Bento: Bringing Network Function Virtualization to Tor

Arushi Arora[1], Christina Garman[1], Michael Reininger[2], Stephen Herwig[2], Nicholas Francino[2], Dave Levin[2]
[1] Purdue University, [2] University of Maryland

1. Summary

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. *Bento* seeks to close this gap by introducing programmable middleboxes into the Tor network, allowing users to install and run sophisticated "functions" on willing Tor routers, further improving anonymity, resilience to attack, performance of hidden services, and more. Also, *Bento* does not require modifications to Tor and can run on the live Tor network. Additionally, we give a list of motivating example functions that can significantly extend the capabilities of Tor to meet users' anonymity needs and nimbly react to new threats.

2. Overview of Bento

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, say, a website fingerprinting attack by correlating traffic patterns with known websites, thereby potentially violating Tor's unlinkability property. Instead in *Bento* Alice can offload processing that would have typically happened on her own machine to another node in the Tor network, as shown in Fig. 1. We next describe each step at a high level.

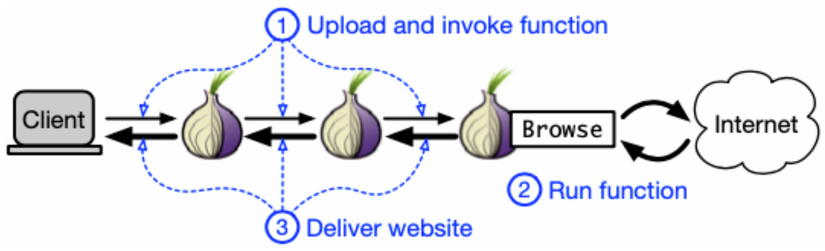

Figure 1: Overview of installing, and executing a `Browser` "function" that runs on another Tor node, downloads a given URL, and delivers it, padded to some threshold number of bytes. Note that, from the perspective of an attacker sniffing the client's link, the client uploads a small amount and then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

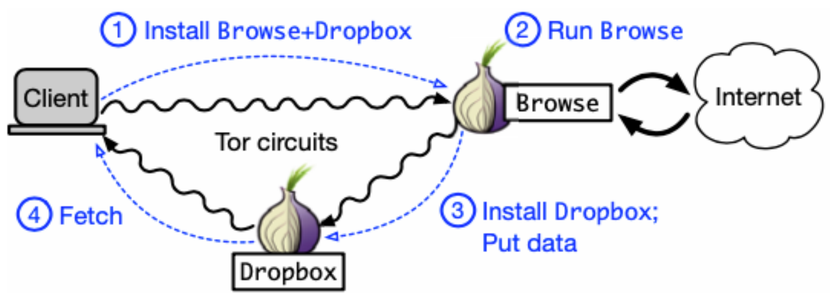

Figure 2: In this motivating example, the user composes two functions: `Browser` which runs a web client to download a website, and `Dropbox` which stores a piece of data to later be fetched.

3. Bento Goals

- Expressiveness
- Protect functions from middlebox nodes
- Protect middlebox nodes from functions
- No Harm to Underlying
- No Extensions to Tor

4. Motivating Examples

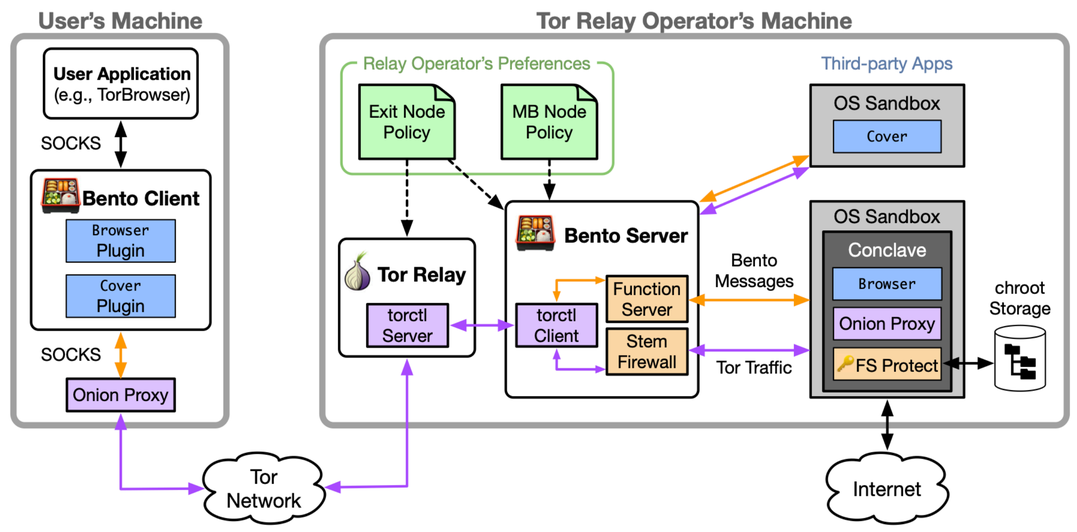

Fig. 3: Design of a Bento middlebox node. Bento sits above an unmodified Tor, and augments it with programmable middleboxes to the Tor network. (Purple arrows denote Tor traffic; orange arrows denote Bento traffic.)

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. 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.

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 that supports two invocations: a "put" of a data file and a subsequent "get" to retrieve it.

These motivating use cases, in Table 1, show that, with just simple programs, a user could significantly extend the capabilities of the Tor anonymity network.

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.

5. Conclusion

In this work, we show programmable anonymity networks are useful, possible, and challenging, yet attainable. 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. We view *Bento* 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.