Pearl

Tor relay implementation in Golang

31 October 2017

Michael McLoughlin
Software Engineer, Uber

Background

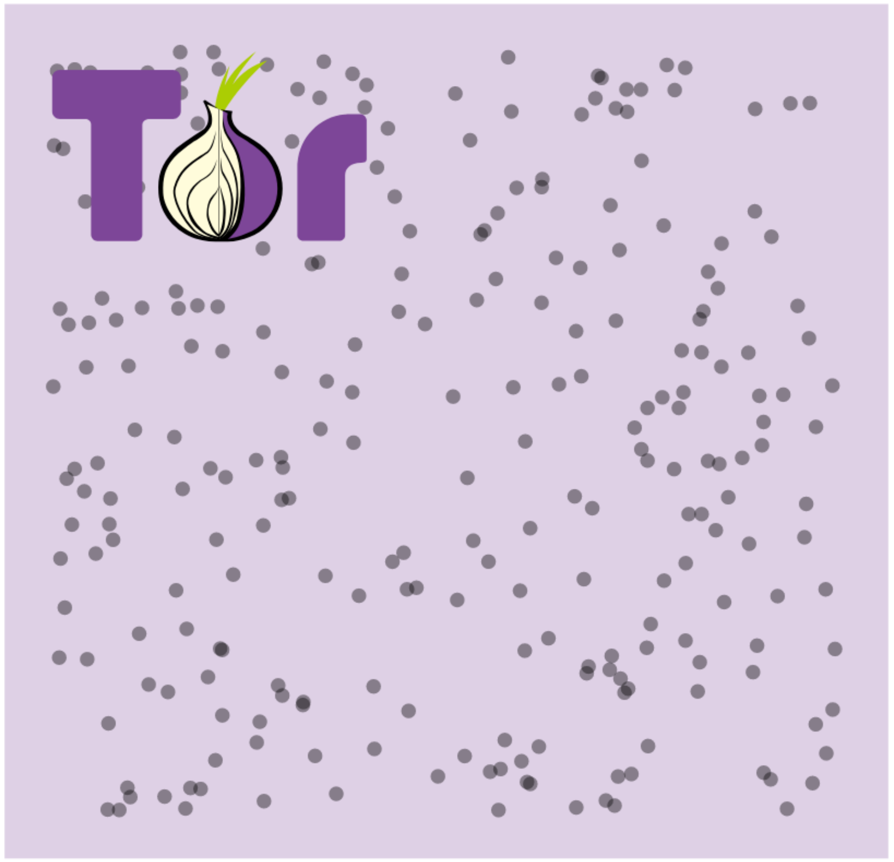
Motivations:

- Learn about the Tor protocol
- Learn by doing!
- Goaded by a glib writeup of a previous Go implementation
- "After a while I realized the language was not suited for the project"

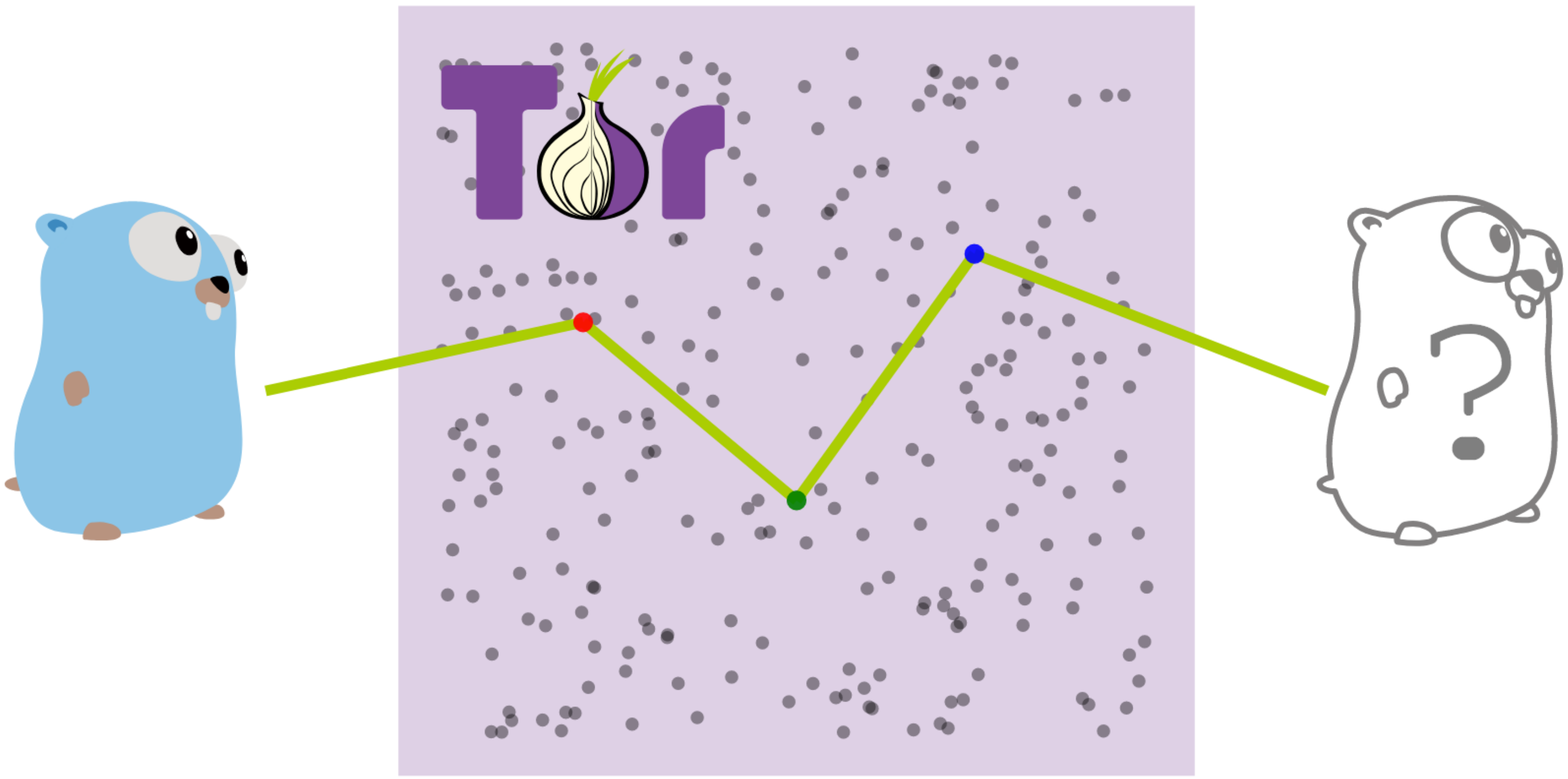
Thus pearl was born out of *hubris* on my part.

- Bradfield Sabbatical Program was a great way to revive my project
- Turns out it's non-trivial

Tor network enables anonymous internet access

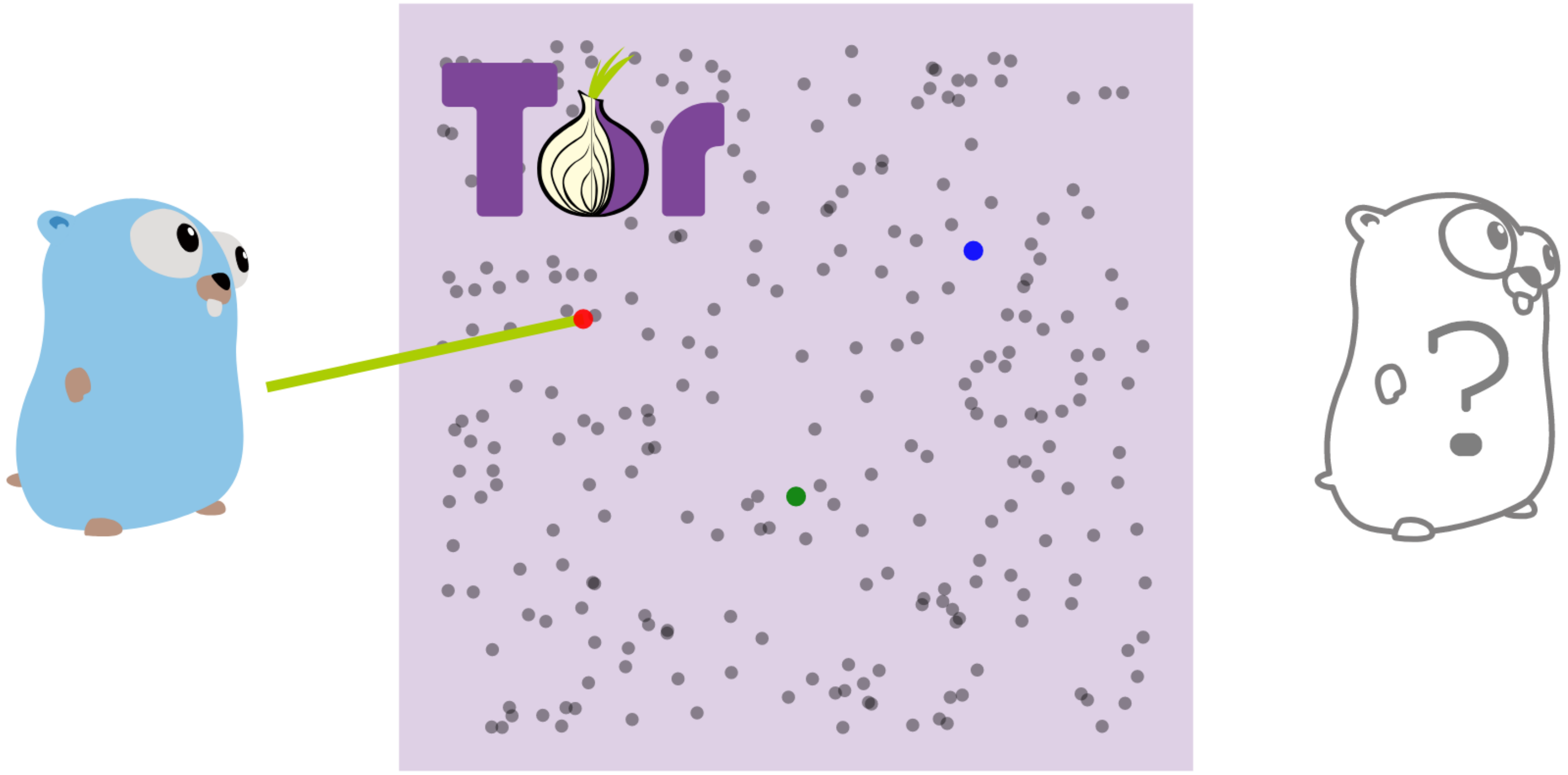


Clients send traffic via a circuit



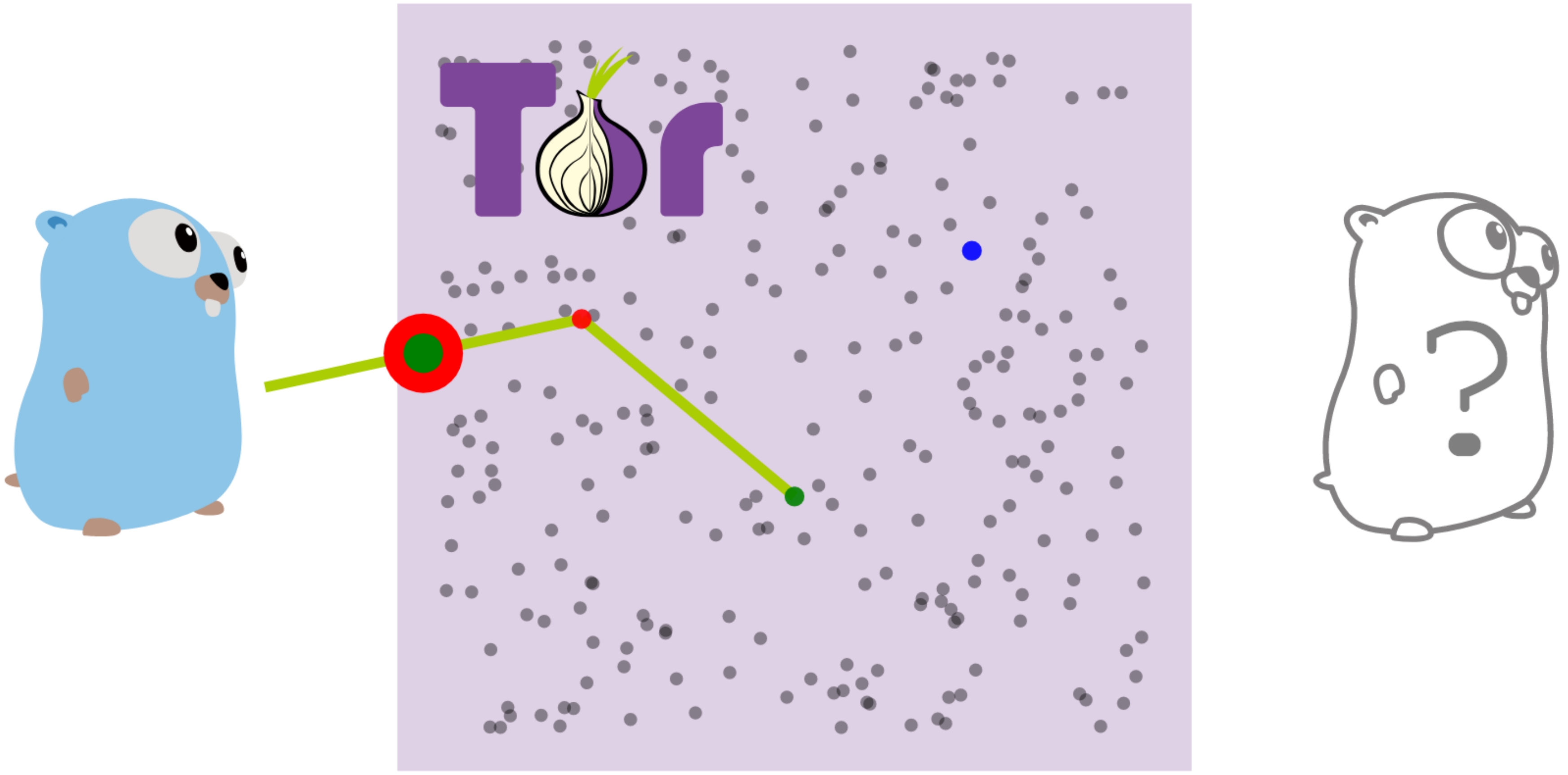
3-hop circuits consist of: **guard** (red), **middle** (green) and **exit**.

Circuit initiated with a CREATE command



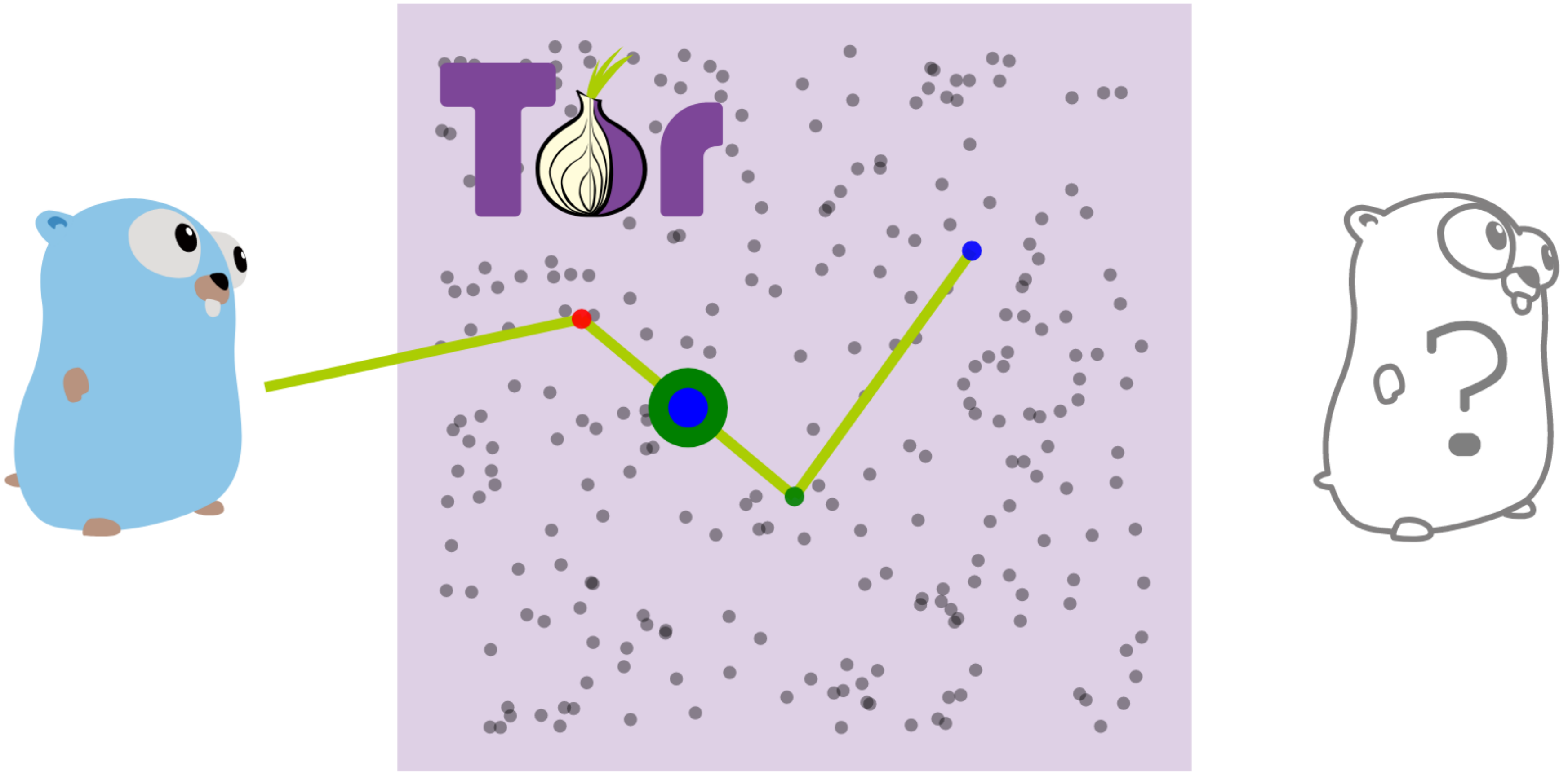
Shared secrets established for symmetric encryption.

Circuit extended with RELAY_EXTEND



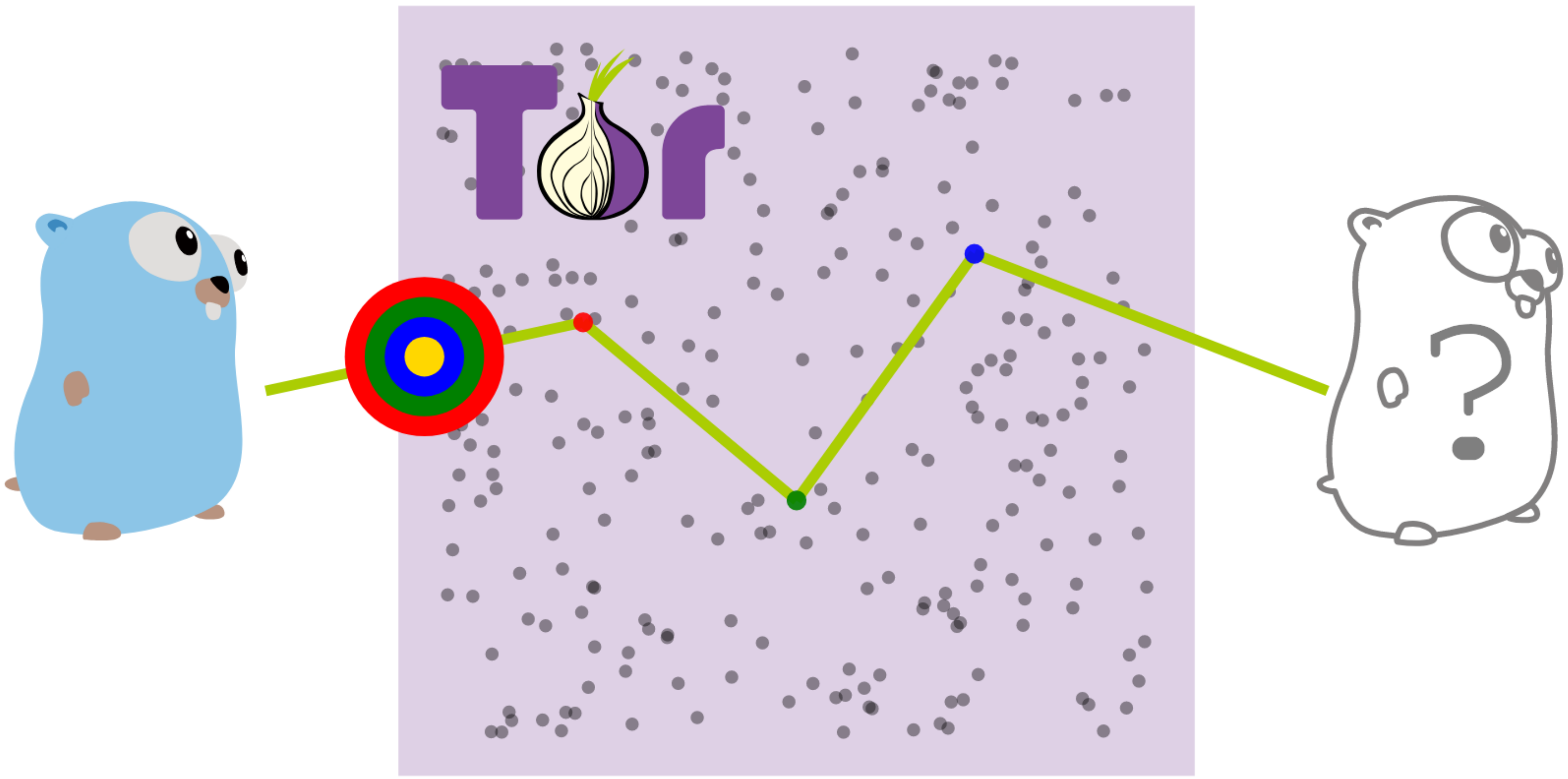
Relay to *green* node without green knowing the gopher's location.

Repeated to complete 3-hop circuit

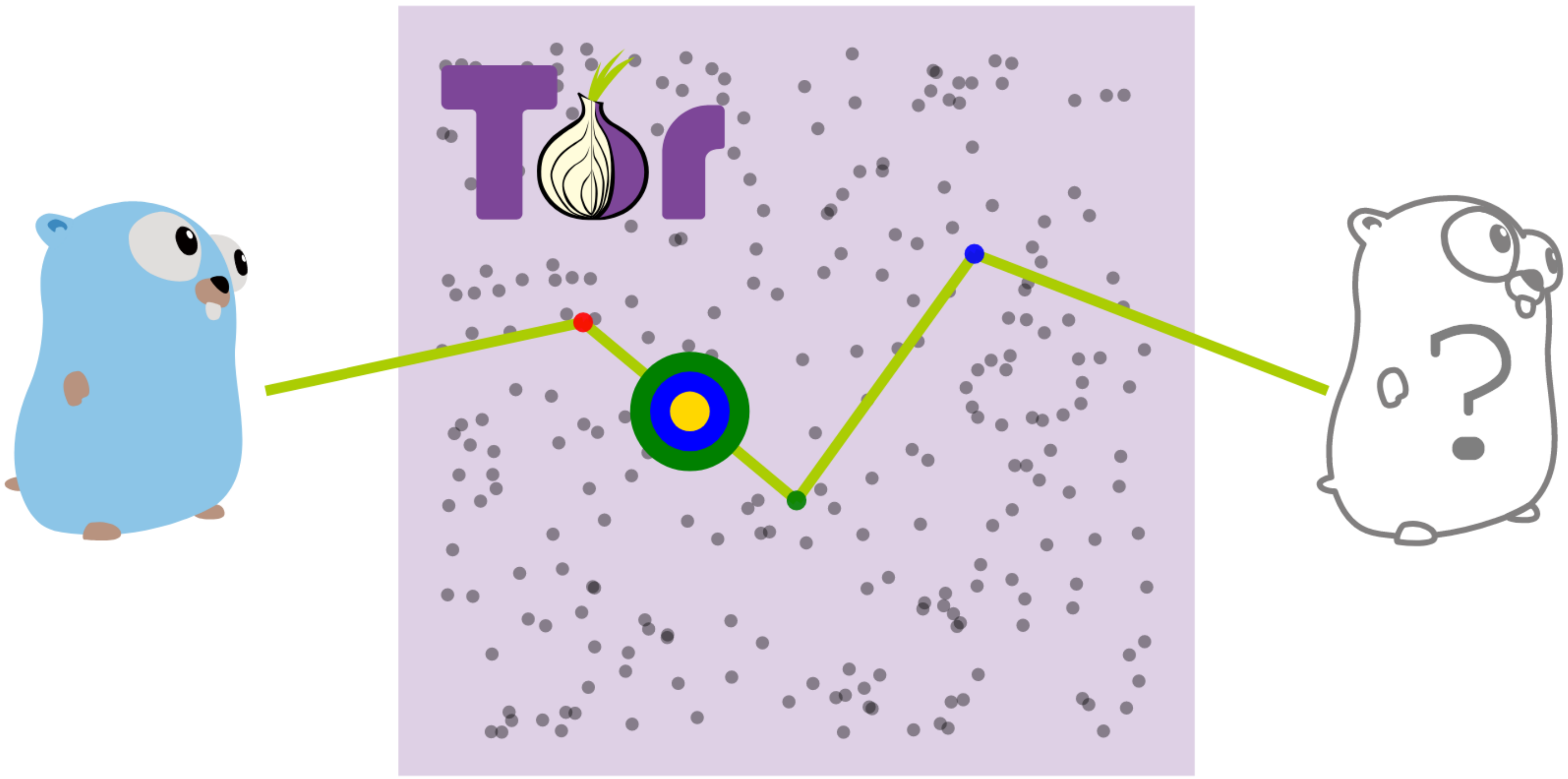


Now streams can be established on the circuit.

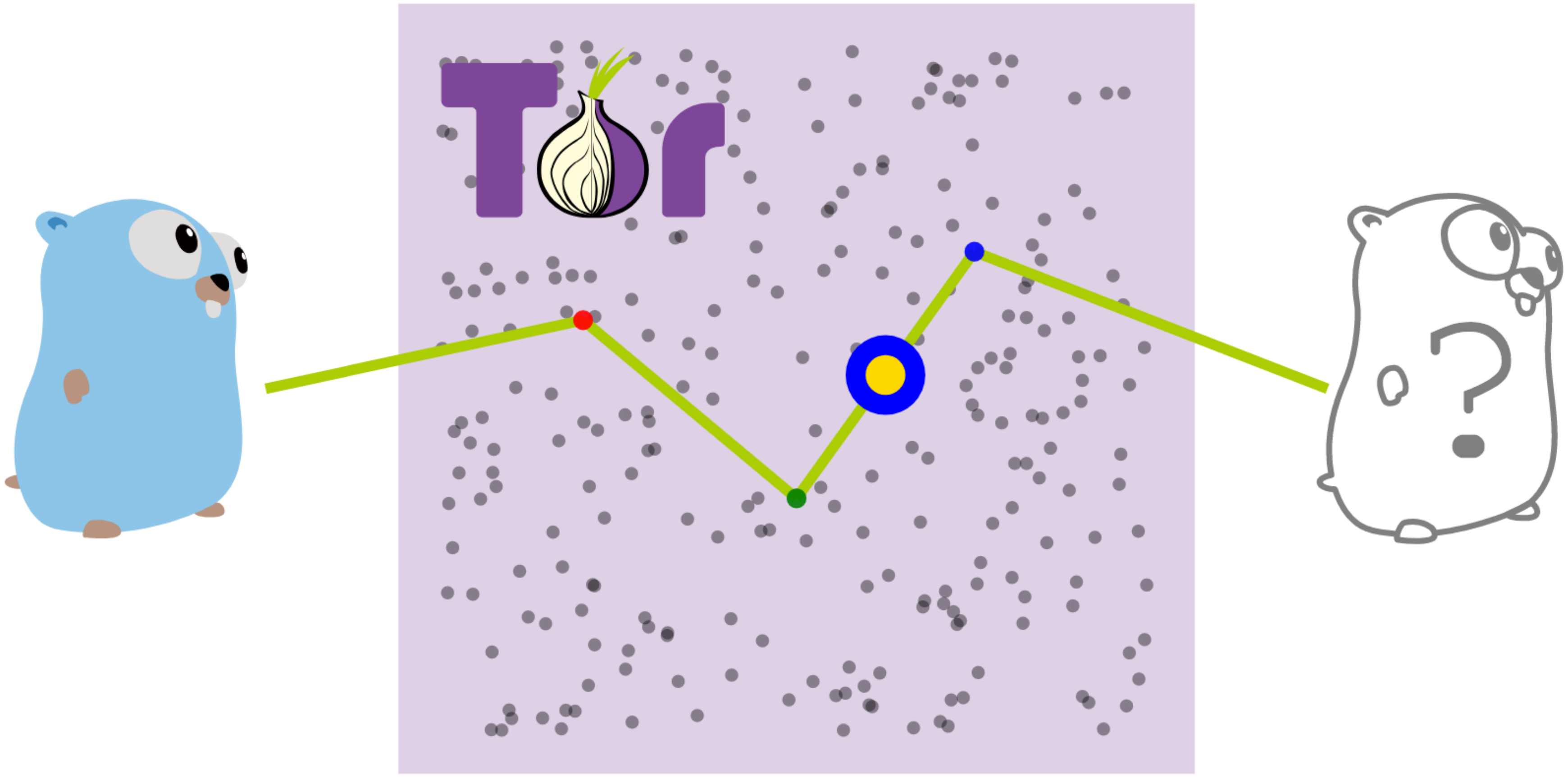
Data can now be relayed



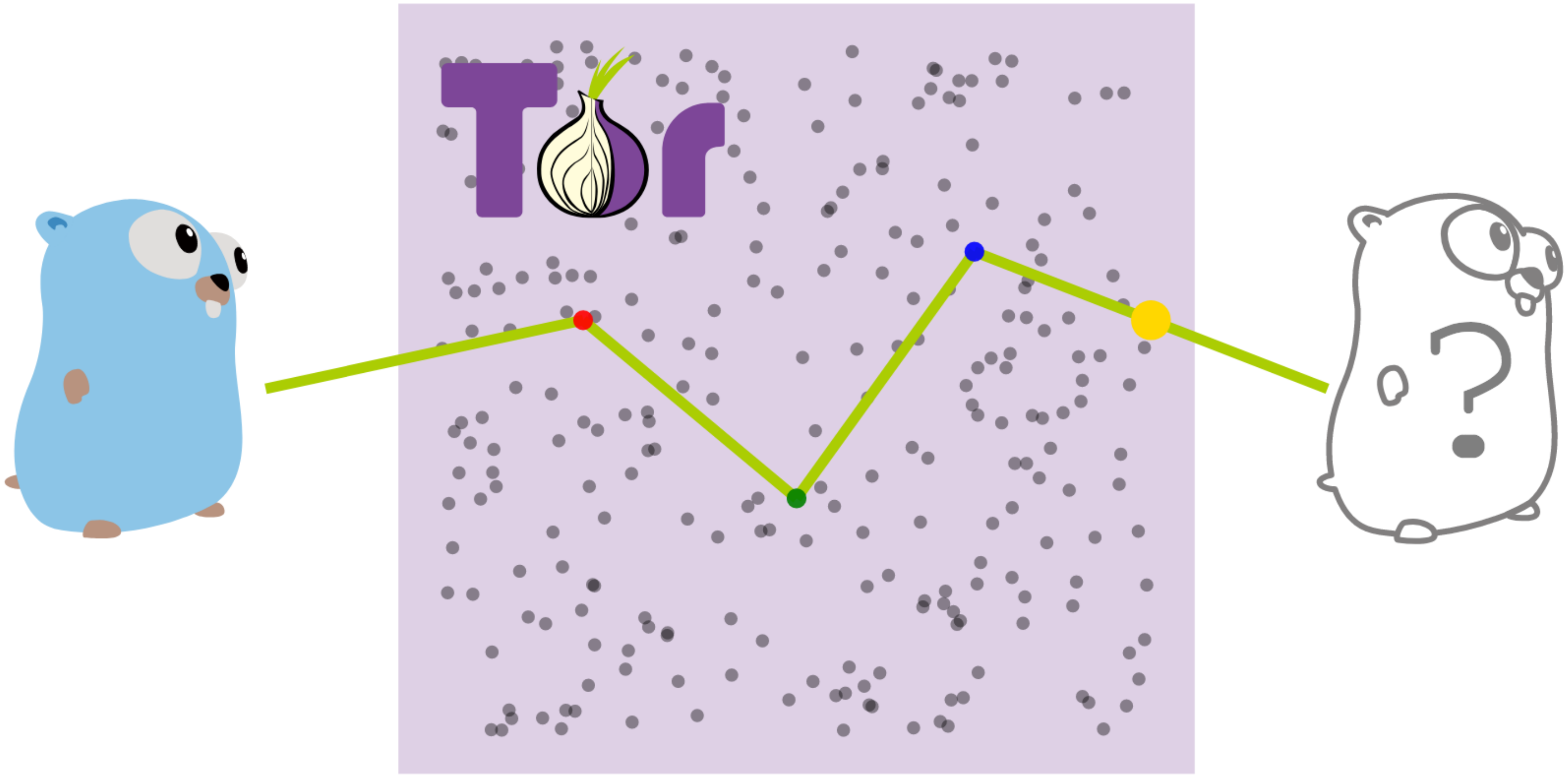
Each node removes its layer of encryption



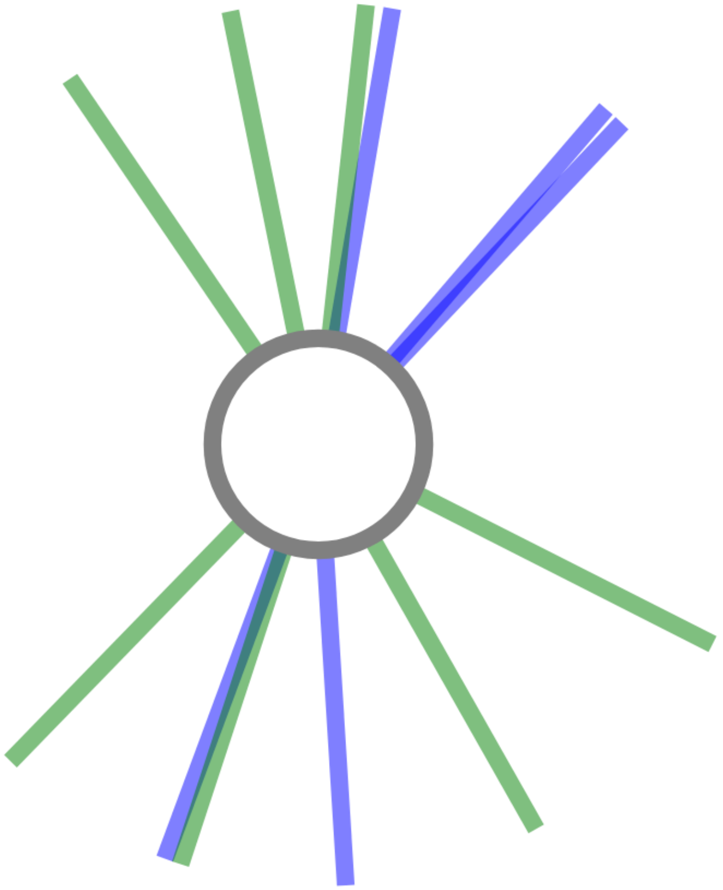
It's like peeling and onion...



Exit node sees the actual packet

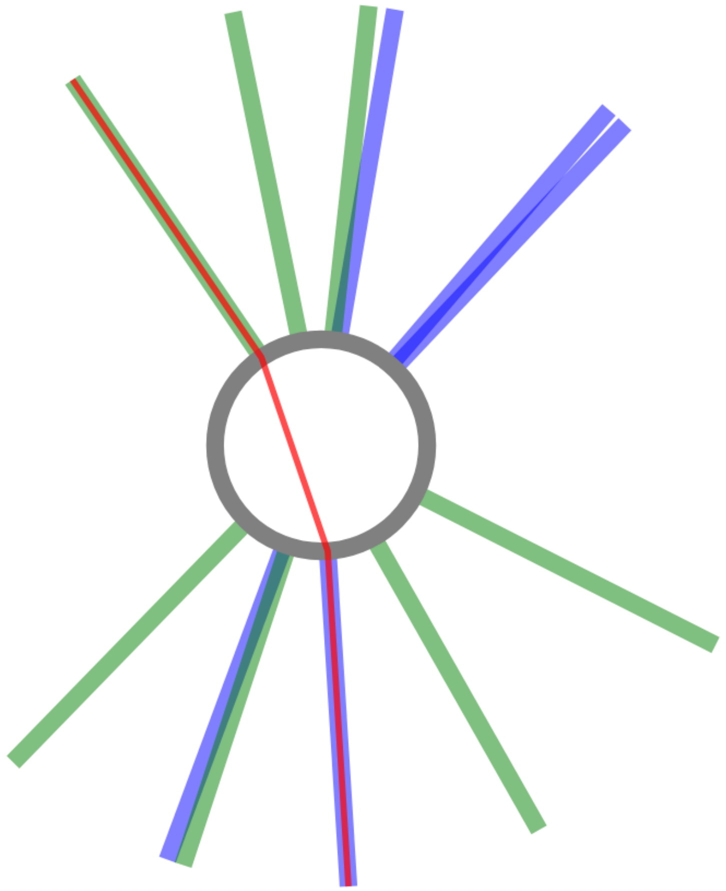


The crux of a relay implementation

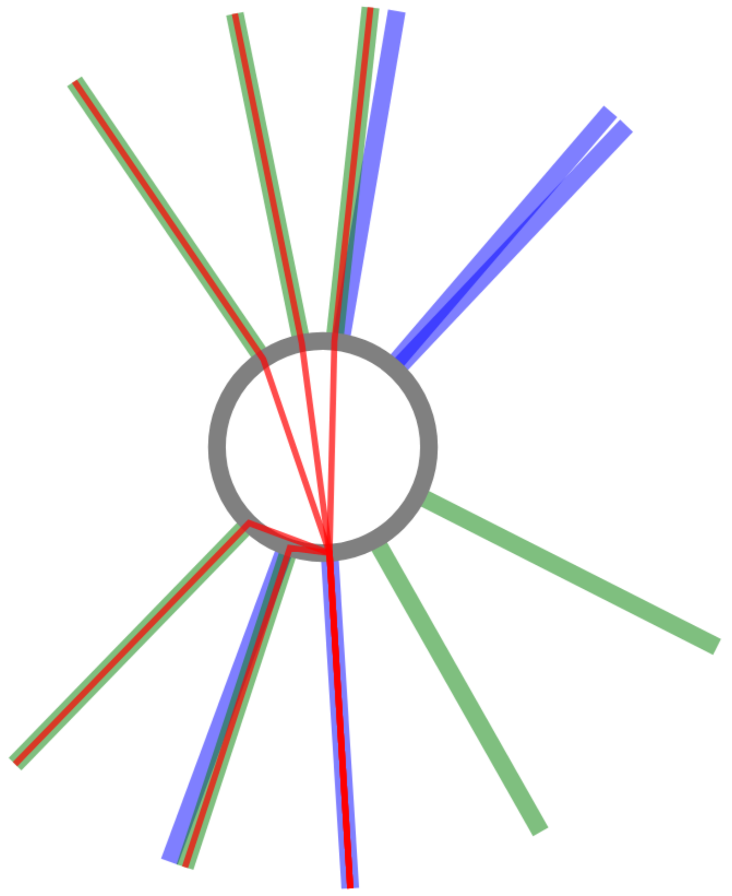


Relay maintains many server and client connections.

Traffic on a circuit is relayed between them



Demux/mux circuit traffic



Relay must demux/mux circuit traffic. Circuits must not interfere with each other.

Why Go?

Strikes a sweet spot

- Fast statically typed compiled language
- Feels like writing in a dynamic language
- Strong tooling, standard libary and ecosystem

Concurrency features:

- **Goroutines:** "green threads" mapped to OS threads, running ~100k is reasonable
- **Channels:** send and receive operations between goroutines

Gained a reputation for:

- Systems programming
- High performance servers

Why not?

Tom van der Woerdt had harsh criticisms:

- Limited cipher suites and poor performance of `crypto/tls`
- Hence forced to use C bindings to `openssl`
- Blamed `cgo` for buildup of OS threads and excessive locking
- Memory usage per connection: 16KB buffer plus 4KB goroutine stack
- Buffered channel: static array implementation causes high memory usage

Maybe we can avoid these problems now:

- `crypto/tls` has massively improved
- Memory problems result from *design decisions*

Challenges

Grunt work:

- Forking `crypto/tls` standard library package
- Parsing protocol data formats
- Navigating a *work-in-progress* spec

Cryptographic details:

- Auth flow
- `NTor` handshake
- Old TAP handshake
- Some hand-rolled algorithms

Concurrency is hard

State managed by an associated goroutine

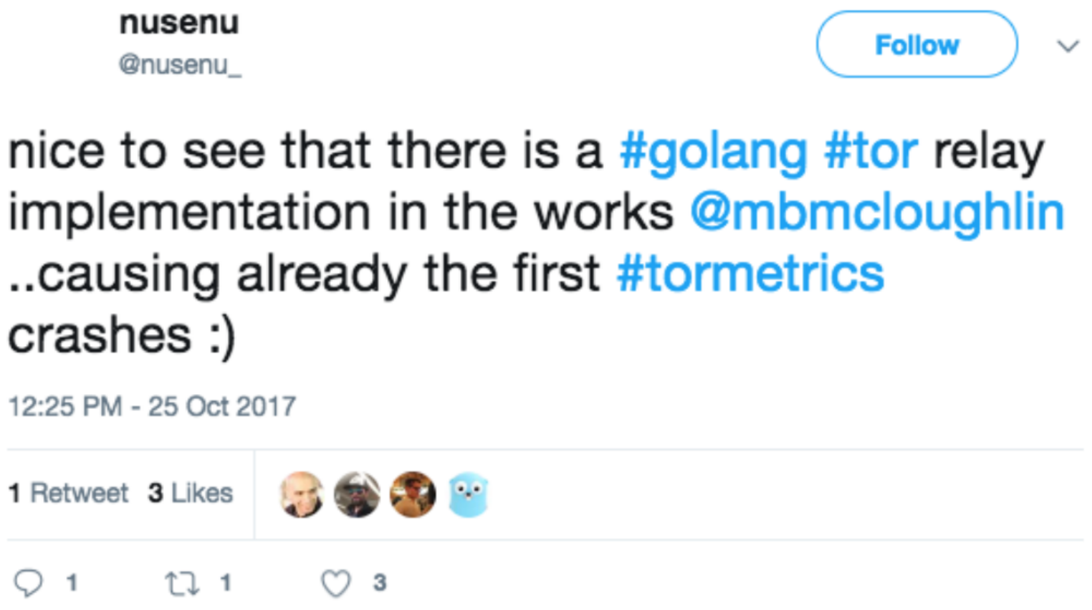
- `Connection`
- `TransverseCircuit`

Connection offloads to circuit via channel:

```
cell, _ := c.ReceiveCell()
switch cell.Command() {
case CommandCreate, CommandCreate2:
    CreateHandler(c, cell)
case CommandCreated, CommandCreated2, CommandRelay, CommandRelayEarly, CommandDestroy:
    s, ok := c.circuits.Sender(cell.CircID())
    if !ok {
        bad()
    }
    s.SendCell(cell)
case CommandPadding, CommandVpadding:
    logger.Debug("skipping padding cell")
default:
    logger.Error("no handler registered")
}
```

Current Status

It's gone viral


nusenu
@nusenu_

Follow

nice to see that there is a #golang #tor relay implementation in the works @mbmcloughlin ..causing already the first #tormetrics crashes :)

12:25 PM - 25 Oct 2017

1 Retweet 3 Likes

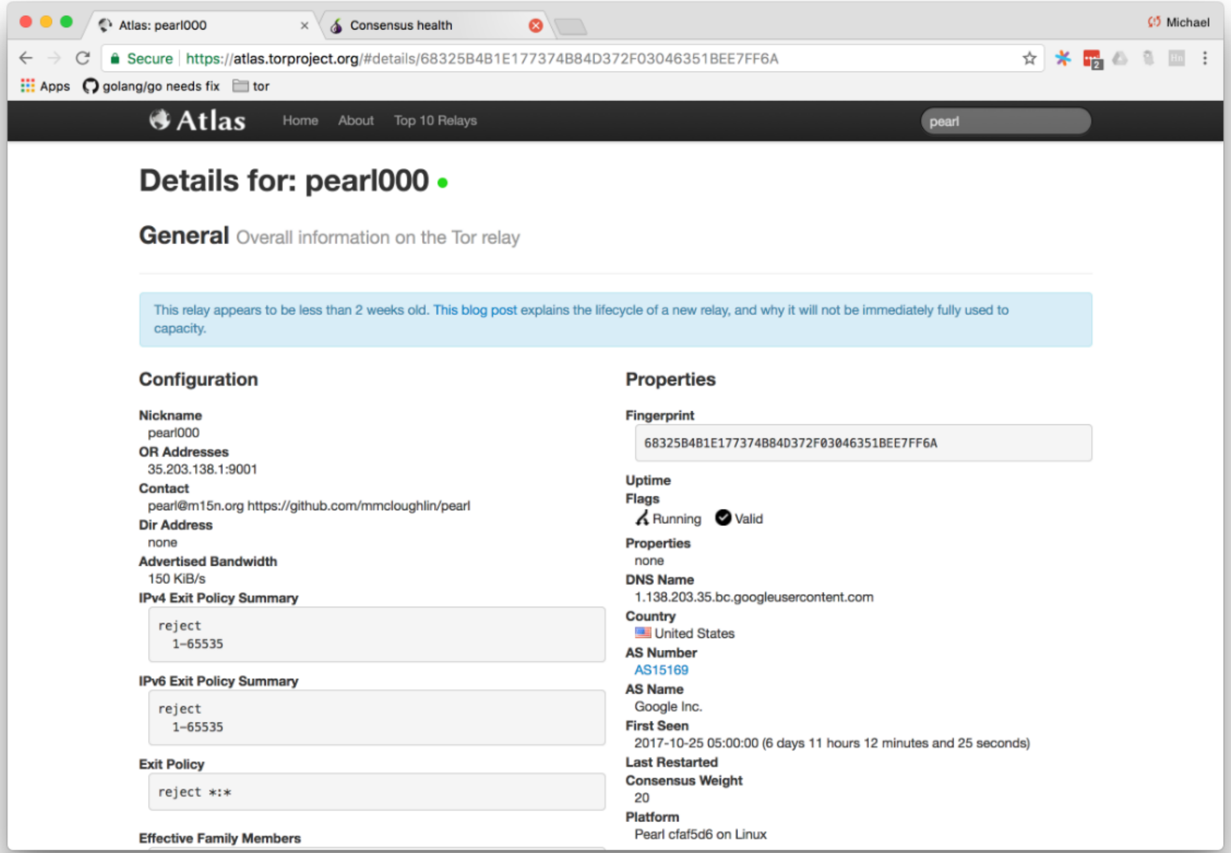
1 1 3

(Not quite. I did crash the tor metrics system though.)

Accepted into consensus

```
$ wget http://86.59.21.38/tor/server/all
$ cat all
router pearl000 35.203.138.1 9001 0 0
signing-key
-----BEGIN RSA PUBLIC KEY-----
MIGJAoGBAKzTaN4tZGv1kiQWBzeuOk+ovr2LtIURlaVC38j6j/fQuYfuAZX/XvV1
fQr9EVh+T617dh+frt2D0QDuzLUvP3hpgVozW94w+Ib85pUCne03f4rj3QYu5Qtg
GvzShslZI6vgyy0g2jAOGa4jxT/UYAcKE5dQo8CBKA6Qb0P5Joc1AgMBAAE=
-----END RSA PUBLIC KEY-----
fingerprint 6832 5B4B 1E17 7374 B84D 372F 0304 6351 BEE7 FF6A
...
platform Pearl cfaf5d6 on Linux
contact pearl@m15n.org https://github.com/mmcloughlin/pearl
...
```

Pearl node running in the Tor network



Details for: pearl000 ●

General Overall information on the Tor relay

This relay appears to be less than 2 weeks old. This blog post explains the lifecycle of a new relay, and why it will not be immediately fully used to capacity.

Configuration

Nickname
pearl000
OR Addresses
35.203.138.1:9001
Contact
pearl@m15n.org https://github.com/mmcloughlin/pearl
Dir Address
none
Advertised Bandwidth
150 KiB/s
IPv4 Exit Policy Summary

```
reject
  1-65535
```

IPv6 Exit Policy Summary

```
reject
  1-65535
```

Exit Policy

```
reject *:*
```

Effective Family Members

Properties

Fingerprint

```
68325B4B1E177374B84D372F03046351BEE7FF6A
```

Uptime
Flags
Running ✓ Valid
Properties
none
DNS Name
1.138.203.35.bc.googleusercontent.com
Country
United States
AS Number
AS15169
AS Name
Google Inc.
First Seen
2017-10-25 05:00:00 (6 days 11 hours 12 minutes and 25 seconds)
Last Restarted
Consensus Weight
20
Platform
Pearl cfaf5d6 on Linux

Future Work

There is *a long way* to go.

Near term:

- Nurse **production** deployment
- Develop more realistic local **testing**
- Focus on **performance**
- Revisit **parser**

Longer term expand protocol support:

- Exit node, client side, hidden services...

Alternatively:

- Contribute to new go-tor project or the official tor project

Acknowledgements

- **Myles** for our daily check-in and keeping me on the straight and narrow
- **Oz** and **Myles** for setting up the program
- **Bradfield Sabbatical Program** for the community and emotional suport
- **Uber** for letting me participate

Thank you

Michael McLoughlin
Software Engineer, Uber
mmcloughlin@gmail.com
http://mmcloughlin.com
@mbmcloughlin