RETHINKING UPDATES IN ANONYMOUS COMMUNICATION NETWORKS

Jules DEJAEGHERE
PhD student – Faculty of Computer Science, UNamur

March 25th, 2024, Luxembourg

Outline

Overview of Tor

Tor protects you by bouncing your communications around a distributed network of relays run by volunteers all around the world.
Tor Project [5]

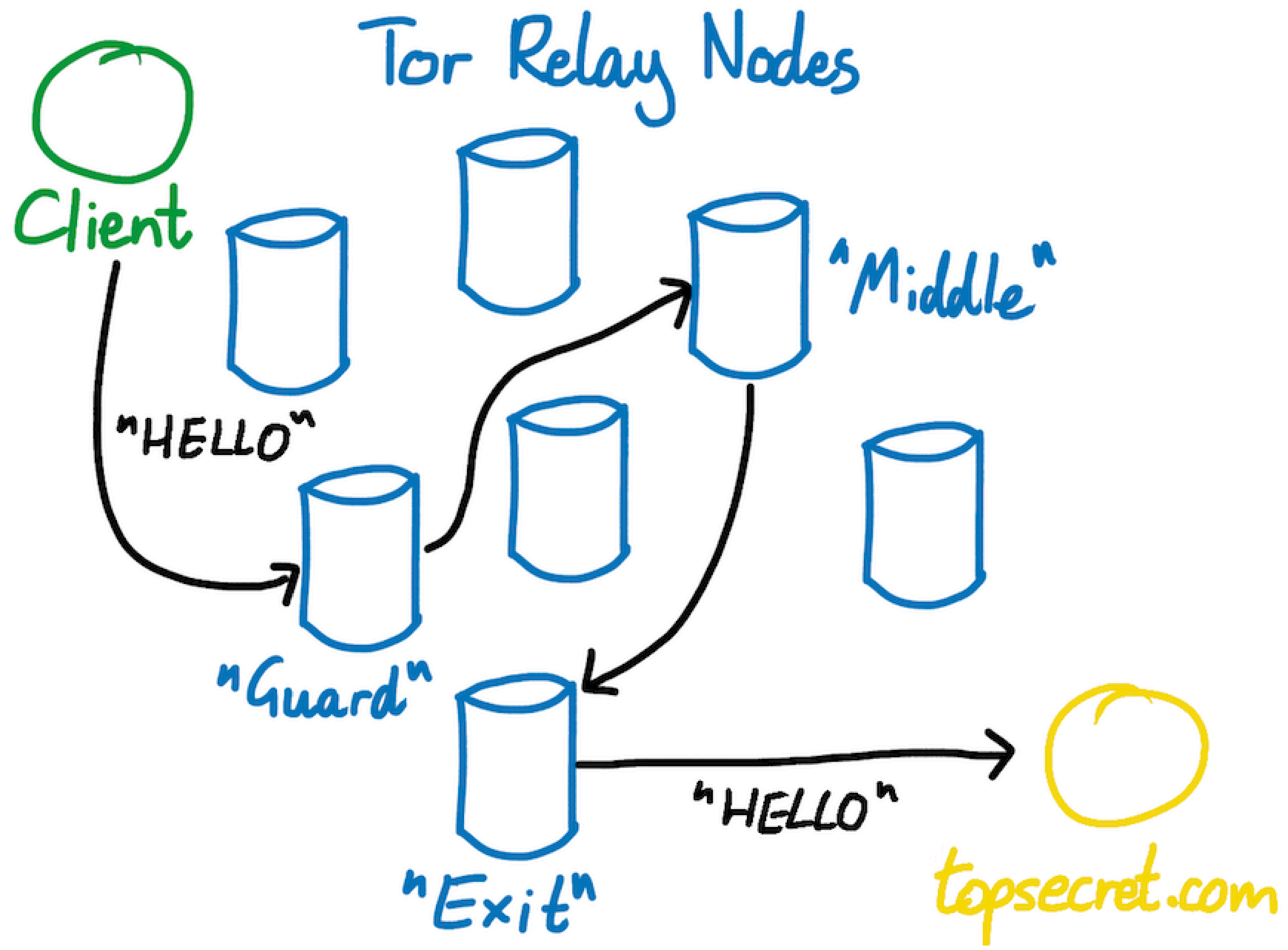
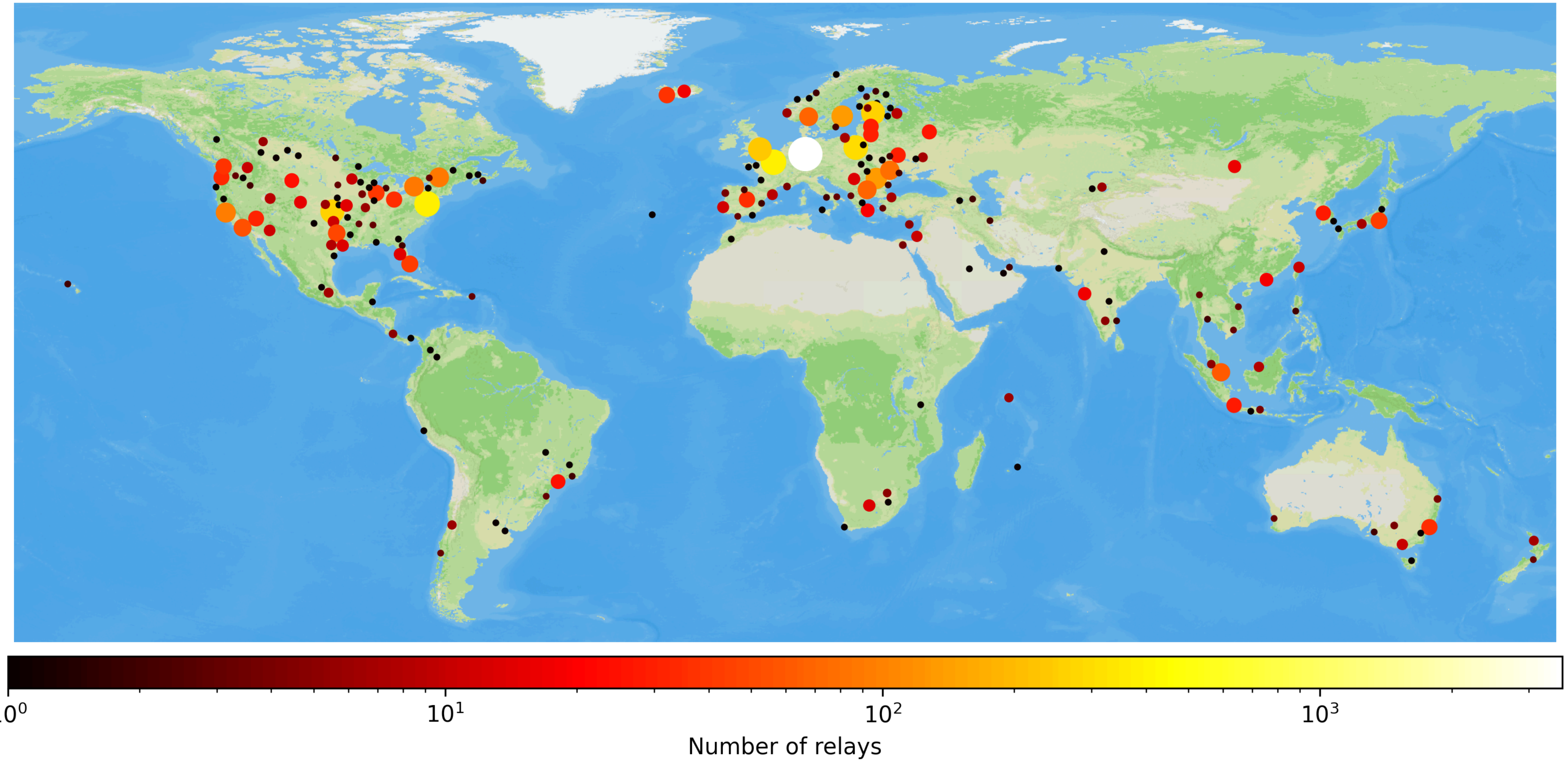


Image : Robert Heaton

Tor relays



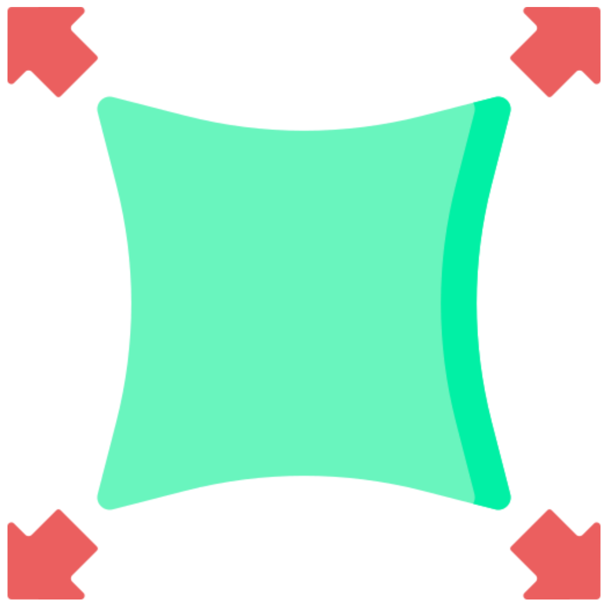
Location of the 7522 Tor relays, as of March 21st 2024 2PM UTC

Why do we care about updates?

Fixing bugs

Fixing security issues

Bringing new features



IT'S 2024

WHY DON'T WE JUST AUTOMATICALLY UPDATE?

For client software — The easy case

Auto-update is already widely used

⇒ Update upon restart

Plugins or addons even allow third-party devs to extend functionality

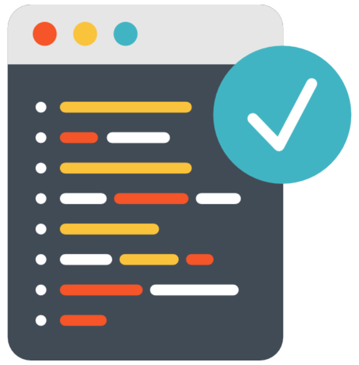
⇒ Many trust models exist

For server software — The tricky case

The software cannot be stopped

What if the update fails?

Need scripts to handle the update



NEW TAKE ON

SOFTWARE UPDATES

Updates are part of normal operation

- Updating should not require external scripts
- Update process should be platform independent
- Updates should happen automatically

Updates are hot swappable

- New code is loaded at runtime
- No need for admin to login

Updates may fail

- The core software can unload failing updates
- Rollback to previous version is automatic

HOW DO WE ACHIEVE THIS?

Two main tools

Rust, for its type and memory safety and for its macros

WebAssembly, a portable binary-code format

Architecture of a typical program

1. The *core software* with a default implementation of all the features
   - Contains hooks (places where updates can be applied)
   - Is the most stable part of the application
   - Embeds a WebAssembly runtime
2. Updates are WebAssembly modules
   - The updates are attached to a hook in the core

Execution of a typical program

1. When the *core software* reaches a hook
    - Check if an update module is available (locally)
    - Yes: execute the module
    - Else: execute the default implementation code
2. When a new update is published by the devs
    - The core fetches the update module (application specific)
    - When the hook is reached the next time, the new module is used

Developer workflow

1. Create the application the usual way
2. Define hooks (where future updates will be applied)
3. Define interface for updates
4. Define a distribution strategy for the updates
5. Write and release update modules

LET'S CREATE A SIMPLE GREETING APPLICATION

Greeting application — Core and hook

```rust
use hooked::hooked;
wasmtime::component::bindgen!("greeting-world" in "wit");

fn main() {
    let b = Person{
        name: "Alice".to_string(),
        age: 5};
    println!("{:?}", say_hello(Some(&b)));
    println!("{:?}", say_hello(None));
}

#[hooked(fn_name = "hello", world_name="greeting-world", binding_struct = "HostState")]
fn say_hello(someone: Option<&Person>) -> String {
    match someone {
        Some(person) => { format!("Hi {}", person.name) }
        None => { "Hello stranger!" }
}}

struct HostState;
impl DemoWorldImports for HostState {
    fn current_user(&mut self) -> wasmtime::Result<String> {
        Ok(String::from("Jules"))
}}
```

Core of the greeting application, compiling to native

Greeting application — Interface

```
package testing: demo;

world greeting-world {
  record person {
    name: string,
    age: u32,
  }

  import current-user: func() -> string;

  export hello: func(who: option<person>) -> string;
}
```

Interface definition for the update module, using WebAssembly Interface Types

Greeting application — Distribution

TBD: this will depend on the application, but we plan on providing functions and macros to ease setup of common use-cases

Greeting application — Update

```rust
wit_bindgen::generate!({
    world: "greeting-world",
    path: "../greetings/wit/greetings.wit"
});
struct Demo;

impl Guest for Demo {
    fn hello(person: Option<Person>) -> String {
        match person {
            Some(person) => { format!("Hello {} yo {}!", person.age, person.name) }
            None => { format!("Hello {}!", current_user()) }
        }
    }
}

export!(Demo);
```

Update module for the `say_hello` function, compiling to WebAssembly

SOME ADDITIONAL CONSIDERATIONS

Is WebAssembly secure?

Memory safety	Sandbox with runtime checks Managed stack Traps
Control flow integrity	Type checking Return address on the managed stack Structured control flow only Jump only at start of constructs
API access	API access provided by the host

Summary of WebAssembly security features from Dejaeghere et al. [1]

Is WebAssembly fast (enough)?

When compared to JavaScript
(De Macedo et al. [2])

- PDF reader app: 19.41% faster than JS
- Game Boy emulator: 15.06% faster than JS

When compared to native code
(Jangda et al. [3])

Using SPEC Benchmark [4]

- 1.55× mean slowdown on Chrome
- 1.45× mean slowdown on Firefox

How to trust the updates?

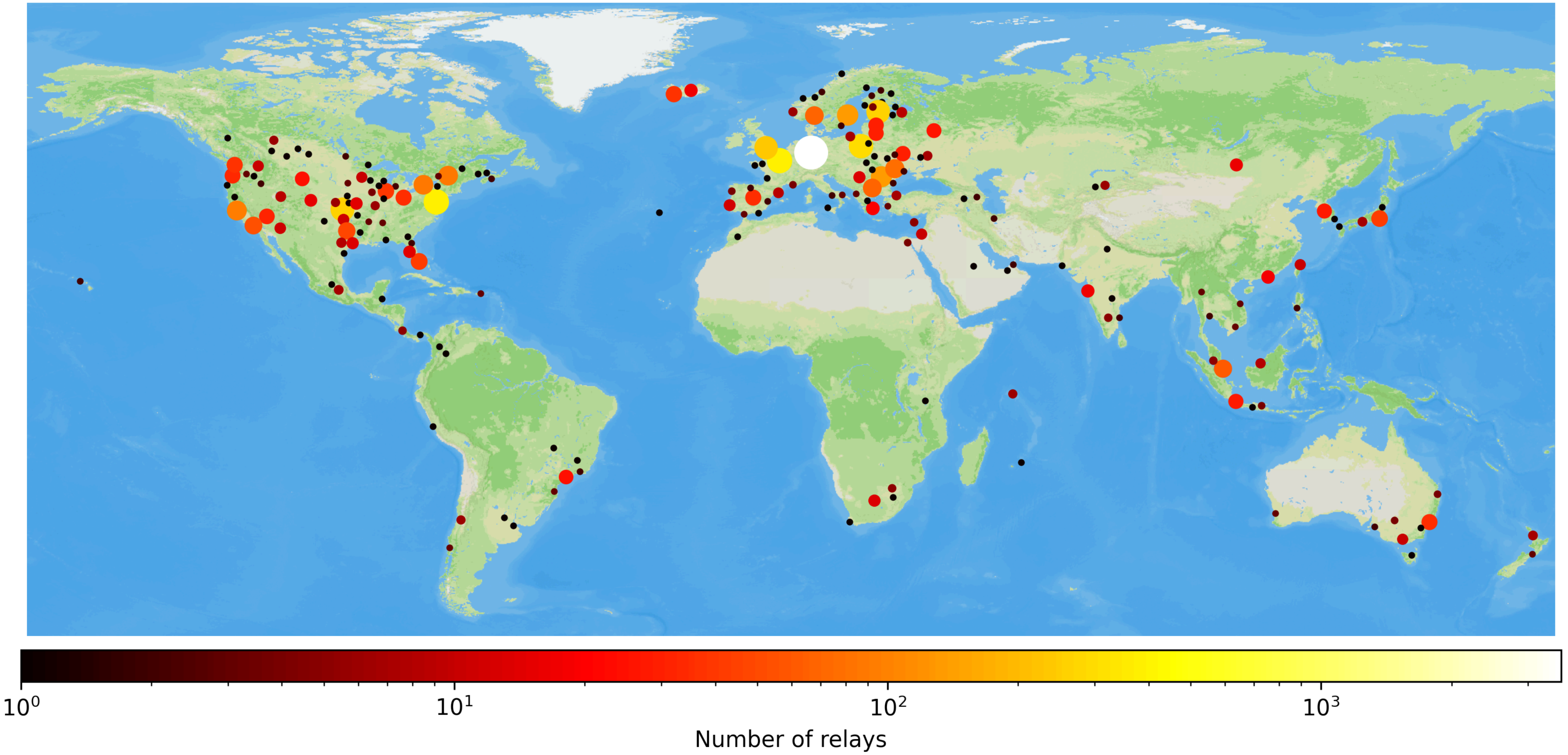
The *core software* can check updates integrity using cryptographic signatures

Trust chain is shorter than usual:

- Usual: developer → package maintainer → users
- Now: developer → users

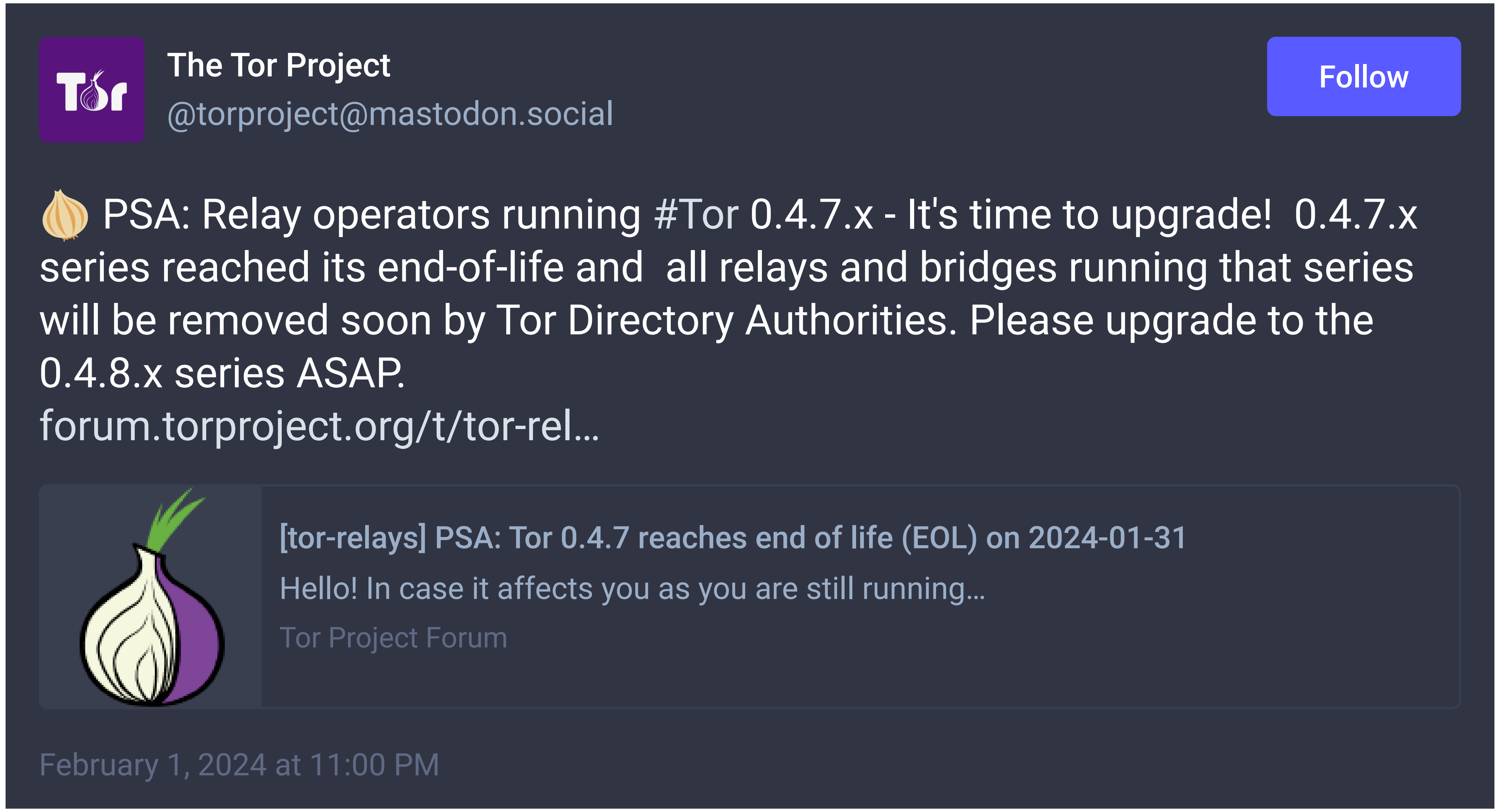
BACK TO TOR

How do we get everyone updated?



Location of the 7522 Tor relays, as of March 21st 2024 2PM UTC

Current strategy to get everyone updated



The Tor Project
@torproject@mastodon.social
Follow

🧅 PSA: Relay operators running #Tor 0.4.7.x - It's time to upgrade! 0.4.7.x series reached its end-of-life and all relays and bridges running that series will be removed soon by Tor Directory Authorities. Please upgrade to the 0.4.8.x series ASAP.
forum.torproject.org/t/tor-rel…

[tor-relays] PSA: Tor 0.4.7 reaches end of life (EOL) on 2024-01-31
Hello! In case it affects you as you are still running…
Tor Project Forum

February 1, 2024 at 11:00 PM

The Tor Project urging relay operators to update before they get excluded from the network

We can probably do better

Updating Tor relays using our framework may

- Get every relay on the latest version
- Enable faster deployment of updates
- Update propagation in a peer-to-peer fashion
- Enable stronger packet policies
- Limit legacy code that developers have to deal with

Applicable beyond Tor

The system has interesting properties for other scenarios : distributed, network-reliant or high-availability applications

RETHINKING UPDATES IN ANONYMOUS COMMUNICATION NETWORKS

Jules DEJAEGHERE

PhD student – Faculty of Computer Science, UNamur

CyberExcellence
By CyberWal

CyberExcellence project, grant 2110186
funded by the SPW Recherche

cyberwal.be
cyberexcellence.be

References

[1] Dejaeghere, J., Gbadamosi, B., Pulls, T. and Rochet, F. 2023. Comparing Security in eBPF and WebAssembly. *Proceedings of the 1st Workshop on eBPF and Kernel Extensions* (New York NY USA, Sep. 2023), 35–41.

[2] De Macedo, J., Abreu, R., Pereira, R. and Saraiva, J. 2022. WebAssembly versus JavaScript: Energy and Runtime Performance. *2022 International Conference on ICT for Sustainability (ICT4S)* (Plovdiv, Bulgaria, Jun. 2022), 24–34.

[3] Jangda, A., Powers, B., Berger, E.D. and Guha, A. 2019. Not So Fast: Analyzing the Performance of WebAssembly vs. Native Code. (2019), 107–120. *2019 USENIX Annual Technical Conference (USENIX ATC 19)* (2019), 107–120.

[4] Standard Performance Evaluation Corporation 2023. SPEC Benchmarks and Tools.

[5] Tor Project About Tor. *Tor Project Support*.

This presentation has been designed using images from Freepik - Flaticon.com.