller, and Eric Wustrow. 2017. Trusted Click: Overcoming Security issues of NFV in the Cloud. In *ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization (SDN-NFVSec).*

[4] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium.*

[5] Stephen Herwig, Christina Garman, and Dave Levin. 2020. Achieving Keyless CDNs with Conclaves. In *USENIX Security Symposium.*

[6] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. 2013. Users get routed: Traffic correlation on Tor by realistic adversaries. In *ACM Conference on Computer and Communications Security (CCS).*

[7] Seong Min Kim, Juhyeng Han, Jaehyeong Ha, Taesoo Kim, and Dongsu Han. 2017. Enhancing Security and Privacy of Tor's Ecosystem by Using Trusted Execution Environments. In *Symposium on Networked Systems Design and Implementation (NSDI).*

[8] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. 2015. Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services. In *USENIX Annual Technical Conference.*

[9] Matthew Prince. [n.d.]. The Trouble with Tor. https://blog.cloudflare.com/the-trouble-with-tor/.

[10] Reporters Without Borders. 2013. Enemies of the Internet 2013, Report. http://surveillance.rsf.org/en/wp-content/uploads/sites/2/2013/03/enemies-of-the-internet_2013.pdf.

[11] Max Schuchard, John Geddes, Christopher Thompson, and Nicholas Hopper. 2012. Routing Around Decoys. In *ACM Conference on Computer and Communications Security (CCS).*

[12] R. Soule, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G.Sirer, and N. Foster. 2014. Merlin:A Language for Provisioning Network Resources. In *ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT).*

[13] Tor: Hidden Service Protocol [n.d.]. Tor: Hidden Service Protocol. https://www.torproject.org/docs/hidden-services.html.en.