Adaptive Resilience: A Three-Pillar Approach to Operational Robustness

November 3, 2025 - IETF 124

Vinicius Fortuna

Network Disruption

- **Identification**: Adversaries pinpoint specific traffic.
- **Interference**: They block or manipulate that traffic.

A Survey of Worldwide Censorship Techniques - RFC 9505

Network disruption

An adversary must differentiate traffic they like, from traffic they don't like or cause collateral damage

Identification + Interference

Network disruption

Identification + Interference

Read

Drop

Inject

Privacy violation

Resilience violation

Three Pillars

01	**Identification**	Make traffic difficult for adversaries to identify
02	**Interference**	Employ robust strategies to evade disruption
03	**Agility**	Build with composable, extensible APIs to rapidly evolve our protocols

Pillar 1 - Prevent Identification

Assume the network is compromised. Think of it as a "Zero Trust" model for everyone. Make traffic opaque

- **Encrypted DNS**
- **Encrypted ClientHello** (ECH)

Important: the **endpoints are still in control**. Parental controls and malware protection are still possible

We still need good solutions for hiding IP addresses

Pillar 2 - Mitigate Interference

More robust and smarter strategy selection. Consider all the variables:

- Protocols
  - Parameters
- Endpoints
  - Addresses
- Paths/Interfaces

We need **Happy Eyeballs on steroids**.

Related work: Happy Eyeballs v3, SVCB RR, Alt-Svc, TAPS

Pillar 2 - Mitigate Interference - dynamic selection

Passive health monitoring

Intra app (`getintra.org`):

- Try TLS connection. If reset or timed out, try again with new strategy (e.g. TCP split, TLS Record Fragmentation). Ideally this would be on the application layer

Active Health Monitoring (probes)

Outline SDK Smart Dialer:

- Find DNS service, Find TLS strategy for a domain, combine them.

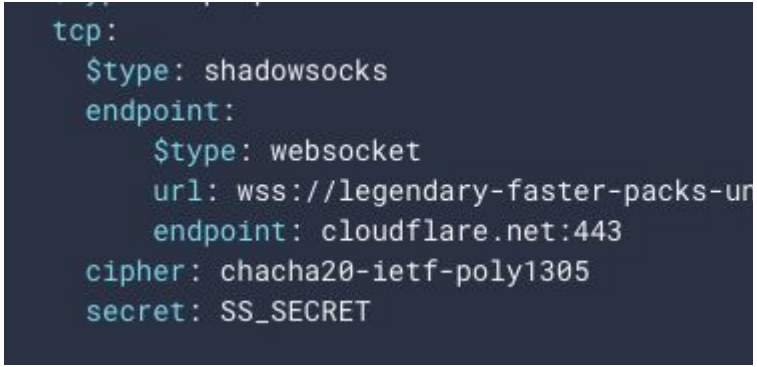
Pillar 3 - Agility

Networking libraries should provide

- **Common-language/API** (domain resolver, stream dialer, packet dialer, HTTP client, …)
- **Cross-platform Building Blocks** (implementations)
- **Hooks/extension points** (inject dialers, extensions, parsers, …)
- **Composition** (e.g. DNS over CONNECT-UDP over h3)

Pillar 3 - Agility - Outline Config Composition Example

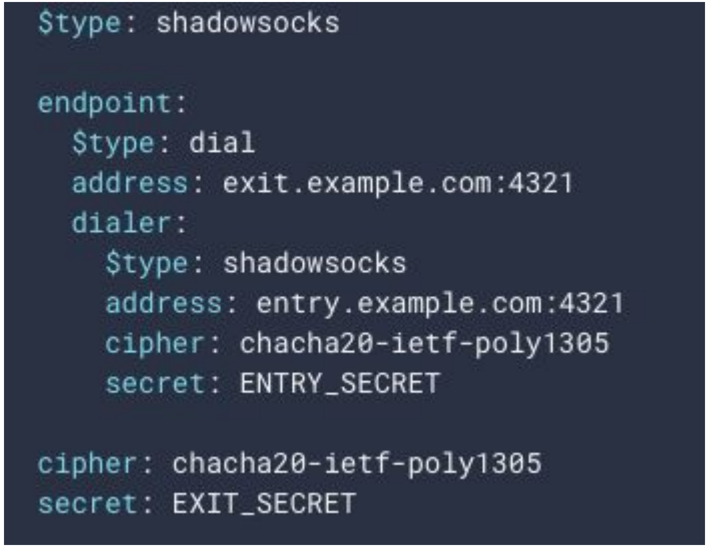
Shadowsocks-over-Websocket:



```
  tcp:
    $type: shadowsocks
    endpoint:
        $type: websocket
        url: wss://legendary-faster-packs-un
        endpoint: cloudflare.net:443
    cipher: chacha20-ietf-poly1305
    secret: SS_SECRET
```

"Strategy" selection is a strategy too (first, round-robin, least loaded…)

2-hop Shadowsocks



```
$type: shadowsocks

endpoint:
  $type: dial
  address: exit.example.com:4321
  dialer:
    $type: shadowsocks
    address: entry.example.com:4321
    cipher: chacha20-ietf-poly1305
    secret: ENTRY_SECRET

cipher: chacha20-ietf-poly1305
secret: EXIT_SECRET
```

Pillar 3 - Agility - What if...

```
myserver.example.org.   300   IN HTTPS 1  .
alpn="h3,h2,h3|masque(e=cdn.example.com)"
```

```
myserver.example.org.   300   IN HTTPS 1  .
alpn="h3(tls=(sni=front.example.org)),h2"
```

Pillar 3 - Agility - Missing Building Blocks

- **Cross-platform C-API for the system resolver.**
  - Significant blocker for HTTPS/SVCB RR, ECH
- **C-APIs for dialers/endpoints/connections**
  - Allows for composing transports across languages.
- **Extensible DNS APIs**
  - Let me inject parsing of HTTPS/SVCB RR, and SVCB Params not already supported by the language.
- **More granular and extensible TLS APIs**
  - Let me inject extensions so I can implement ECH/GREASE etc in my application code
  - Let me use just the record layer. Or just the messaging layer.

Goal: implement and use new protocols, before core libraries and OSes support them

Thank you

www.vinifortuna.com

fortuna@google.com