DNSDist:

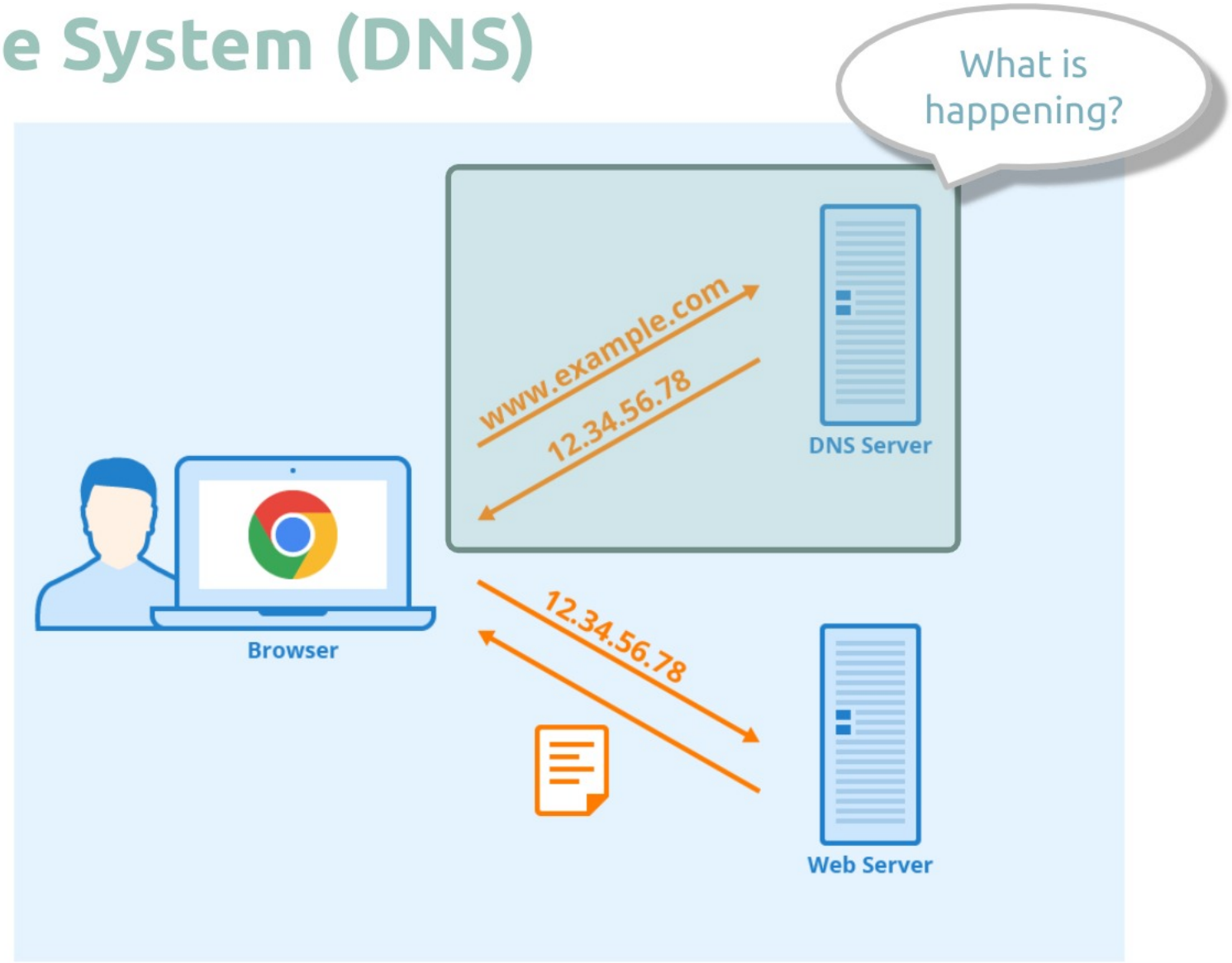
The power of DNS Load-balancing including DDoS mitigation

Bahram Bahrambeigy
19.05.2020

Agenda

- DNS and its recursive process
- DNS Load-balancing
- DNS security attacks and concerns
- Introducing DnsDist
- Caching, Packet Policies, Rules
- Security features: DoH, DoT, DNSCrypt
- Kernel features: eBPF, SO_REUSEPORT

Domain Name System (DNS)



DNS recursive process

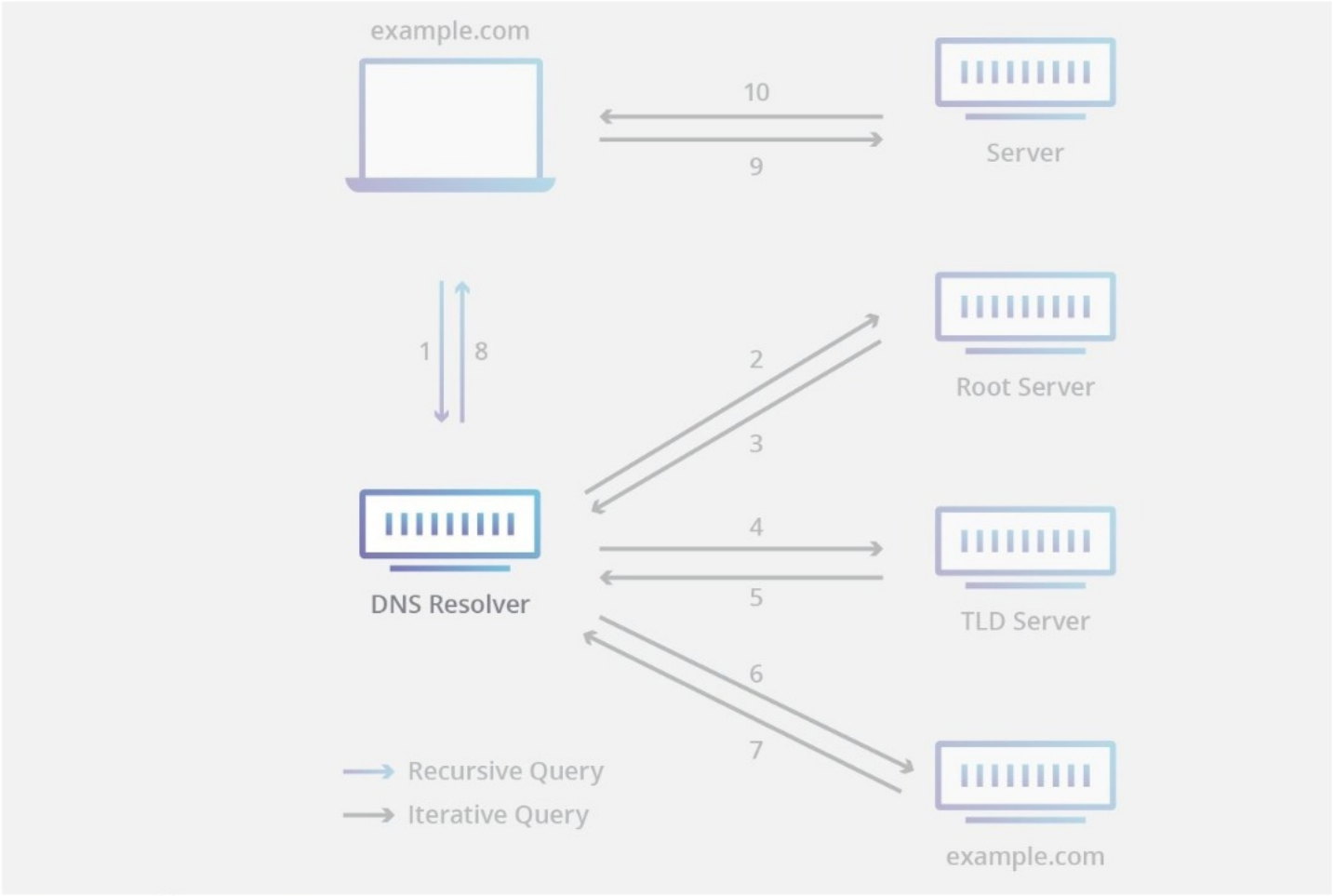
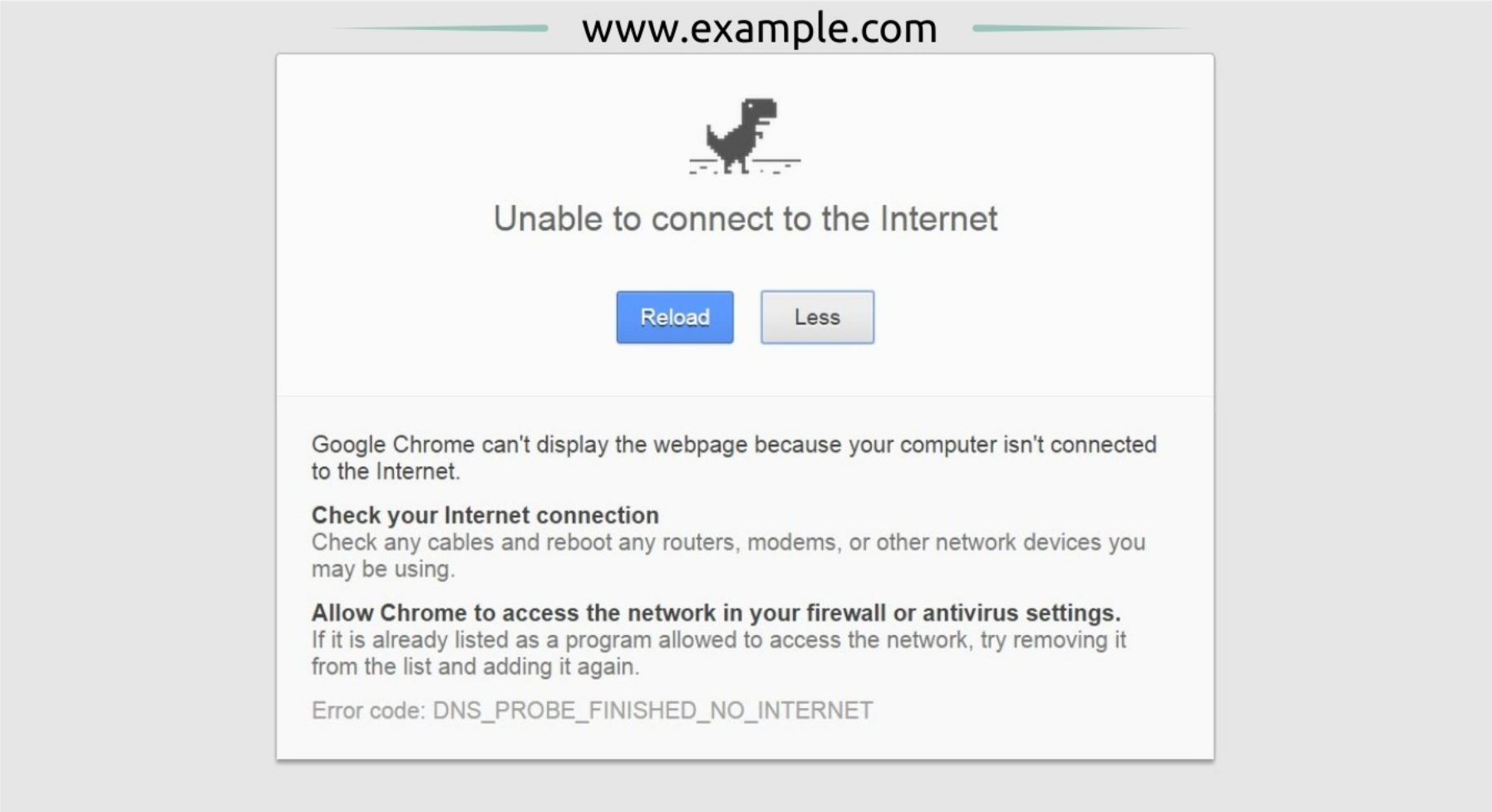


Image: Cloudflare.com

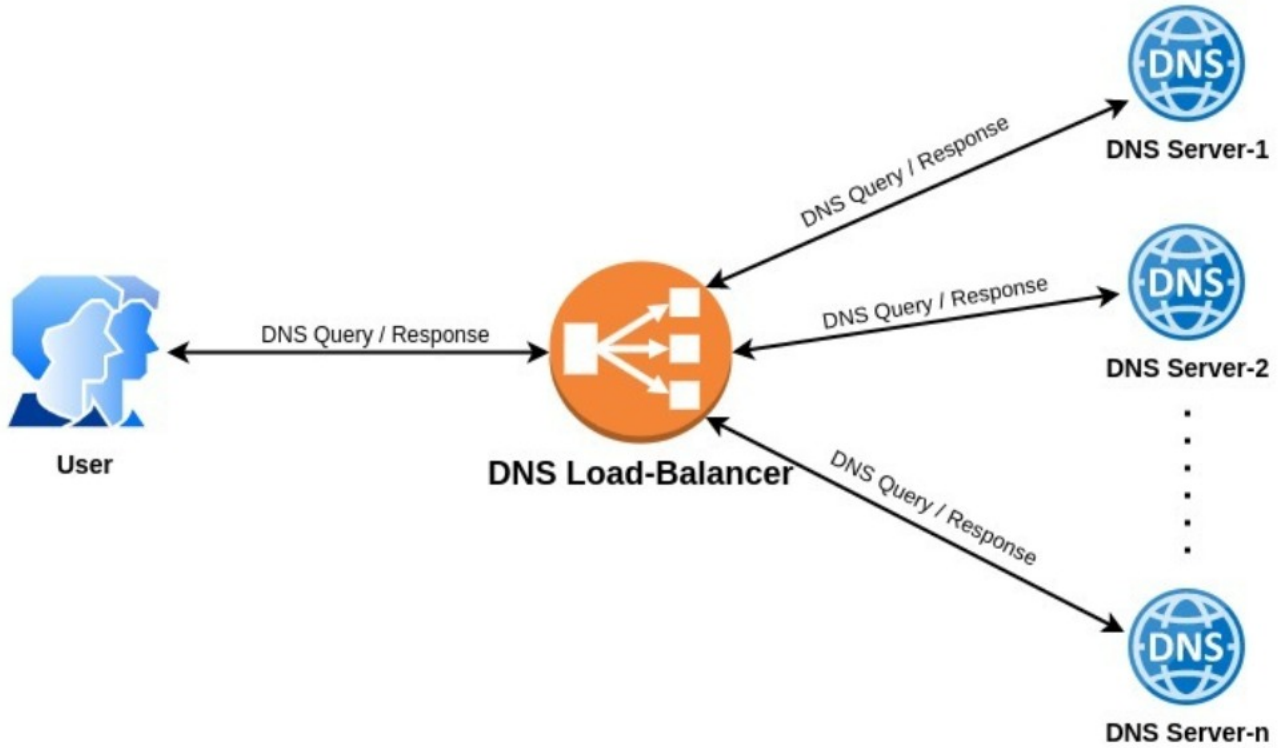
1. Query the Resolver (`example.com`) ?
2. Query the Root Servers (`.com`) ?
3. Root Response: "`a.gtld-servers.net`"
4. Query the TLD Servers (`example.com`) ?
5. TLD Response: "`ns1.example.com`"
6. Query the NS Server (`example.com "A"`) ?
7. NS Response: "`1.2.3.4`"
8. Resolver Response: "`1.2.3.4`"
9. Client Web Request to: "`1.2.3.4`"
10. Response to Client from: "`1.2.3.4`"

What if DNS goes down ?



Necessity of DNS Load-balancing

- Distributing the load
- Various Policies
- Cache sharing
- Security protocols
  (DoH, DoT, DNSCrypt, etc)
- Extra layer of protection



Each client can produce ~ 200k queries

Top DNS Attacks/Vulnerabilities

- DNS Amplification attack
- DNS Reflection attack
- NX Domain attack
- Phantom Domain attack
- Lock-up Domain attack
- DNS Cache poisoning
- DNS Hijack
- DNS Tunneling

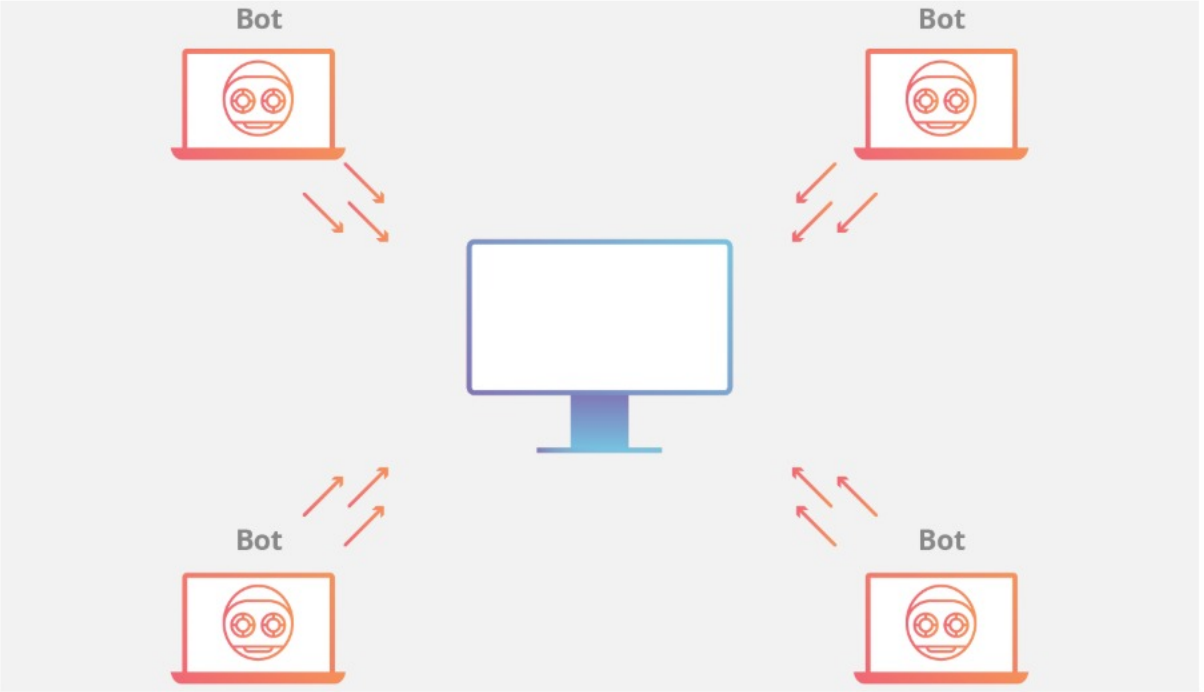


Image: Cloudflare.com

DNS Attacks - Reflection and Amplification

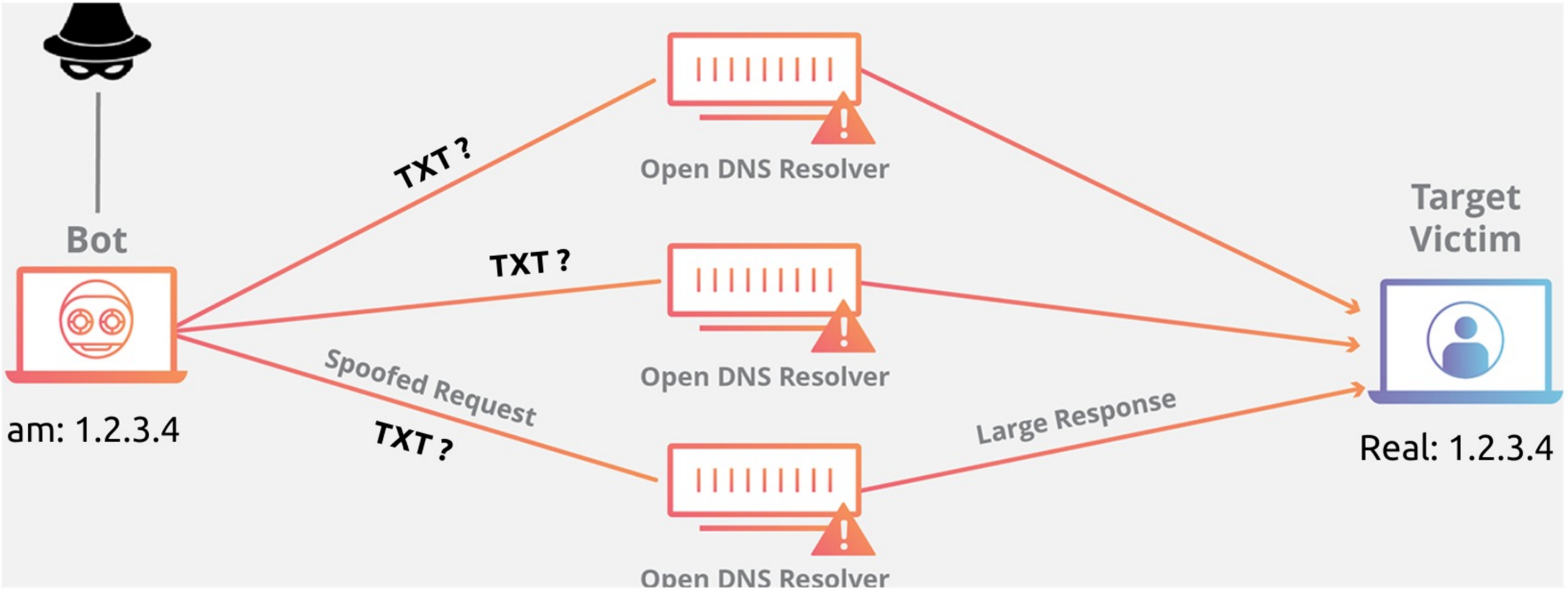


Image: Cloudflare.com

DNS Attacks - NX Domain

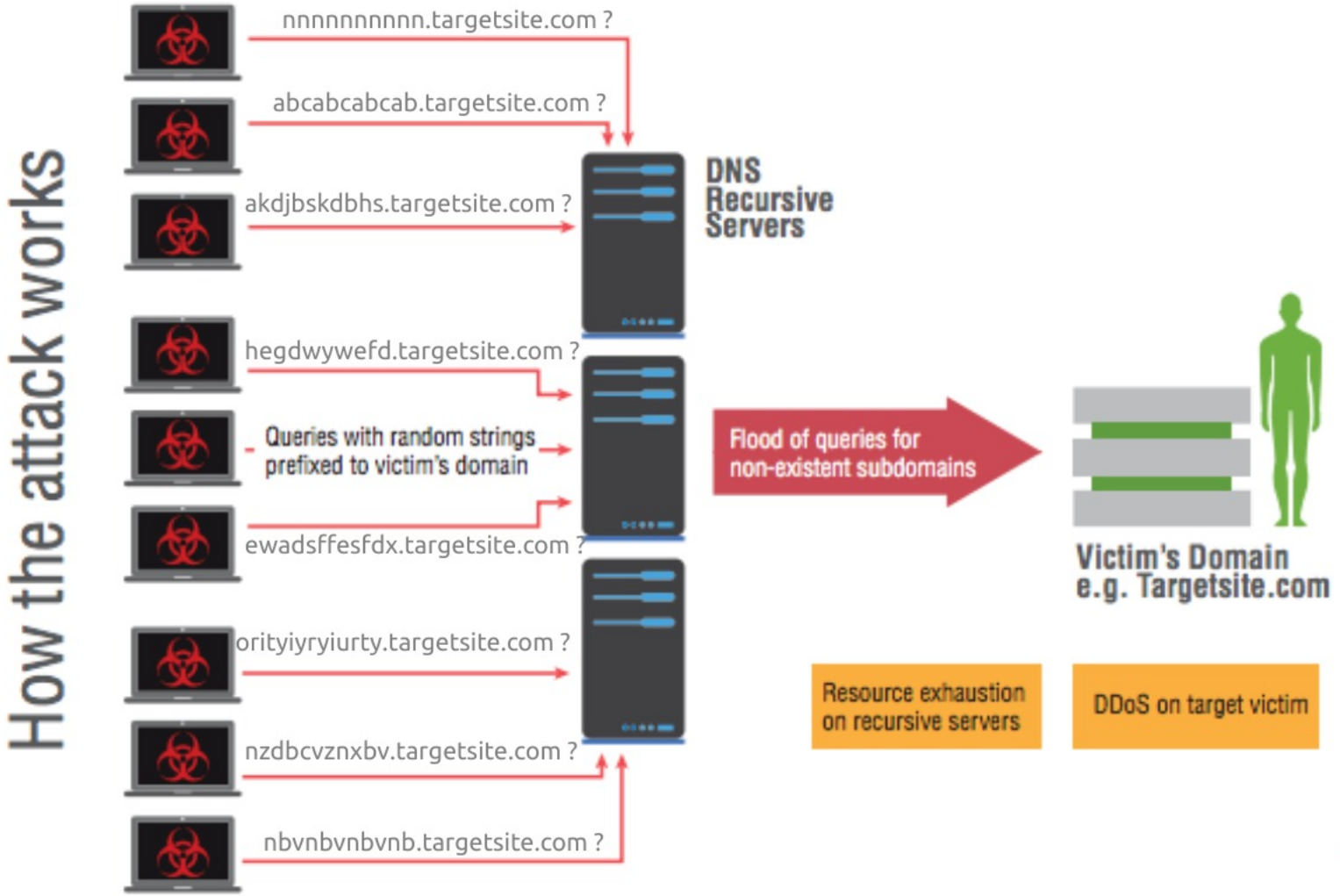
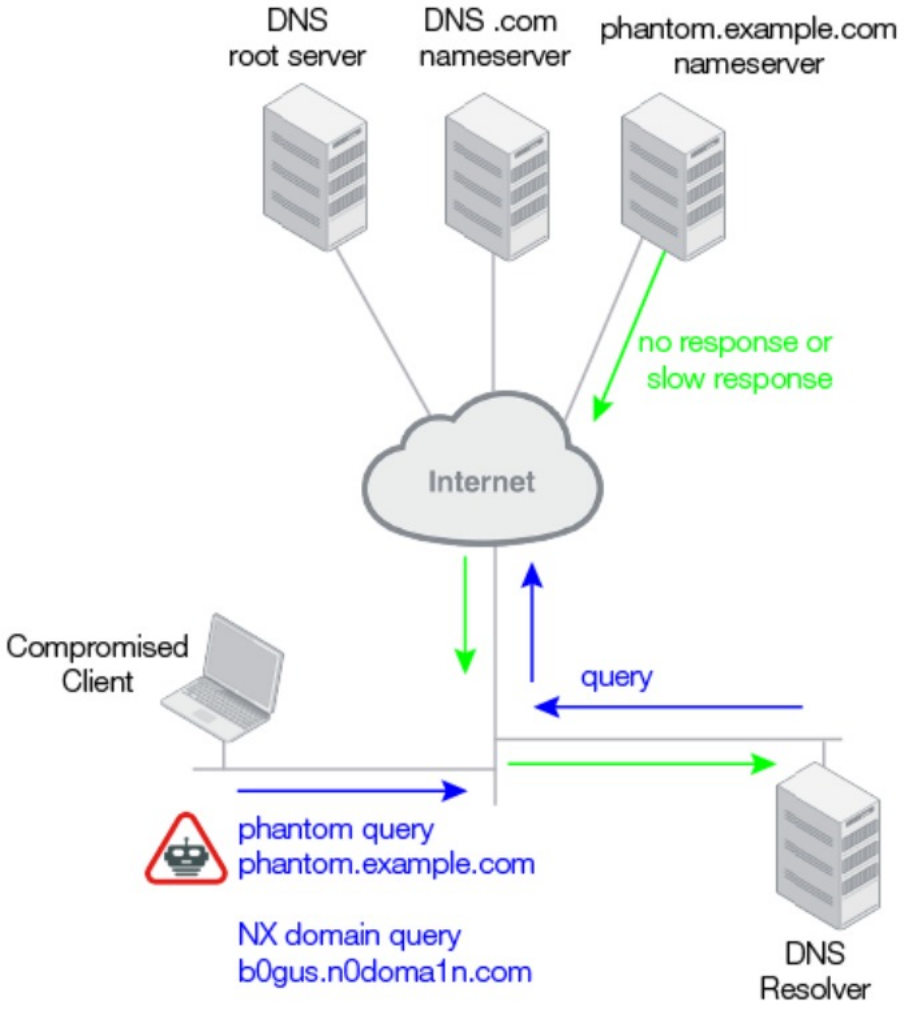


Image: Infoblox.com

DNS attacks - Phantom / Lock-up

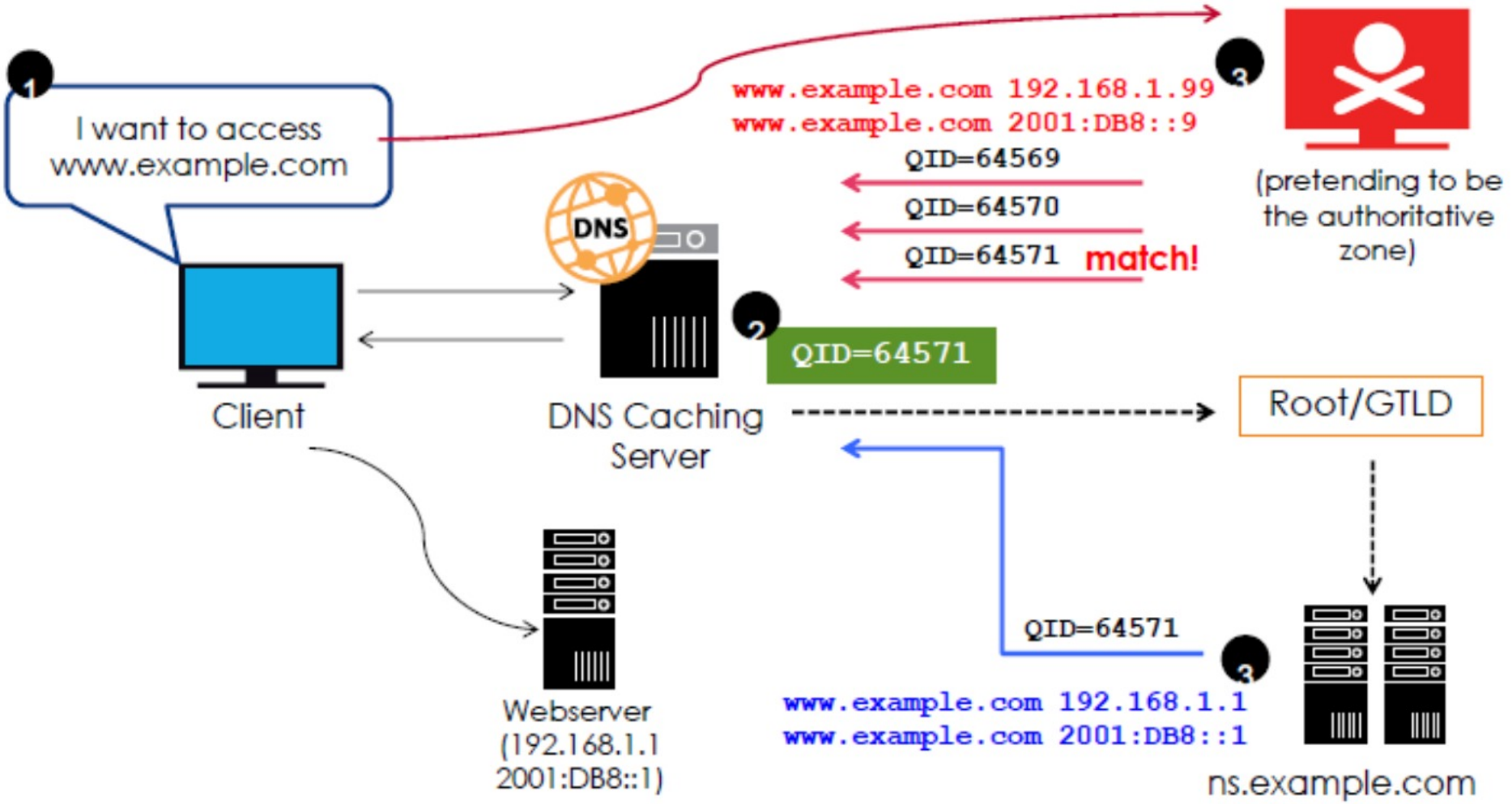


> Phantom = UDP

> Lock-up = TCP

Image: fortinet.com

DNS attacks - Cache poisoning



DNS attacks - Hijacking

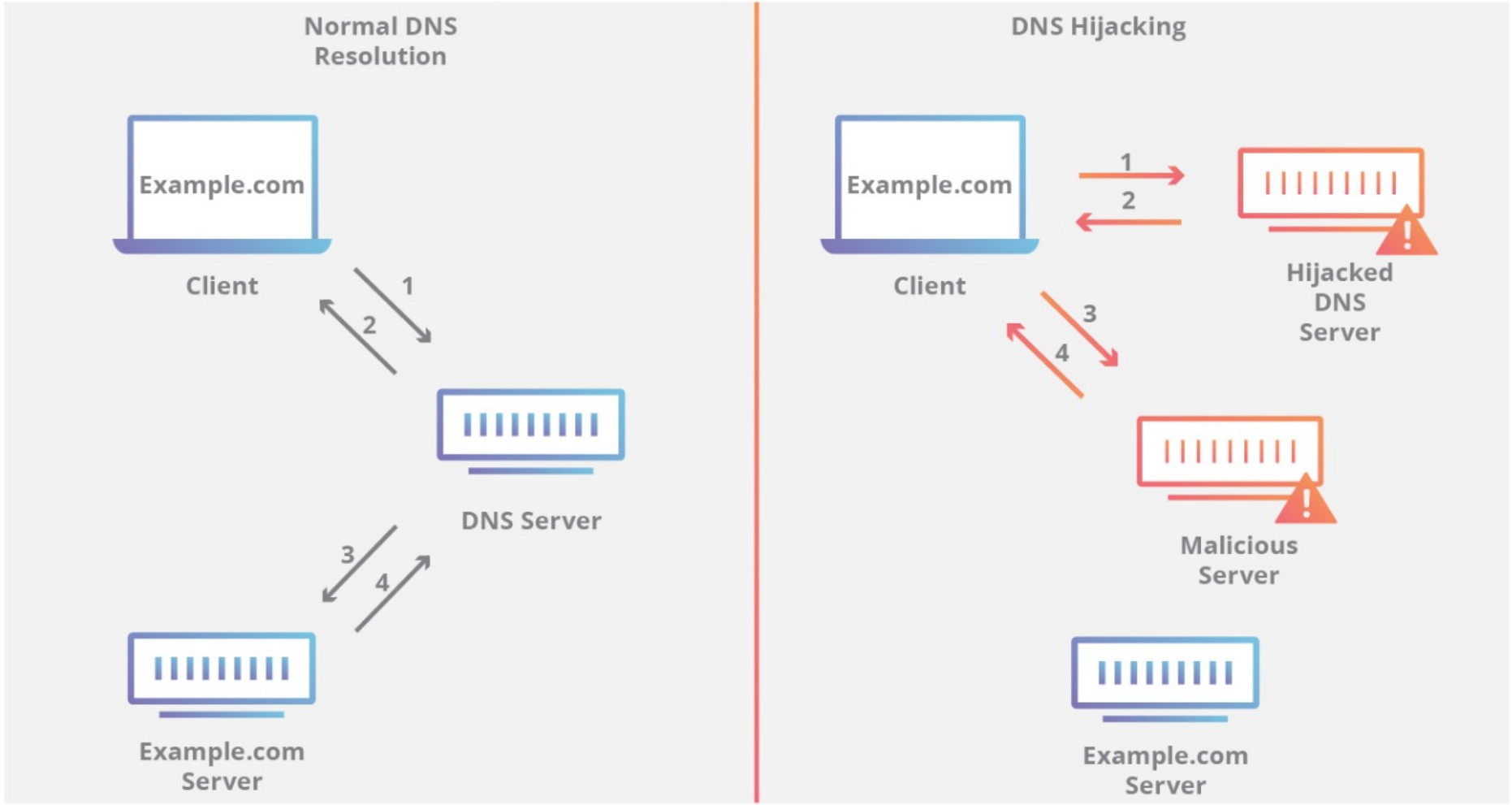


Image: Cloudflare.com

DNS attacks - Tunneling

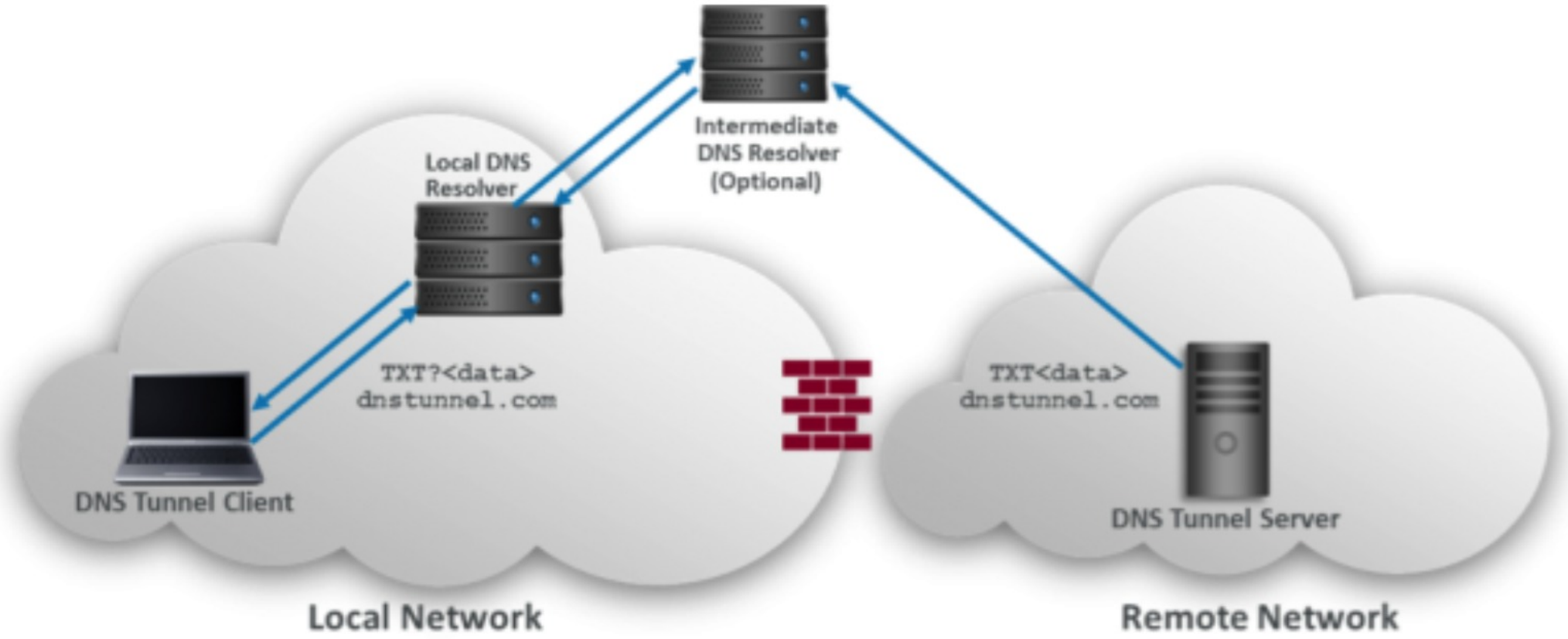
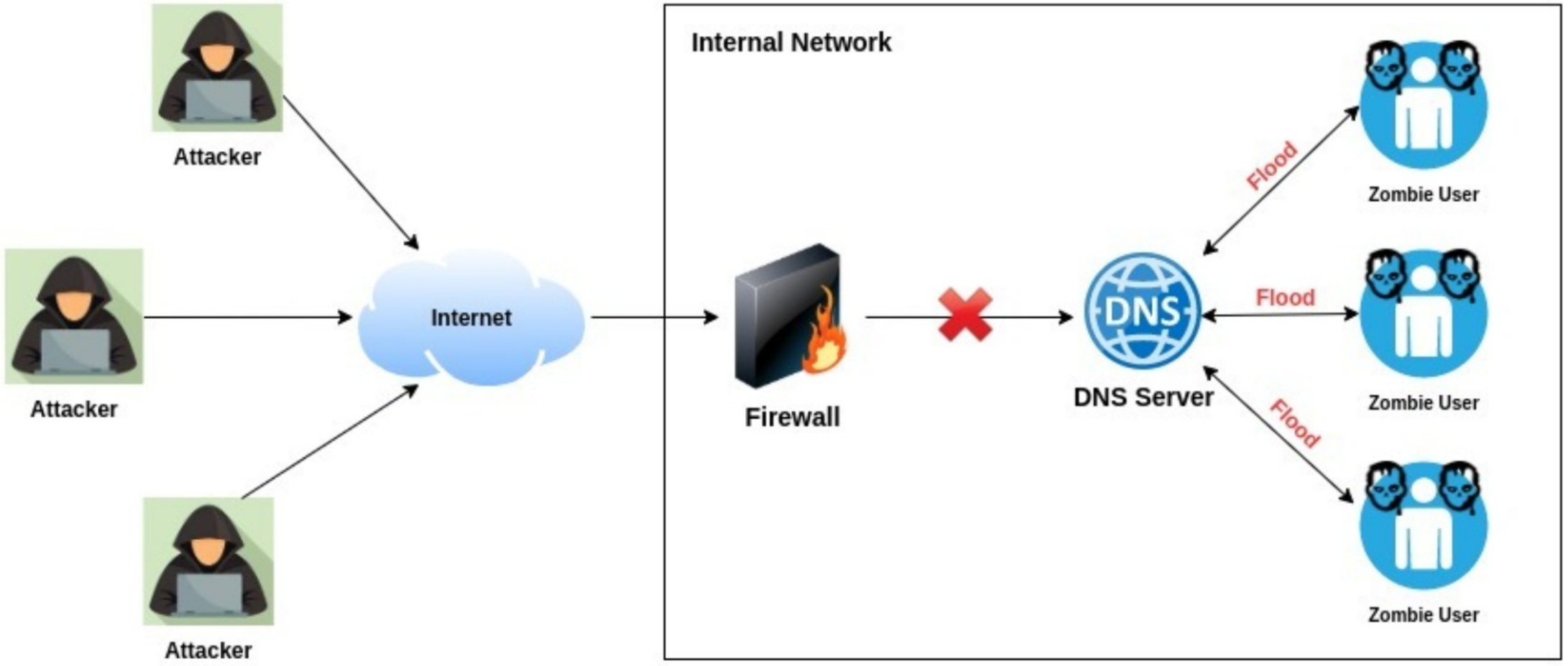


Image: Inside-out.xyz

Not open resolver ? No problem !



DNSDist

- A highly DNS-, DoS- and abuse-aware loadbalancer
- Route traffic to the best server, delivering top performance to legitimate users
- Shunting or blocking abusive traffic.
- Dnsdist is dynamic, its configuration language is Lua and it can be changed at runtime
- Its statistics can be queried from a console-like interface or an HTTP API.

DNSDist - Sample Config

```
setLocal("192.0.2.1")
addLocal("2001:db8::2:1")


addACL("192.0.2.0/24")
addACL("2001:db8::/32")

newServer("198.51.100.1", qps=20)
newServer("198.51.100.2", qps=1000)


setServerPolicy(firstAvailable)

-- first server within its QPS limit
```

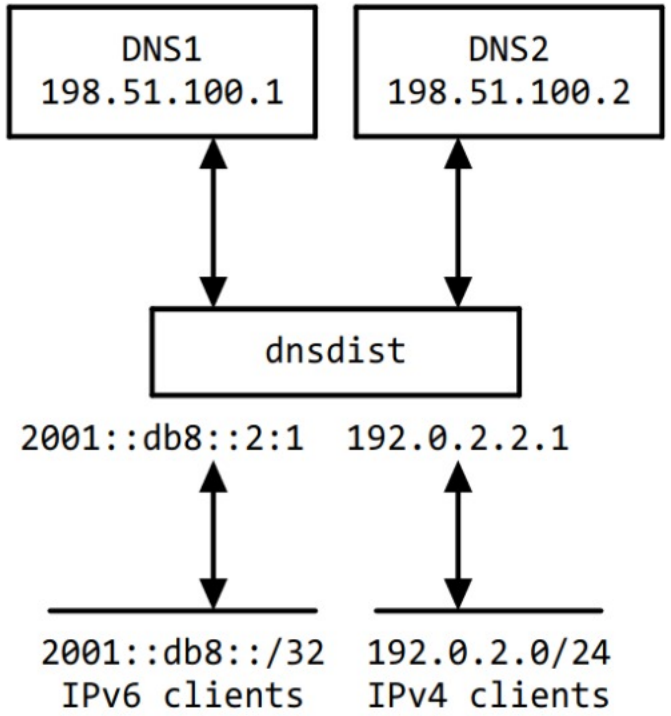


Image: berthub.eu

DNSDist - Server pools

- Any number of servers can belong to a group

```
newServer({address="192.0.2.3"})

newServer({address="192.0.3.3", pool="abuse"})

addAction({'bad-domain1.example', 'bad-domain2.example.'}, PoolAction("abuse"))
```

Send all queries for "bad-domain1.example." and "bad-domain2.example" to the "abuse" pool

DNSDist - Load balancing methods

- `leastOutstanding`
- `firstAvailable`
- `wrandom`
- `whashed`
- `chashed`
- `roundrobin`
- Custom Lua policies

DNSDist - Custom load balancing

```
newServer("192.168.1.2")
newServer({address="8.8.4.4", pool="numbered"})

function splitSetup(servers, dq)
  if(string.match(dq.qname:toString(), "%d"))
  then
    print("numbered pool")
    return leastOutstanding.policy(getPoolServers("numbered"), dq)
  else
    print("standard pool")
    return leastOutstanding.policy(servers, dq)
  end
end

setServerPolicyLua("splitsetup", splitSetup)
```

DNSDist - Effective Caching

```
pc = newPacketCache(10000, {maxTTL=86400, minTTL=0, temporaryFailureTTL=60,
staleTTL=60, dontAge=false})

getPool(""):setCache(pc)
```

- Parameters:
- The first parameter (10000) is the maximum number of entries stored in the cache, and is the only one required.
- maxTTL: the maximum lifetime of an entry in the cache.
- minTTL: the minimum TTL an entry should have to be considered for insertion in the cache.
- TemporaryFailureTTL: the TTL used for a Server Failure or a Refused response.
- StaleTTL: the TTL that will be used when a stale cache entry is returned.
- dontAge: a boolean that when set to true, avoids reducing the TTL of cached entries.

DNSDist - Packet route

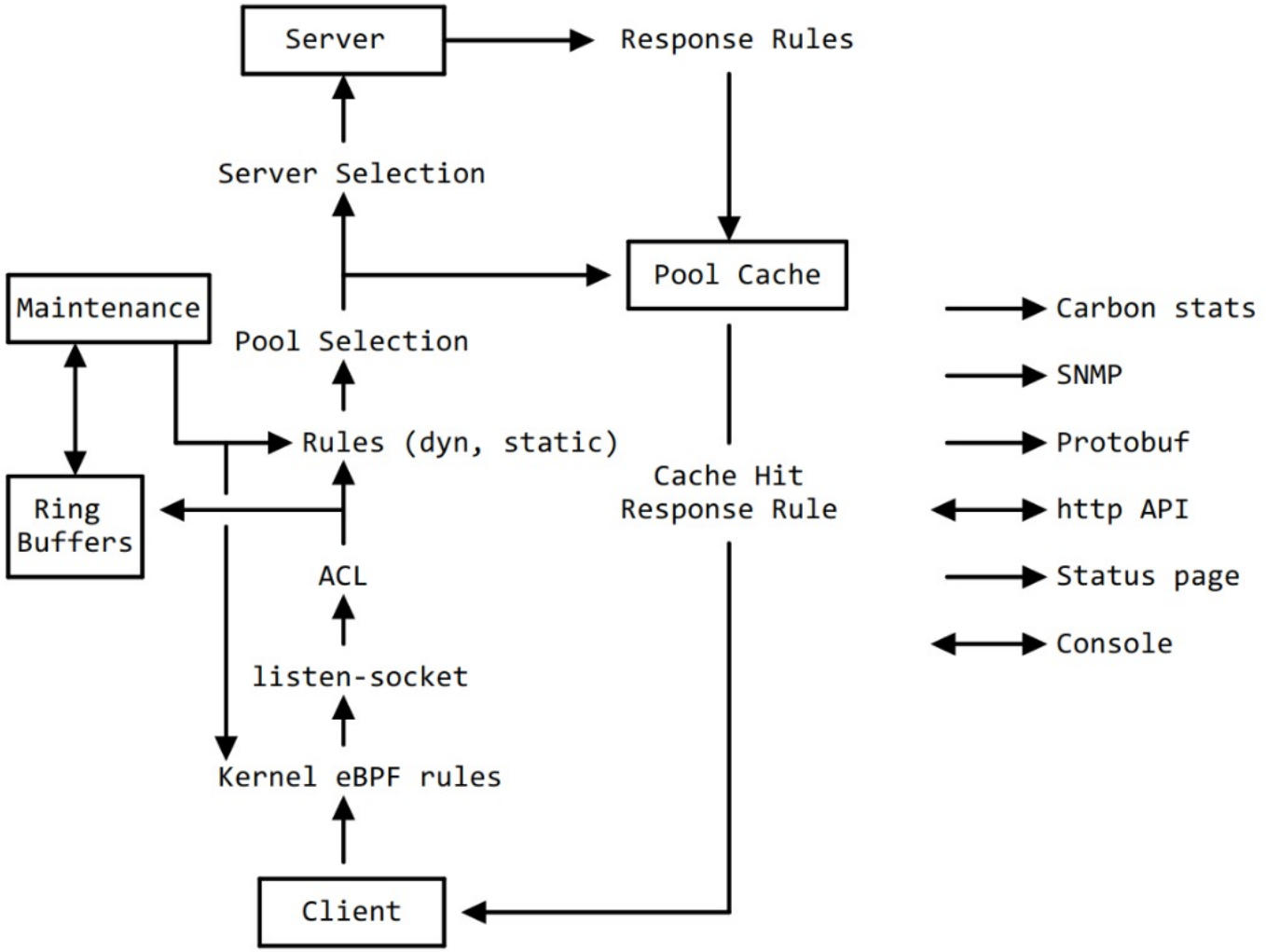


Image: berthub.eu

DNSDist - Packet policies

- Receive packets on one or several listening addresses
- Checking ACL
- Matching based on rules (Dyn, Static)
- Perform actions:
  - *Drop*
  - *Turn into an answer directly*
  - *Forward to a downstream server*
  - *Modify and forward to a downstream and modify back*
  - *Delay*

DNSDist - Packet Policies - Example 1

```
addAction(MaxQPSIPRule(5), DelayAction(100))
```

- This measures traffic for any traffic per IPv4 or IPv6 /64
- if traffic for such an address (range) exceeds 5 qps, it gets delayed by 100ms.

DNSDist - Packet Policies - Example 2

```
addAction(MaxQPSIPRule(5, 32, 48), NoRecurseAction())
```

- Strips the Recursion Desired (RD) bit from traffic per any IPv4 address and per /48 of IPv6 that exceeds 5 qps.
- This means any those traffic bins is allowed to make a recursor do ‘work’ for only 5 qps.

DNSDist - Packet Policies - Example 3

```
addAction(MaxQPSIPRule(5), TCAction())
```

- This will respectively drop traffic exceeding that 5 QPS limit per IP or range, or return it with TC=1, forcing clients to fall back to TCP.
- To Drop:

```
addAction(MaxQPSIPRule(5), DropAction())
```

DNSDist - Dynamic rules

- To set dynamic rules, based on recent traffic, define a function called `maintenance()` in Lua.
- It will get called every second, and from this function you can set rules to block traffic based on statistics.

```
function maintenance()
  addDynBlocks(exceedQRate(20, 10), "Exceeded query rate", 60)
end
```

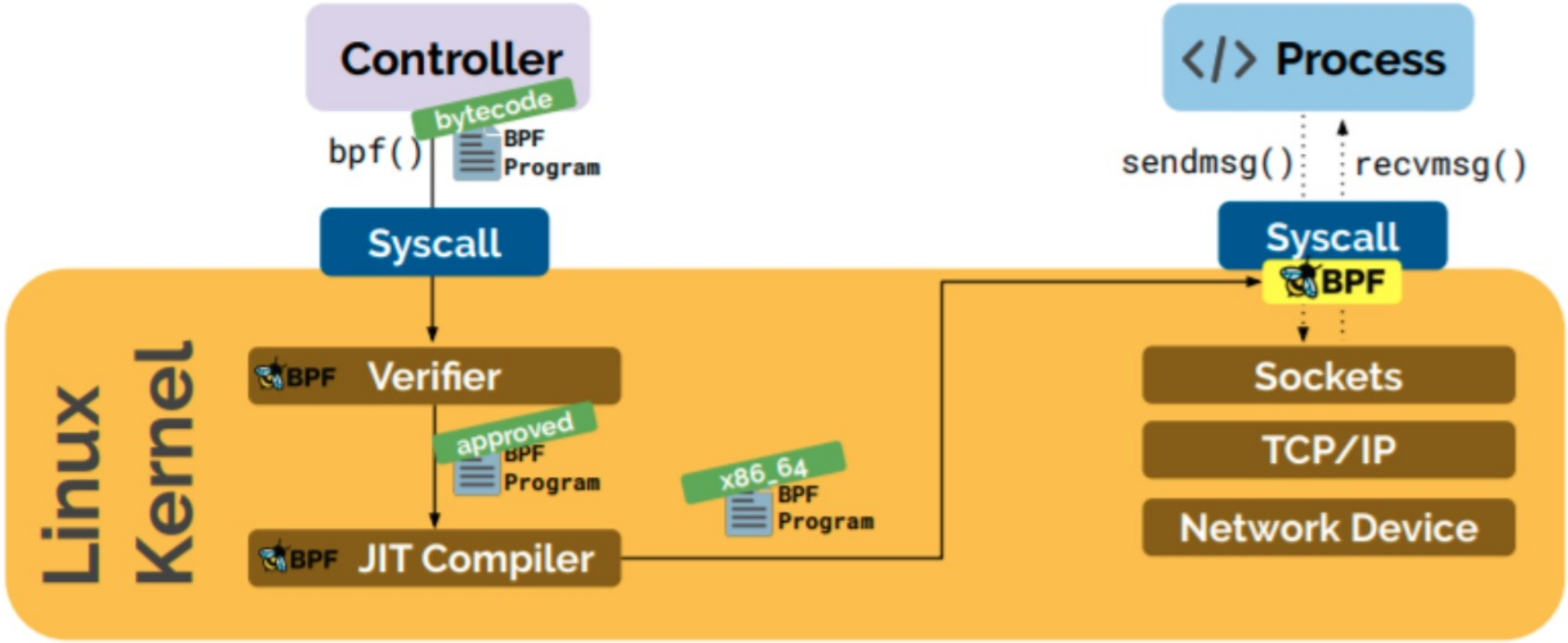
- This will dynamically block all hosts that exceeded 20 queries/s as measured over the past 10 seconds, and the dynamic block will last for 60 seconds.

DNSDist - Dynamic rules example

```
function maintenance()
  addDynBlocks(exceedQRate(30, 10), "Exceeded query rate", 60)
  addDynBlocks(exceedNXDOMAINs(20, 10), "Exceeded NXD rate", 60)
  addDynBlocks(exceedServFails(20, 10), "Exceeded ServFail rate", 60)
  addDynBlocks(exceedQTypeRate(DNSQType.ANY, 5, 10), "Exceeded ANY rate", 60)
  addDynBlocks(exceedRespByterate(1000000, 10), "Exceeded resp BW rate", 60)
end
```

Note: Bandwidth unit is number of bytes per second (in this case 1MB)

eBPF


Controller
bytecode
bpf()
BPF
Program
Syscall
Linux
Kernel
BPF Verifier
approved
BPF
Program
BPF JIT Compiler
x86_64
BPF
Program
</> Process
sendmsg()
recvmsg()
Syscall
BPF
Sockets
TCP/IP
Network Device

DNSDist - eBPF support

- Enhanced version of BPF (Berekly Packet Filtering)
- Available on Linux kernel (4.1+) with compiling flags: `CONFIG_BPF`, `CONFIG_BPF_SYSCALL`, ideally `CONFIG_BPF_JIT`)
- Discard incoming packets in kernel-space
- Example dynamic rule with eBPF:

```
bpf = newBPFFilter(1024, 1024, 1024) -- maxV4, maxV6, maxQNames
setDefaultBPFFilter(bpf)
dbpf = newDynBPFFilter(bpf)
function maintenance()
        addBPFFilterDynBlocks(exceedQRate(20, 10), dbpf, 60)
        dbpf:purgeExpired()
end
```

DNSDist - SO_REUSEPORT

- Added on Linux kernel (3.9+)
- Allows multiple sockets on the same host to bind to the same port
- The kernel distributes packets to listening sockets and apps
- DNDDist sample config:

```
addLocal("192.0.2.1:53", {reusePort=true})
addLocal("192.0.2.1:53", {reusePort=true})
addLocal("192.0.2.1:53", {reusePort=true})
addLocal("192.0.2.1:53", {reusePort=true})
```

DNSDist - DoH, DoT, DNSCrypt

- DoT support:

```
addTLSLocal('192.0.2.55', '/etc/ssl/certs/example.com.pem',
'/etc/ssl/private/example.com.key')
```

- DoH support:

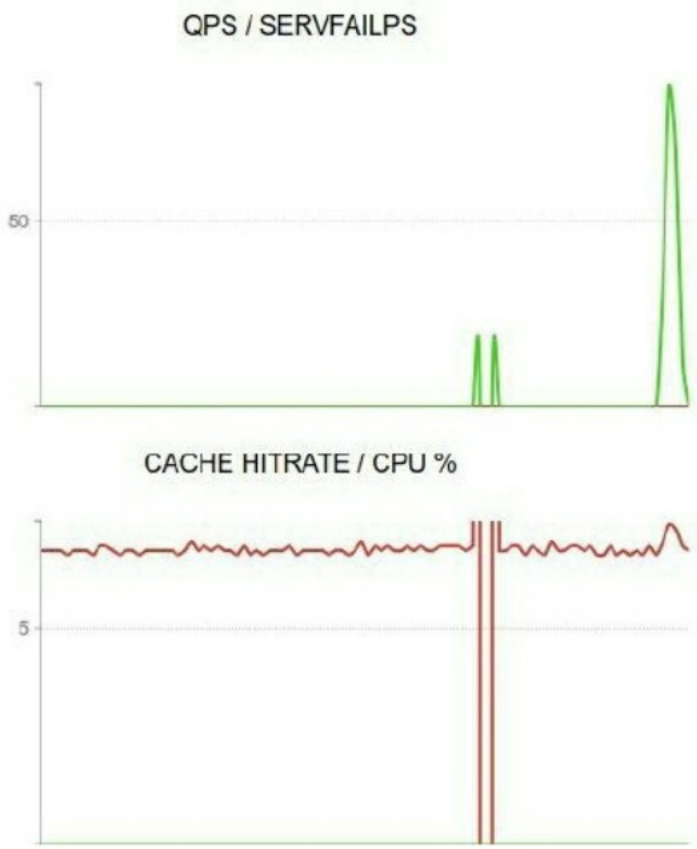
```
addDOHLocal('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem',
'/etc/ssl/private/example.com.key')
```

- DNSCrypt support:

```
addDNSCryptBind("127.0.0.1:8443", "2.providername",
"/path/to/resolver.cert", "/path/to/resolver.key")
```

DNSDist - Builtin webserver

- `webserver("127.0.0.1:8083", "supersecretpassword", "supersecretAPIkey")`



#	Name	Address	Status	Latency	Queries	Drops	QPS	Out	Weight	Order	Pools
0	PDNS-Recursor1	127.0.0.1:1001	up	0.04	38	0	0.00	0	1	1	
1	PDNS-Recursor2	127.0.0.1:1002	up	0.04	41	0	0.00	0	1	1	
2	Bind-Recursor1	127.0.0.1:2001	up	0.04	30	0	0.00	0	1	1	
3	Bind-Recursor2	127.0.0.1:2002	up	0.04	24	0	0.00	0	1	1	
4	Unbound-Recursor1	127.0.0.1:3001	up	0.04	68	0	0.00	0	1	1	

#	Rule	Action	Matches
0	((qtype==AXFR) || (qtype==IXFR))	set rcode 5	0
1	qtype==AAAA	drop	0

#	Response Rule	Action	Matches
	No response rules defined		

Dyn blocked netmask	Seconds	Blocks	Reason
No dynamic blocks active			

Kernel-based dyn blocked netmask	Seconds	Blocks
192.168.0.220	60	0

References

- https://dnsdist.org
- https://www.infoblox.com/wp-content/uploads/infoblox-ebook-top-ten-dns-attacks.pdf
- https://berthub.eu/tmp/dnsdist-md/dnsdist-diagrams.md.html
- Http://www.brendangregg.com/ebpf.html
- https://www.infoq.com/presentations/facebook-google-bpf-linux-kernel/

Demo

Demo Time