# FRNOG 25 Meeting:
## BIND9 – Recursive Client Rate limiting

Cathy Almond,  Sr. Technical Support Engineer

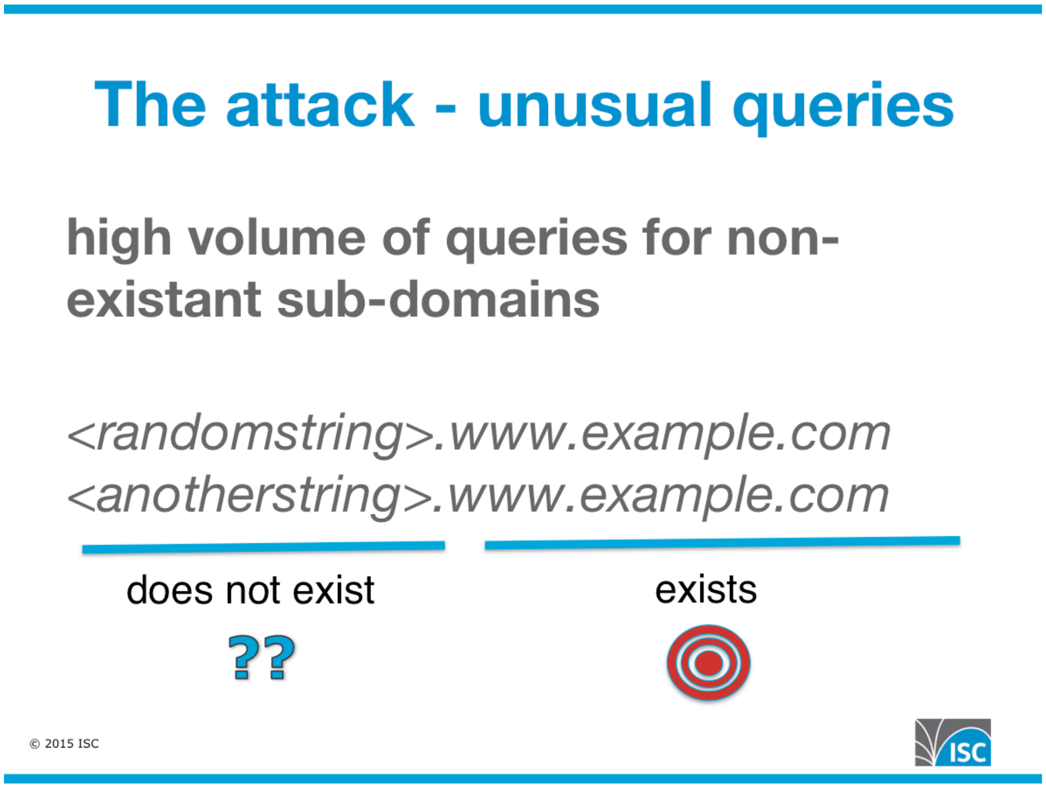
## Presenter

Cathy Almond

ISC Senior Technical Support Engineer, Support Team Lead

## Agenda

1. **Pseudo-random subdomain attack**
2. Recognizing the attack
3. Recommended mitigation
4. Results from live environments
5. Any questions?



## The attack - unusual queries

high volume of queries for non-existant sub-domains

*<randomstring>.www.example.com*
*<anotherstring>.www.example.com*

does not exist ??

exists

**HEADLINE: This type of random name query traffic is now well-known to most operators of large Recursive DNS Servers**

In early 2014 ISC and other DNS server providers started to receive many reports of problems encountered by DNS Recursive Servers (though the earliest known report was in 2009).

Initially believed to be an attack on Recursive Servers, but later understood to be targeting Authoritative Servers (so became known as 'collateral damage' or 'water torture')

The signature of this attack traffic is a series of client queries, for a domain that exists, but where the name in the domain does not exist.

The non-existent names are all different (or if they repeat, they do so only as a client timeout and retry, or after the original series of names is exhausted and repeats).

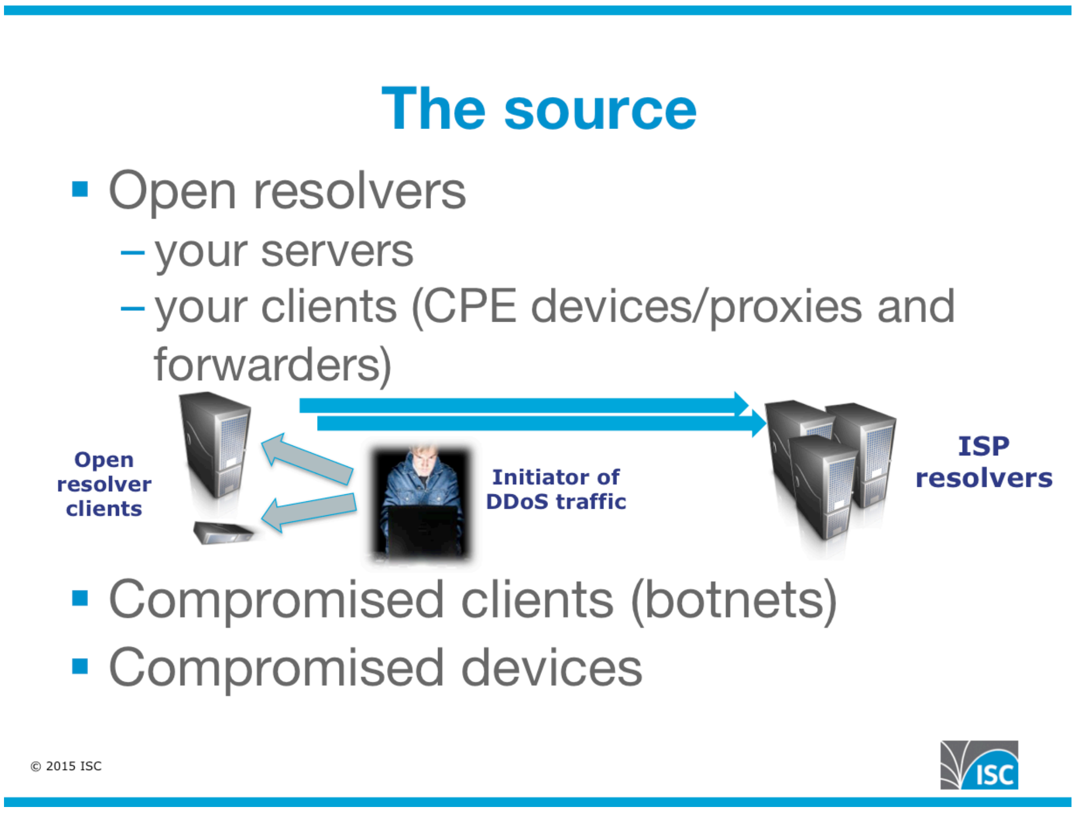
Often the 'www' label is included between the main domain name labels and the random 'attack' label.

Analysis of the names by other DNS industry researchers suggests that they're generated by an algorithm, so while they appear to be random to humans, they're actually not truly random from a mathematical cryptographic perspective.   So they're only 'PSEUDO-RANDOM'.

(But it's their uniqueness rather than their randomness that is significant for causing the most harm to their targets.)

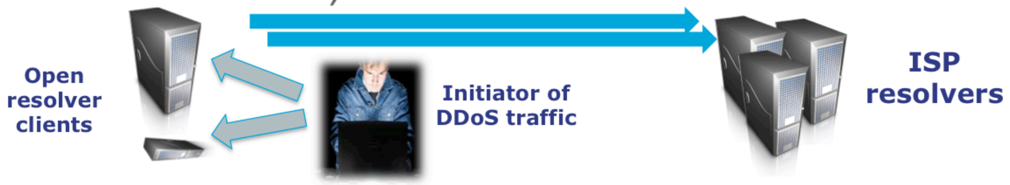
** The expected response to these queries is NXDOMAIN

** This is because, whilst the domain exists, the name within that domain does not



## The source

- Open resolvers
  - your servers
  - your clients (CPE devices/proxies and forwarders)



- Compromised clients (botnets)
- Compromised devices

**HEADLINE: There are many sources of this traffic, and the sophistication of the attack continues to evolve…**

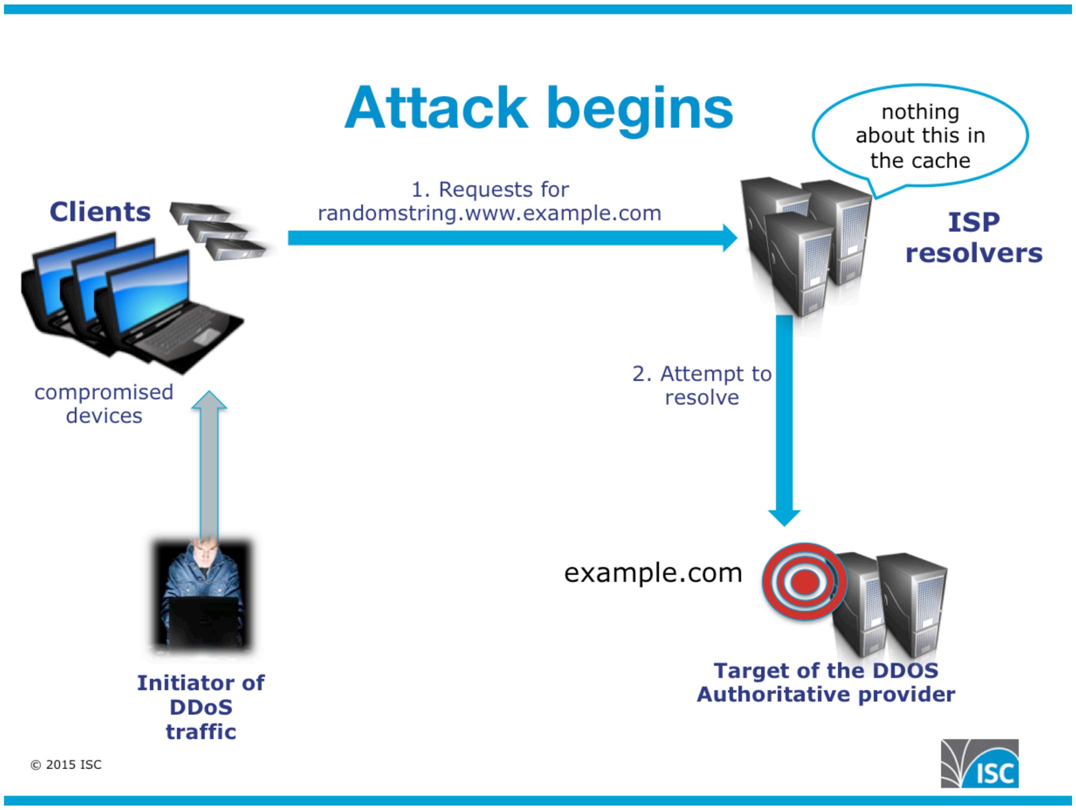
EarlyDDoS initiators made liberal use of DDoS for hire services that took advantage of lists of open IPv4 resolvers (readily available) as a means of spreading the source of the client queries to make them harder to detect and block.

Much of this query traffic found its way to ISP resolvers, by means of insecure home gateways that were acting as open resolvers by forwarding queries received on their Internet-facing IP address, to the network provider's recursive DNS server(s).  Typically rach client sends only a few queries – making it unrealistic to identify specific sources of badness in order to block them. Other open-resolver scenarios include running resolvers that are themselves open or providing forwarding services to business customers who are running open resolvers on their own sites.

(Note though: if you have clients that have broken CPE devices, attack queries *to* those devices will be traversing your network to reach them – you could, quite reasonably in most cases, block queries *TO* port 53 on your addresses you assign to your clients)

New and more sophisticated approaches are now being identified, such as compromised clients or devices (e.g. Internet webcams) running malware.

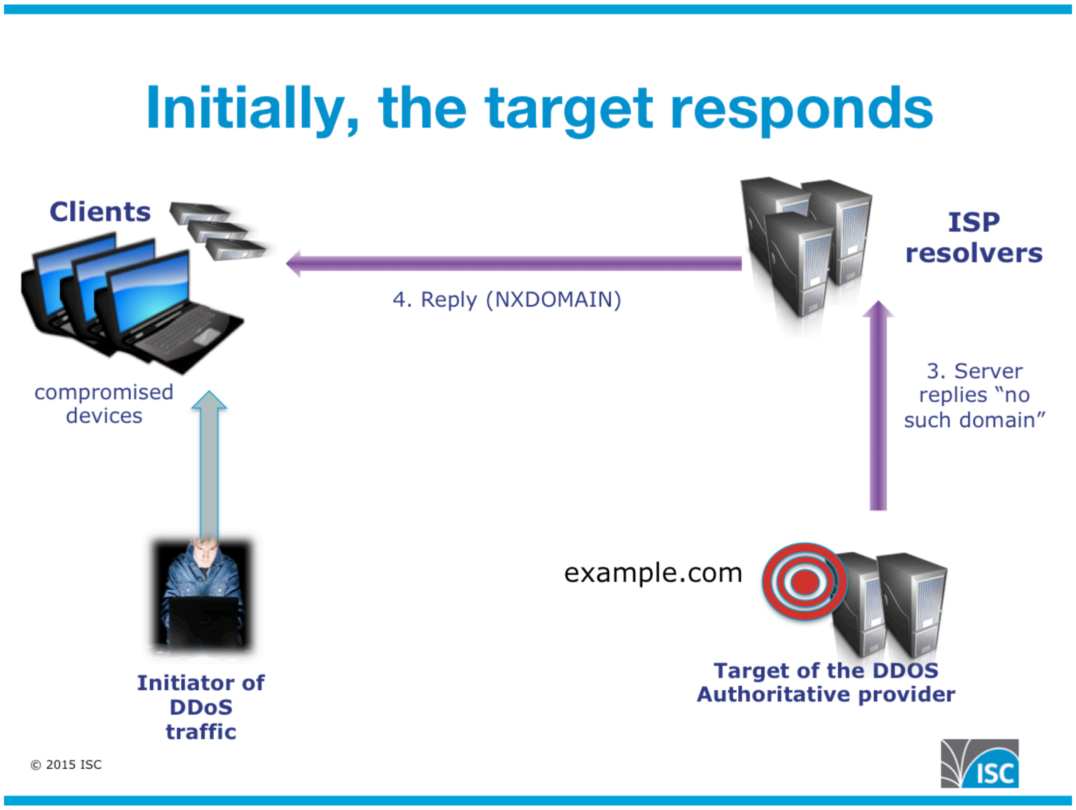
With the more recent approaches, the query load per client is much much higher, and the attackers often seek to obscure the source by spoofing the source address to be one (or several) that are still within the client's subnet.



## Attack begins

nothing about this in the cache
Clients
1. Requests for randomstring.www.example.com
ISP resolvers
compromised devices
2. Attempt to resolve
example.com
Initiator of DDoS traffic
Target of the DDOS Authoritative provider
ISC

**HEADLINE: Why this is a problem for Recursive Server Operators…. (the next series of slides explains why)**



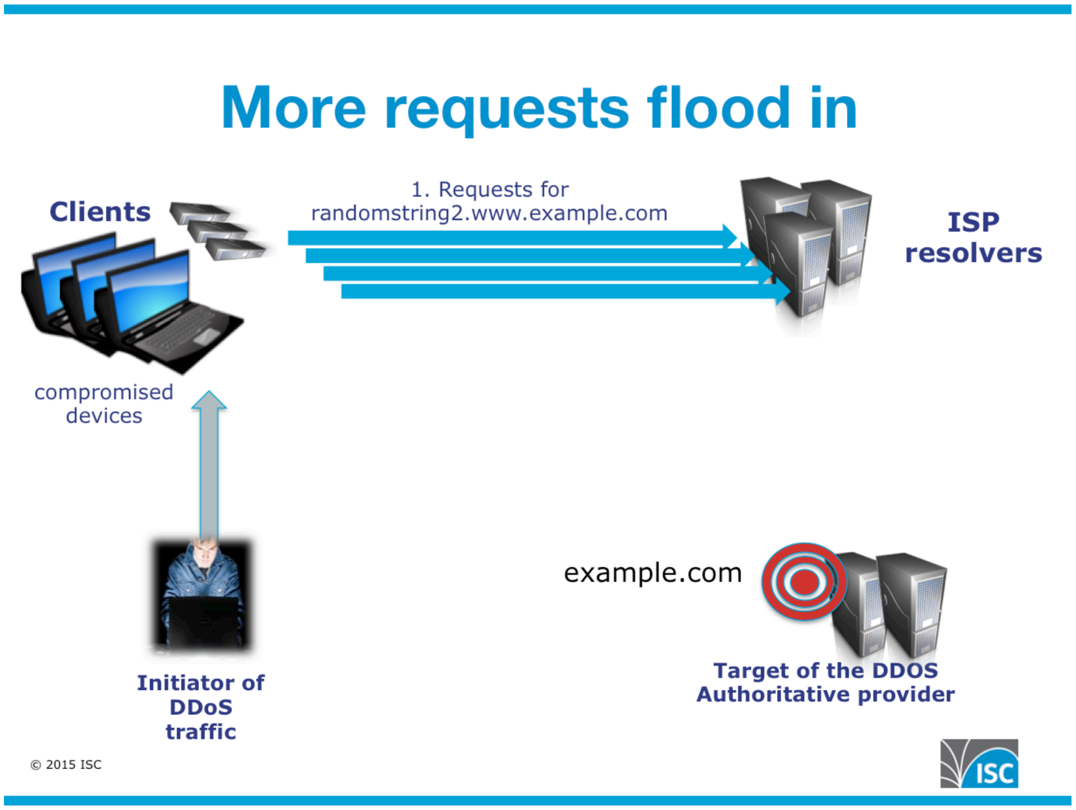
## Initially, the target responds

Clients
ISP resolvers
4. Reply (NXDOMAIN)
3. Server replies "no such domain"
compromised devices
example.com
Initiator of DDoS traffic
Target of the DDOS Authoritative provider

**HEADLINE: One client query and one NXDOMAIN response is a normal occurrence.**

On it's own, this is a simple query, and the response is NXDOMAIN

What could go wrong?

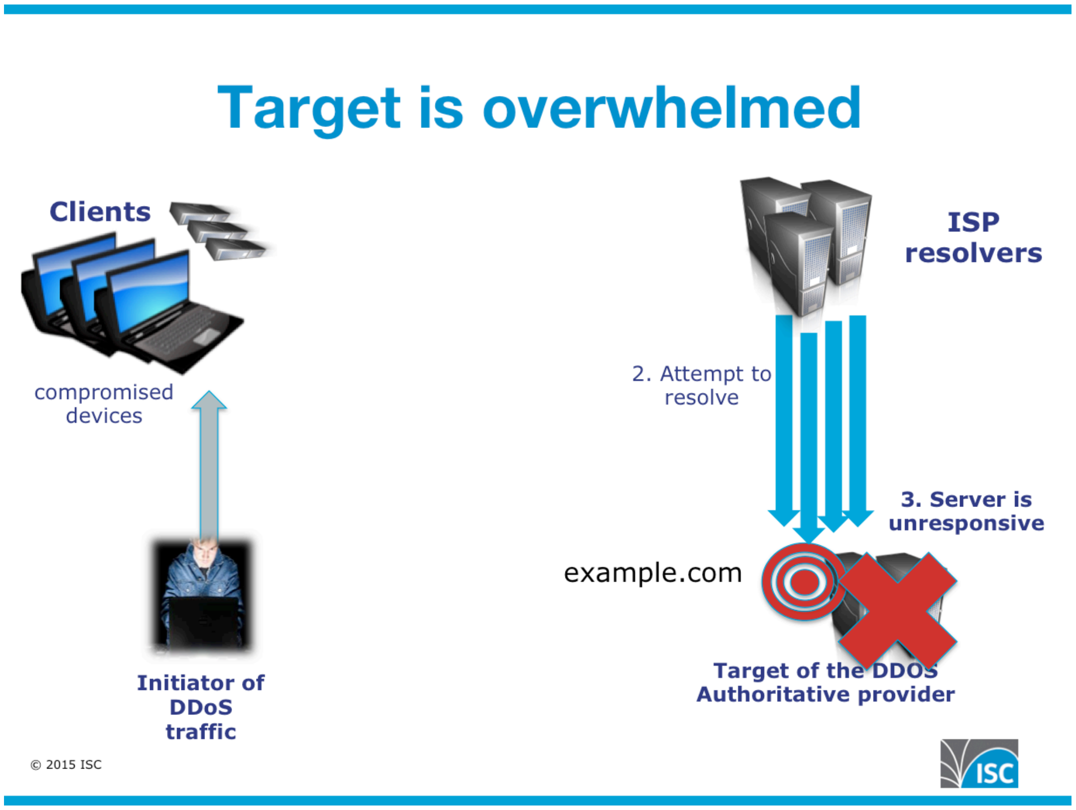


## More requests flood in

Clients
1. Requests for randomstring2.www.example.com
ISP resolvers
compromised devices
example.com
Initiator of DDoS traffic
Target of the DDOS Authoritative provider
© 2015 ISC
ISC

**HEADLINE: Many unique client queries, each requiring full recursion to return an NXDOMAIN is abnormal**

But it’s not just one client query – it’s many, and they’re all unique (apart from retries).



## Target is overwhelmed

Clients
compromised devices
Initiator of DDoS traffic
ISP resolvers
2. Attempt to resolve
3. Server is unresponsive
example.com
Target of the DDOS Authoritative provider
© 2015 ISC

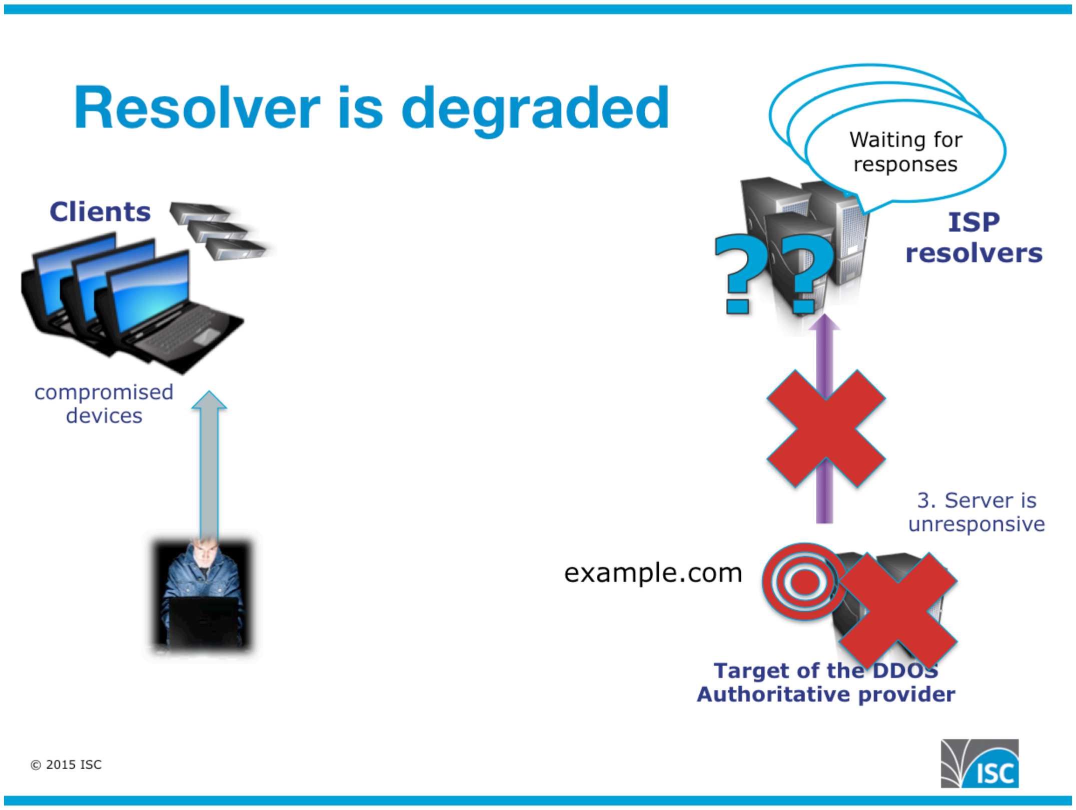
**HEADLINE: Typically, under this onslaught, the authoritative servers will become unresponsive**

Since they're all unique, there's nothing in cache – each one has to be answered by the resolver sending the query to the authoritative servers for the domain

Eventually the target is overwhelmed (it is almost certainly be stormed by client queries from all over the Internet)

OR

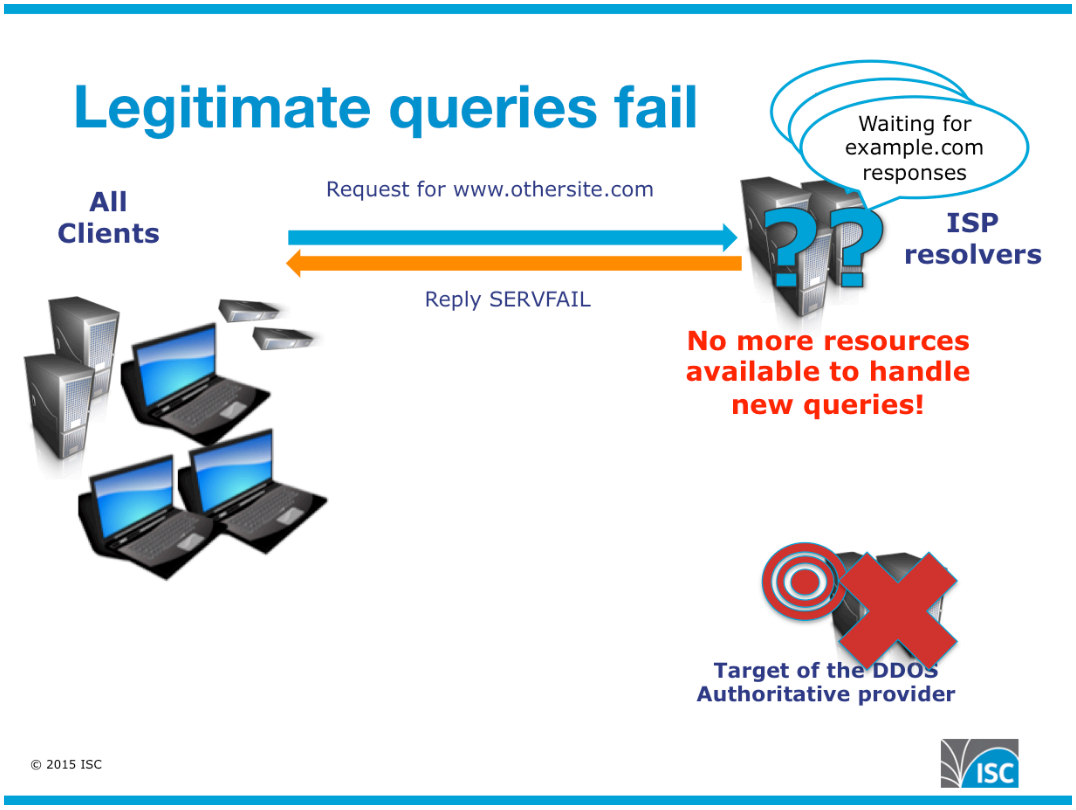
The target implements traffic rate limiting – and since our resolver in the example above is the source of a lot of queries… (next slide…)



## Resolver is degraded

**HEADLINE: It is a problem for a Recursive Server when Authoritative servers stop responding to them**

No answers come back – and the Resolver is left hanging around waiting for many responses to queries that it has sent out.



## Legitimate queries fail

All Clients
Request for www.othersite.com
Reply SERVFAIL
Waiting for example.com responses
ISP resolvers
No more resources available to handle new queries!
Target of the DDOS Authoritative provider

**HEADLINE: Recursive Servers only have finite resources with which to keep track of pending queries**

This is the point where recursive servers start to feel the heat. When handling a client query that can't be answered immediately from cache or from an authoritative zone that it serves, a recursive server has to maintain a 'context' for the client – that is, what has been asked, who asked it etc.. while it goes off to get the answer from the Authoritative Servers on the Internet.

The BIND configuration option 'recursive-clients' provides a maximum of how many of these contexts named can store, before it has to stop accepting new recursive client queries for resolving.

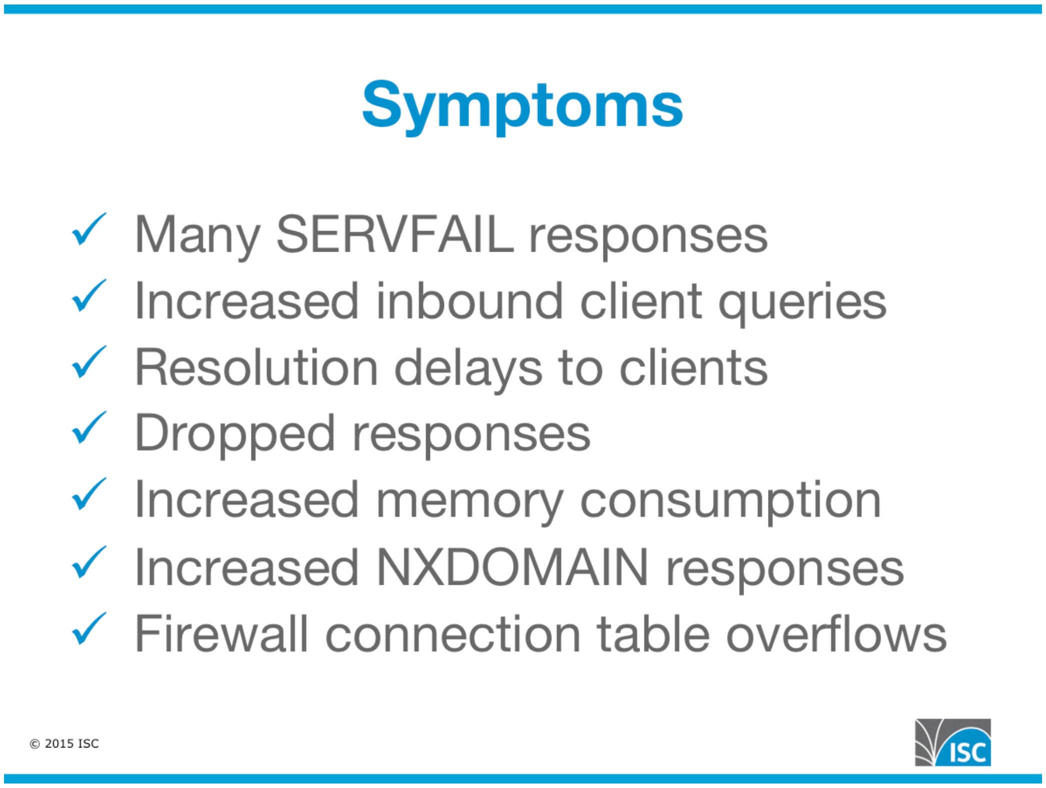
This controls the use of BIND's resources – directly (the number of clients that are allowed to be 'waiting for a response' at any moment in time), and indirectly (the other resources that named is using for doing this onward resolution, such as tracking the onward queries themselves and open sockets etcetera).

Two things can start happening at this point:

1) New (legitimate) client queries can't be accepted, because the limit has been reached (dropped or SERVFAILed)
2) New client queries are accepted (because the limit is set very high), but still fail, because other resources are exhausted.

## 2. RECOGNIZING THE ATTACK

**HEADLINE: How do you know if your servers are suffering from this type of attack?**



## Symptoms

- ✓ Many SERVFAIL responses
- ✓ Increased inbound client queries
- ✓ Resolution delays to clients
- ✓ Dropped responses
- ✓ Increased memory consumption
- ✓ Increased NXDOMAIN responses
- ✓ Firewall connection table overflows

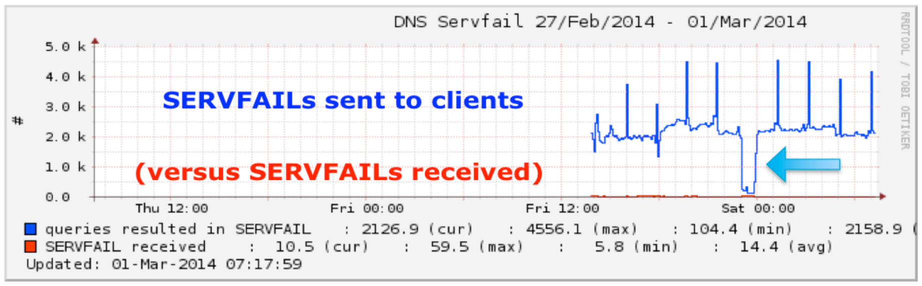
**HEADLINE: It is A Big Clue when you see your servers sending back SERVFAIL responses to clients that should resolve OK.**

The first symptom that is usually noted and investigated is an increase in SERVFAIL responses to clients – the legitimate ones.

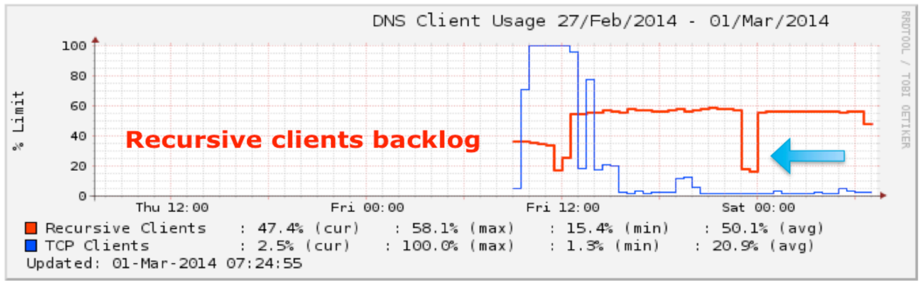
After this, it depends what else is being monitored, but usually we hear about:

- An increase in inbound client queries (due to the attack traffic)
- Sometimes an increase in NXDOMAIN responses (replies to the attack queries)
- Resolution delays and failures (because the recursive server's resources are exhausted)
- Increased memory consumption by named (twofold – there may be more cached NXDOMAIN pseudo records in cache – remember each DDoS query is unique; also, the increase in resources handling the backlog of recursive client contexts and other structures requires named to use more memory)
- Sometimes the additional traffic translates to firewall problems too.

# Evidence



```
■ queries resulted in SERVFAIL   : 2126.9 (cur)   : 4556.1 (max)   : 104.4 (min)   : 2158.9 (
■ SERVFAIL received    :   10.5 (cur)   :   59.5 (max)   :    5.8 (min)   :   14.4 (avg)
Updated: 01-Mar-2014 07:17:59
```



```
■ Recursive Clients     : 47.4% (cur)    : 58.1% (max)    : 15.4% (min)    : 50.1% (avg)
■ TCP Clients           : 2.5% (cur)    : 100.0% (max)    : 1.3% (min)    : 20.9% (avg)
Updated: 01-Mar-2014 07:24:55
```

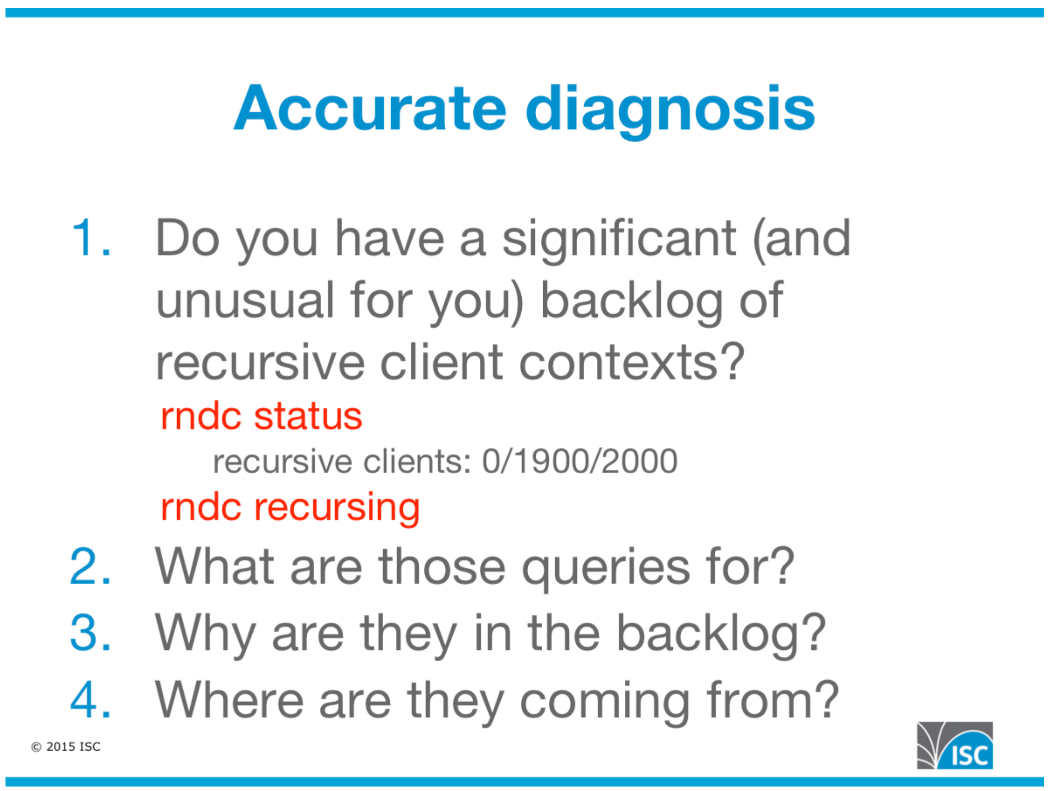
**HEADLINE: A big Recursive Clients backlog is A Bad Thing**

Here’s an early example of a server failing under attack (the attack commenced from earlier than the monitoring was initiated).

In the top graph we’re seeing the volume of SERVFAIL responses sent back to clients (versus the ones received from authoritative servers and passed on). In other words, these are LOCALLY GENERATED ‘I can’t resolve this’ responses. This is a huge SERVFAIL rate – and it’s much higher than just the increase in DDoS traffic – what’s happening is that legitimate client queries are getting SERVFAIL responses instead of answers.

In the bottom graph the number of outstanding recursive client queries (recursive client contexts is being tracked in parallel. The limit is 10,000 and while the attack is ongoing, is hitting around 6000. It’s this backlog which is both evidence of a serious problem, as well as a significant contributing factor to the failures of good client queries.

At the point where the onslaught paused briefly and the backlog dropped below 2000, suddenly the SERVFAIL rate plummets too (and at this time, named should only be sending back SERVFAIL responses to clients where the query really is unresolvable).



## Accurate diagnosis

1. Do you have a significant (and unusual for you) backlog of recursive client contexts?
   rndc status
      recursive clients: 0/1900/2000
   rndc recursing
2. What are those queries for?
3. Why are they in the backlog?
4. Where are they coming from?

**HEADLINE: Make sure that you are alerted to unusual increases in resource consumption and then look more closely at what is going on.**

From ‘rndc status’, the first number is the actual backlog of recursive client contexts. The other two numbers are the soft limit (calculated for you by named) and the hard limit (you configure this with option recursive-clients).

If you hit the hard limit, named just drops the client; if you hit the soft limit, named will still accept the new client query, but it will at the same time stop processing and send back a SERVFAIL to the oldest outstanding client query it has in its backlog.

The rndc recursing command tells named to dump ‘all client queries in progress’ to a file – by default this is named.recursing (in named’s default directory). With the output from rndc recursing, you can dig further. You’ll be able to see:

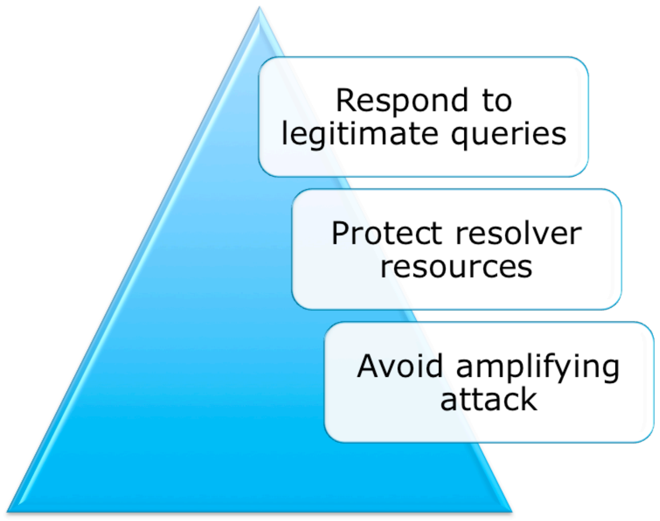
- Which clients are sending them
- What names are being queried

Looking at the open sockets can sometimes also be useful if the pattern of names in the query backlog is too diverse. Perhaps all of those diverse domains are hosted on the same small number of authoritative servers, none of whom are responding? Check your named server logging – are you hitting any resource limits?

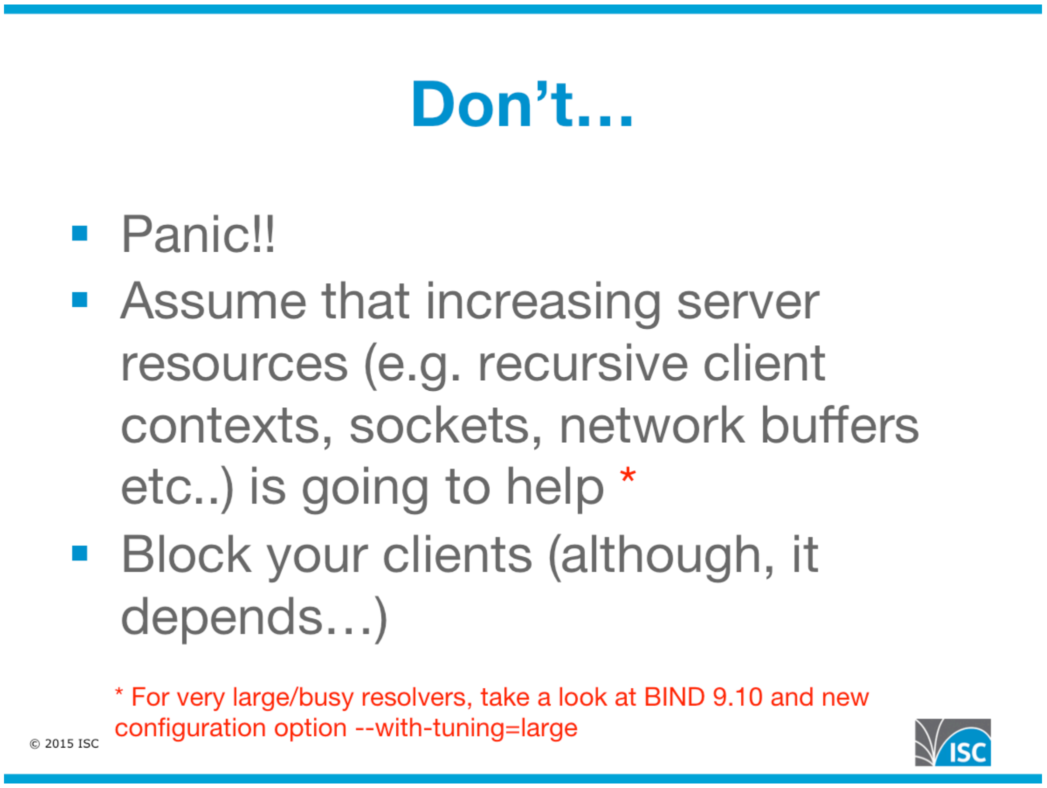
## 3. MITIGATION

HEADLINE: So.. what can you do?

## Mitigation Goals



**HEADLINE: What do you want to achieve?**



## Don’t…

- Panic!!
- Assume that increasing server resources (e.g. recursive client contexts, sockets, network buffers etc..) is going to help *
- Block your clients (although, it depends…)

* For very large/busy resolvers, take a look at BIND 9.10 and new configuration option --with-tuning=large

**HEADLINE: There are several wrong ways to respond to the situation…**

Increasing e.g. recursive clients (BIND) or similar on other DNS server software is not ultimately going to help

Don’t set recursive-clients larger than 3500 – the problem will expand to fill the space made available for it…

*(Aside: DO set recursive clients larger than 1000 on older versions of BIND)*

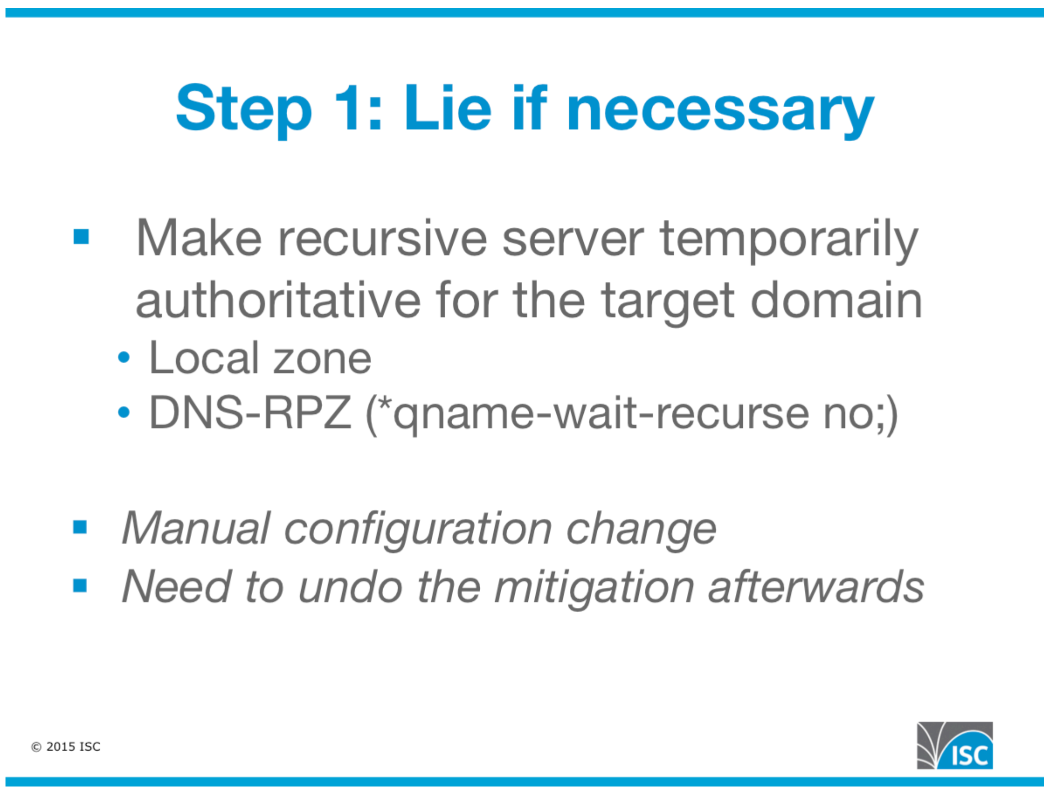
The resources you’re consuming are being used (in the majority) for clients whose queries are anyway doomed to fail

Management of the increased resources has its own overhead

Far better to tackle the root cause of the problem!

If you have many many clients each only contributing a handful of queries to the problem – realistically you can’t block them

You might (if you have just a small number of clients sending the majority of ‘problem’ queries) be able to identify and block the source.



## Step 1: Lie if necessary

- Make recursive server temporarily authoritative for the target domain
  - Local zone
  - DNS-RPZ (*qname-wait-recurse no;)

- *Manual configuration change*
- *Need to undo the mitigation afterwards*

**HEADLINE: This is actually pretty good First Aid – but it's responding to the injury, not preventing it.**

This is usually the first thing that DNS admins try.

The logic goes something like this:

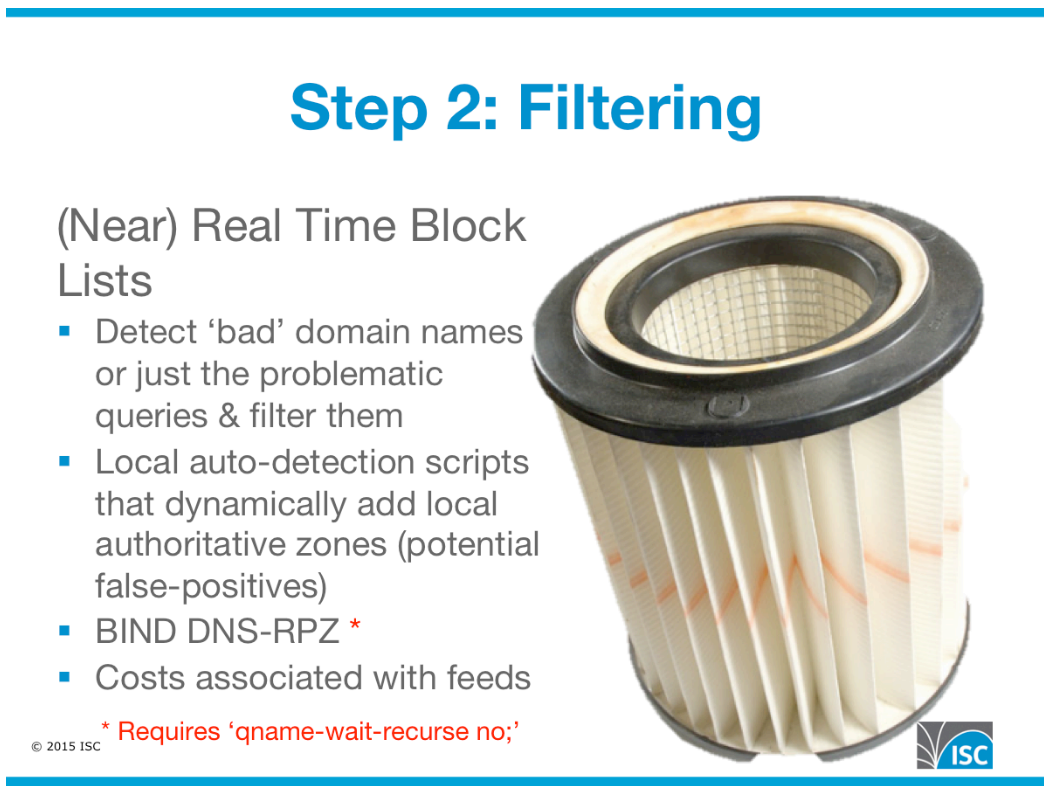
- These queries are for non-existent names
- The authoritative servers (IF they were responding), would send back an NXDOMAIN anyway
- Let's cut out the delays and send back NXDOMAIN immediately
- (…or just drop the queries – can do this with DNS-RPZ)

It's good – but it's far from perfect.

It's manual intervention

It's intervention AFTER the problem has impacted your service

At some point you have to decide whether or not it's same to remove the mitigation and allow legitimate queries through to the non-responding servers once more.



## Step 2: Filtering

(Near) Real Time Block Lists

- Detect 'bad' domain names or just the problematic queries & filter them
- Local auto-detection scripts that dynamically add local authoritative zones (potential false-positives)
- BIND DNS-RPZ *
- Costs associated with feeds

* Requires 'qname-wait-recurse no;'

**HEADLINE: This is more sophisticated than before, but it's still application of First Aid to an injury that has already occurred.**

The next step is usually the 'but we could automate this!' idea. There are two main techniques:

a) Monitoring and scripted inspection of the recursive clients backlog to identify the 'patterns' and add local zones or DNS-RPZ rules to NXDOMAIN or drop the 'problem' client queries

b) Subscription to a service provider that does this for you by means of a slaved response policy zone.

*Note that if you're using DNS-RPZ, either with your own Response Policy Zone, or using a 3rd party feed, you will need to have a version of DNS-RPZ that includes the option to filter on the query name and return an answer BEFORE doing recursion – that is, querying other authoritative servers. It doesn't help to filter after querying the authoritative servers - because that's where named runs into difficulties when those servers fail to respond.*

*The option you need is qname-wait-recurse no;*

*(It's a little more complex than that, as the other rules in the policy zone may still override the suppression of recursion (because recursion would be needed in order for those rules to be executed on the results) – but more details can be*

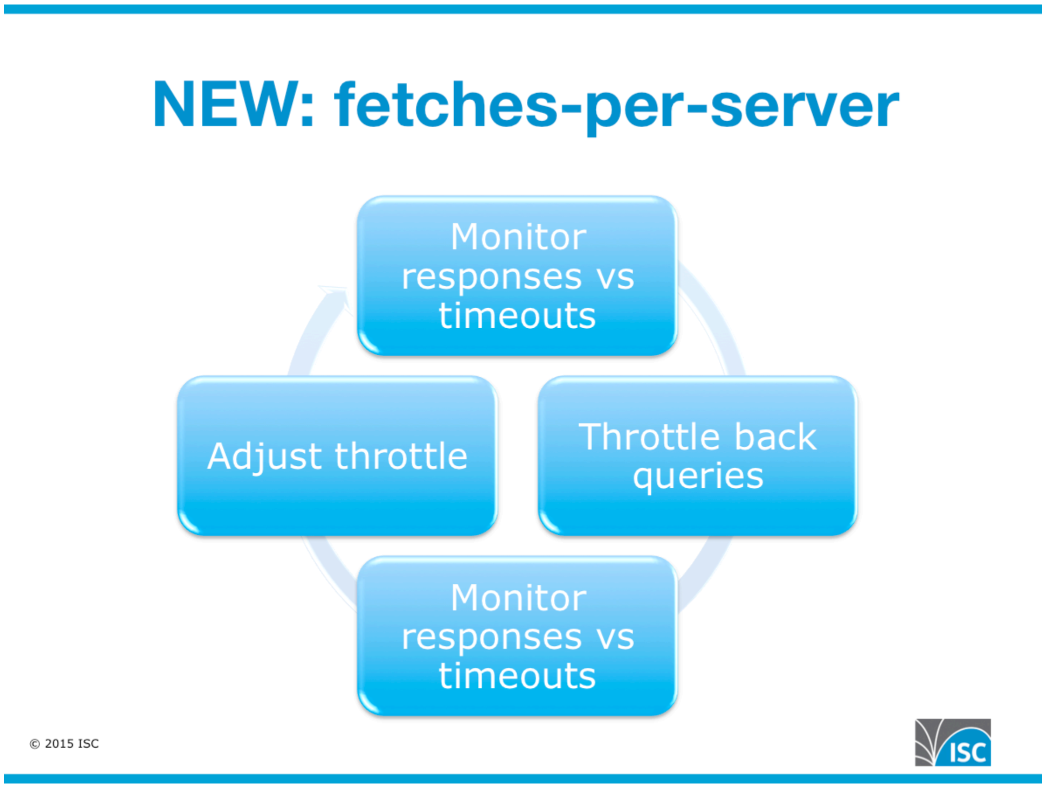
## Step 3: Rate-limiting

**HEADLINE: We want to prevent the injury to the Recursive Server before it happens instead of waiting for the non-responding servers to timeout and cause problems.**

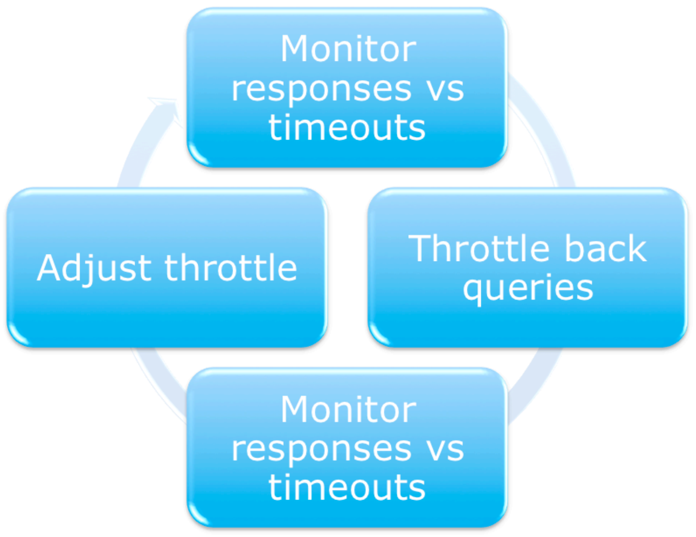
We put our thinking caps on – and realized that the best place to identify that there was a problem, as well as what was causing it, was in BIND itself. And where better also, to apply the appropriate mitigations by automatically rate-limiting the recursive client queries that were causing the problem.

In most instances, once the authoritative servers have stopped responding, we will be sending back a SERVFAIL, but only after waiting for the timeout. Since it is the use of server resources while waiting that causes problems, we can choose not to wait, and to send back the SERVFAIL immediately instead when we detect that that a problem is starting to happen.

So that’s what we did – with extensive testing along the way, both in-house and in production environments offered (to mutual benefit) by some of our support customers who were having real problems.



## NEW: fetches-per-server



**HEADLINE: The rate-limiting works on the basis of monitoring outstanding fetches (this is not the same as recursive clients)**

These new throttles, or client query rate-limiters work on the basis of monitoring and responding to outstanding **'fetches'**. What's a 'fetch'?

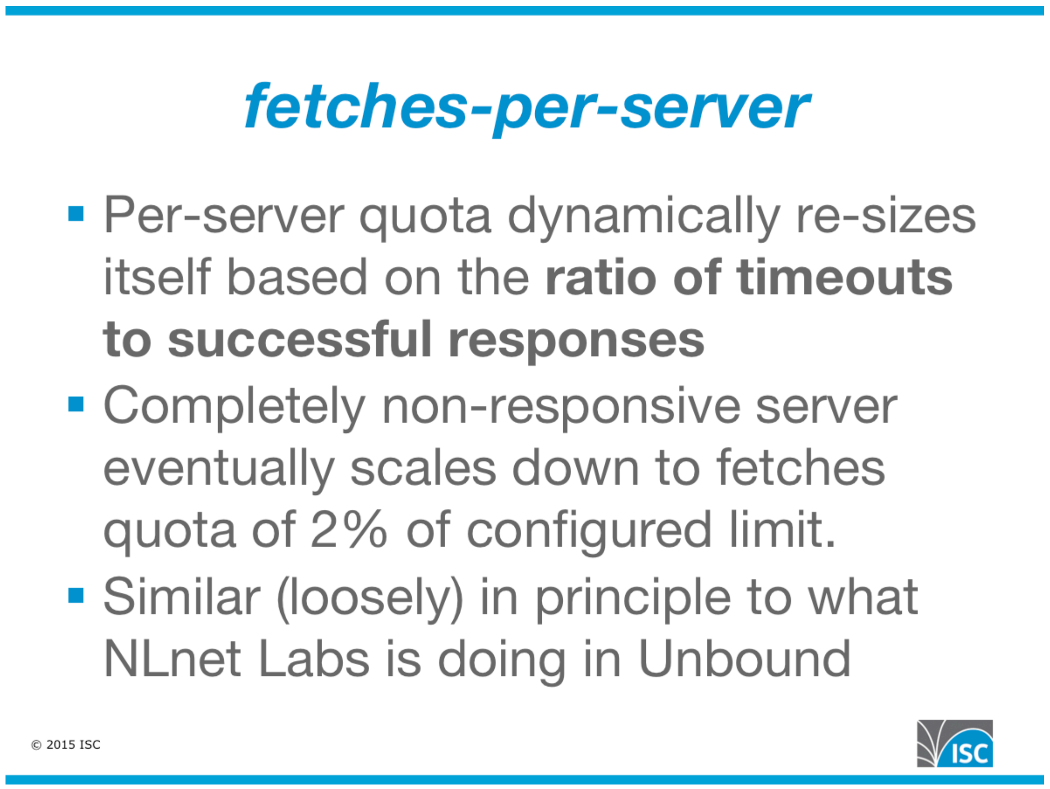
Think of 'fetches' as being the the queries that named has to make – possibly several of them per client query – in order to get the answers into cache so that the client query can be given a response

The throttles are applied when there are too many outstanding fetches

- For the same zone
- To the same authoritative server

This first throttle is the one we see as being the main defense mechanism because it is:

- self-tuning (based on the rate of responses/timeouts)
- works for all outstanding fetches to a non-responding server, even if the



## *fetches-per-server*

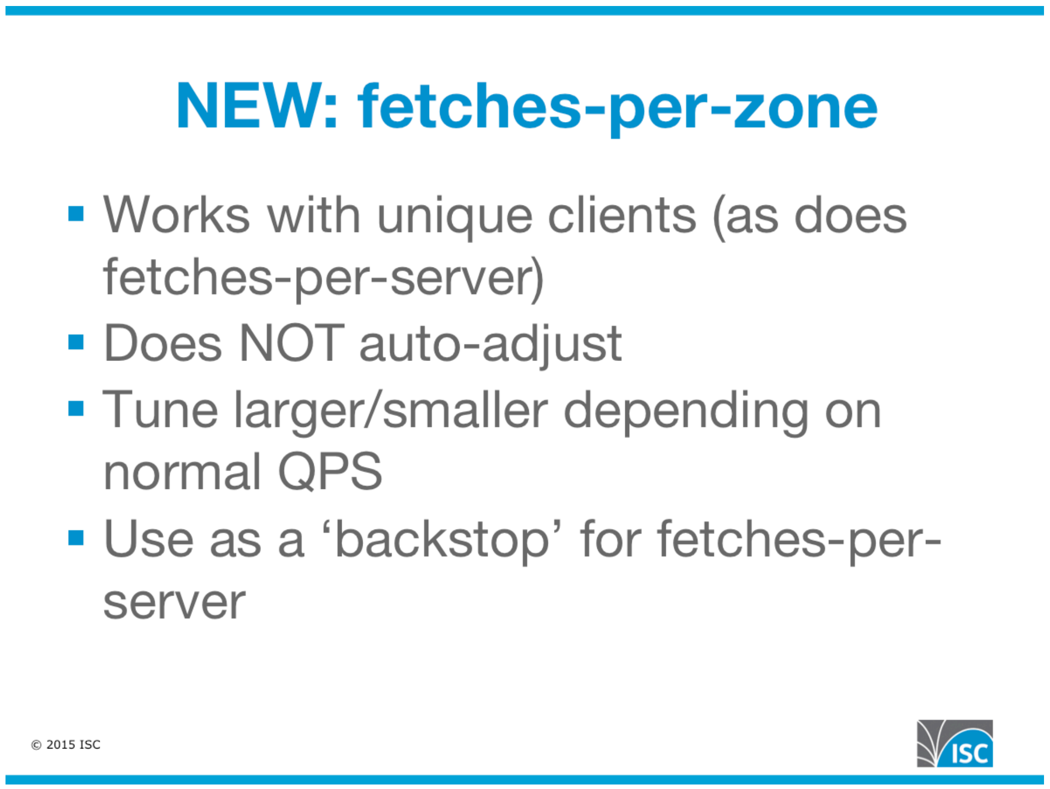
- Per-server quota dynamically re-sizes itself based on the **ratio of timeouts to successful responses**
- Completely non-responsive server eventually scales down to fetches quota of 2% of configured limit.
- Similar (loosely) in principle to what NLnet Labs is doing in Unbound

**HEADLINE: fetches-per-server should be your primary filter in most cases because it is both self-tuning, and applies to all queries to a non-responding server**

The way this works is that you configure the initial threshold for the backlog of outstanding fetches to any one server in your named.conf file.  If the threshold is reached, then named will start rate-limiting fetches to that server (the client receives SERVFAIL if rate-limited)

While rate-limiting, named monitors the responsiveness of the server, and tunes down the quota, depending on the ratio of fetches that timeout, as opposed to those that receive a successful response.  This means that if/when the server starts responding again, the quota is automatically scaled upwards again (and of course, what will also be happening at the same time, is that the server *is* responding, therefore the backlog is anyway reduced, so recovery is going to be very fast.

The scaling algorithm is documented in the Administrator Reference Manual.



## NEW: fetches-per-zone

- Works with unique clients (as does fetches-per-server)
- Does NOT auto-adjust
- Tune larger/smaller depending on normal QPS
- Use as a 'backstop' for fetches-per-server

**HEADLINE: fetches-per-zone is going to be at its most effective when there are many servers for a zone**

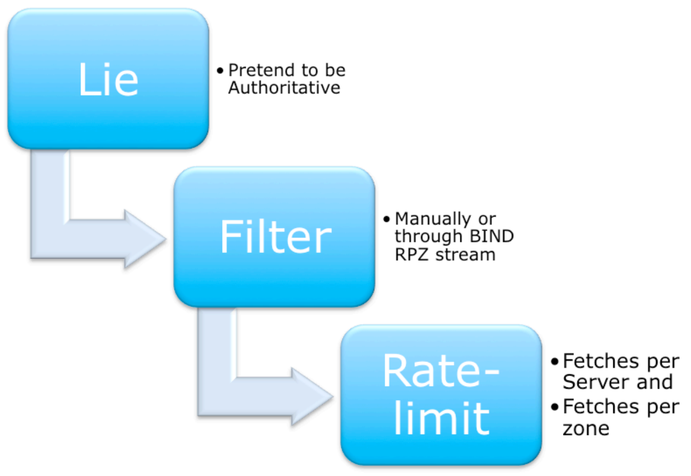
**fetches-per-zone** works best when fetches-per-server threshold isn't being triggered because there are too many of them – but note that it is a static threshold – and one that you would want to set higher or lower depending on the overall volume of queries a server is normally handling.

We recommend 200 as a good starting default for most servers.

And now, a confession! **In the experimental versions of fetches-per-zone, we intended for it to SERVFAIL the clients** but we actually made a coding error and were dropping the client queries instead. This turned out to be an unexpectedly fortunate mistake, as you'll see later when looking at graphic representations of these tuning options in production environments - because it enables us to distinguish between the two of them by virtue of the effect they are having.

It also sparked a discussion of what it's best to do when rate-limiting client

## Mitigation Summary


Lie
• Pretend to be Authoritative
Filter
• Manually or through BIND RPZ stream
Rate-limit
• Fetches per Server and
• Fetches per zone

**HEADLINE: Several techniques available – the first two are reactive, but Recursive Client Rate limiting is proactive and virtually automatic!**

So apart from dealing with the sources of the queries (which won't necessarily solve all your problems anyway), what can you do?

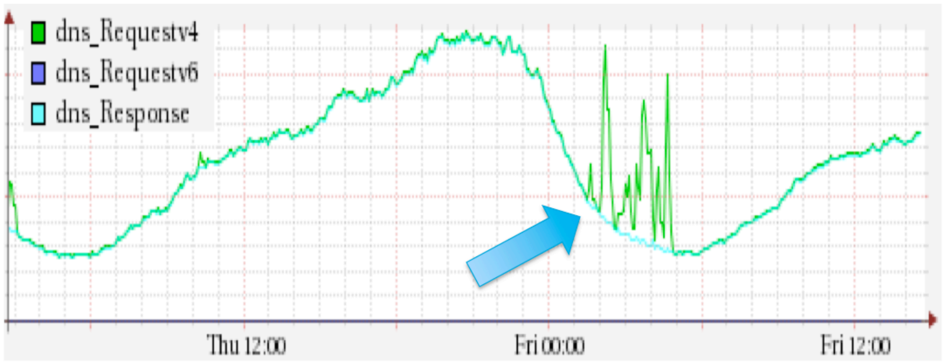
It turns out that there's quite a lot you can do already – and BIND's new recursive client rate limiting will make it even easier to deal with this problem!

## 4. RESULTS FROM LIVE PRODUCTION SYSTEMS

**HEADLINE: Nice new ideas, but do they actually work in practice?**

Well… yes they do!

## fetches-per-zone



Spanish triple-play ADSL carrier & ISP
Roberto Rodriguez Navio, Jazztel Networking Engineering
used with permission

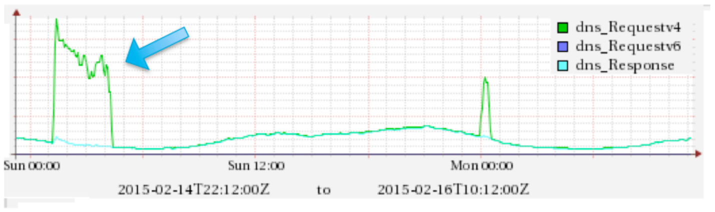
**HEADLINE: fetches-per-zone is limiting the impact of the problem successfully – the peaks of attack traffic don't affect the normal response rates to good clients**

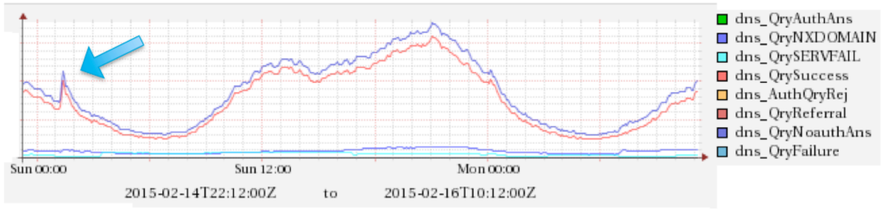
This was the first piece of graphical feedback we received, and we were very excited to see it.

**In the graph, the green line is tracking inbound queries (over IPv4) and the sea blue line is tracking responses sent back.** There is a peak of queries and responses from the start of the day through to the late evening which then dies off rapidly as the the users go to bed – all pretty normal.

But then in the small hours of the morning, **(where the arrow is pointing),** abruptly the attack begins, and we see sharp peaks of DDoS queries that are outside the normal range of expected query traffic for that time of day. **fetches-per-zone** immediately kicks in, and the aqua blue line remains smooth, showing that the attack queries are being dropped, while the genuine client queries are being handled as normal.

## More on fetches per zone





Spanish triple-play ADSL carrier & ISP
Roberto Rodriguez Navio, Jazztel Networking Engineering
used with permission

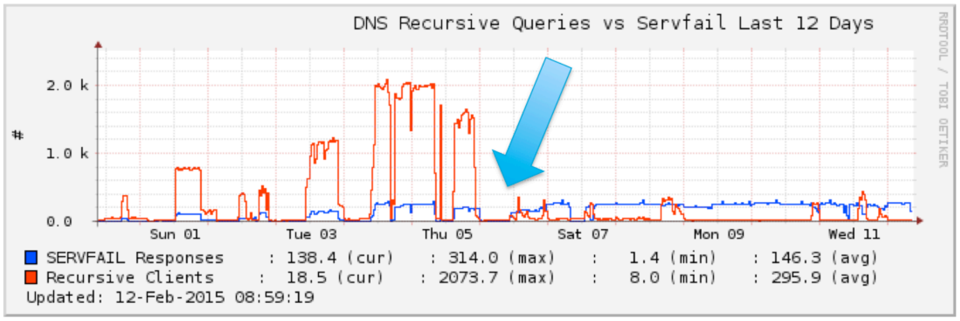
**HEADLINE: An increase in inbound queries with an initial peak in NXDOMAIN responses is one of the signatures of a pseudo-random subdomain attack.**

This is another instance from the same production environment but tracking a few more trends, and with some slightly different scaling.

The TOP GRAPH is tracking the same numbers we saw just before. The BOTTOM, but run over the same time period, is tracking the different response types being returned – **what we're looking at particularly is the NXDOMAIN responses (shown in the dark blue line).**

This slide is interesting because it demonstrates that initially in the attack, there was a small peak in responses – these are NXDOMAIN responses to the unique attack queries that the victim servers have returned before they're overwhelmed and on the top graph, and the aqua line, you can just about make out the small corresponding peak in responses, before the authoritative servers have given up (or implemented Response Rate Limiting against us.

## fetches-per-server



```
■ SERVFAIL Responses    : 138.4 (cur)    : 314.0 (max)    :   1.4 (min)    : 146.3 (avg)
■ Recursive Clients    :  18.5 (cur)   : 2073.7 (max)    :   8.0 (min)    : 295.9 (avg)
Updated: 12-Feb-2015 08:59:19
```

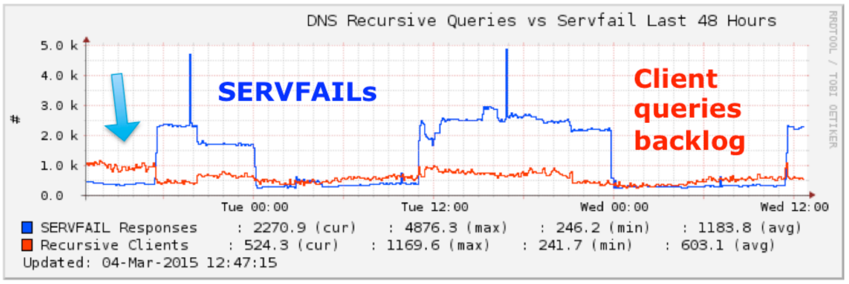
**HEADLINE: fetches-per-server still sends back the SERVFAIL to the client that it would receive anyway, but without consuming server resources**

This monitoring graph is from a different production site (who wished to remain anonymous).

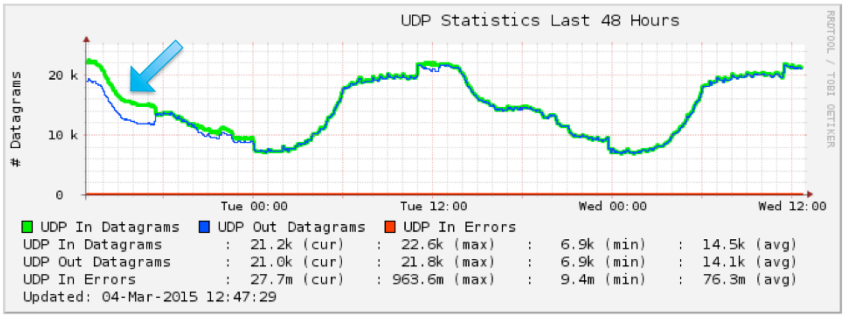
They took a different approach to monitoring – in this case **sampling the backlog of recursive client contexts at intervals**, and mapping it against **SERVFAIL responses being sent to clients**. The reason for tracking the recursive clients backlog is that it's one of the best clues that you have that a recursive server is likely to be running into resource exhaustion problems.

This **shows the point at which fetches-per-server was deployed**. The SERVFAIL responses to the clients continue at a similar rate as before – they would, in any case have been SERVFAIL response to the clients when the fetches to the non-responding authoritative servers timed out, but there's no longer a significant backlog of waiting request consuming server resources because the majority of the SERVFAILs are returned immediately (per fetches-per-server) instead of waiting for timeout.

## per-zone v. per-server



```
■ SERVFAIL Responses    : 2270.9 (cur)   : 4876.3 (max)   : 246.2 (min)   : 1183.8 (avg)
■ Recursive Clients    : 524.3 (cur)   : 1169.6 (max)   : 241.7 (min)   : 603.1 (avg)
Updated: 04-Mar-2015 12:47:15
```



```
■ UDP In Datagrams  ■ UDP Out Datagrams  ■ UDP In Errors
UDP In Datagrams      :   21.2k (cur)   :  22.6k (max)   :   6.9k (min)   :  14.5k (avg)
UDP Out Datagrams     :   21.0k (cur)   :  21.8k (max)   :   6.9k (min)   :  14.1k (avg)
UDP In Errors         :   27.7m (cur)   : 963.6m (max)   :   9.4m (min)   :  76.3m (avg)
Updated: 04-Mar-2015 12:47:29
```

**HEADLINE: Both fetches-per-zone and fetches-per-server have a role to play**

What can happen if you deploy both fetches-per-zone and fetches-per-server?

Here are two different views of the same time period.

**At the start of the period of the attack, it is fetches-per-zone that has been triggered.** This is clearly identified by the gap between inbound queries and outbound responses in the lower graph.

At the same time, (and consistent with fetches-per-zone being in effect) the recursive client queries backlog (red line, top graph) remains constant. That's because this is a static threshold.

You also see a very low rate of SERVFAILs at this point too – that's because of the accidental bug where client queries hitting the fetches-per-zone limit are being dropped versus being SERVFAILed.

Later on (to the right of both graphs) we are seeing the impact of fetches-per-server in the peaks of SERVFAILs (but without corresponding peaks in Recursive Clients – and as confirmed by the logging).

*If you have both deployed, then fetches-per-zone is checked first, but it's possible that queries that are not spilled by fetches-per-zone will subsequently be limited*

## Comparison

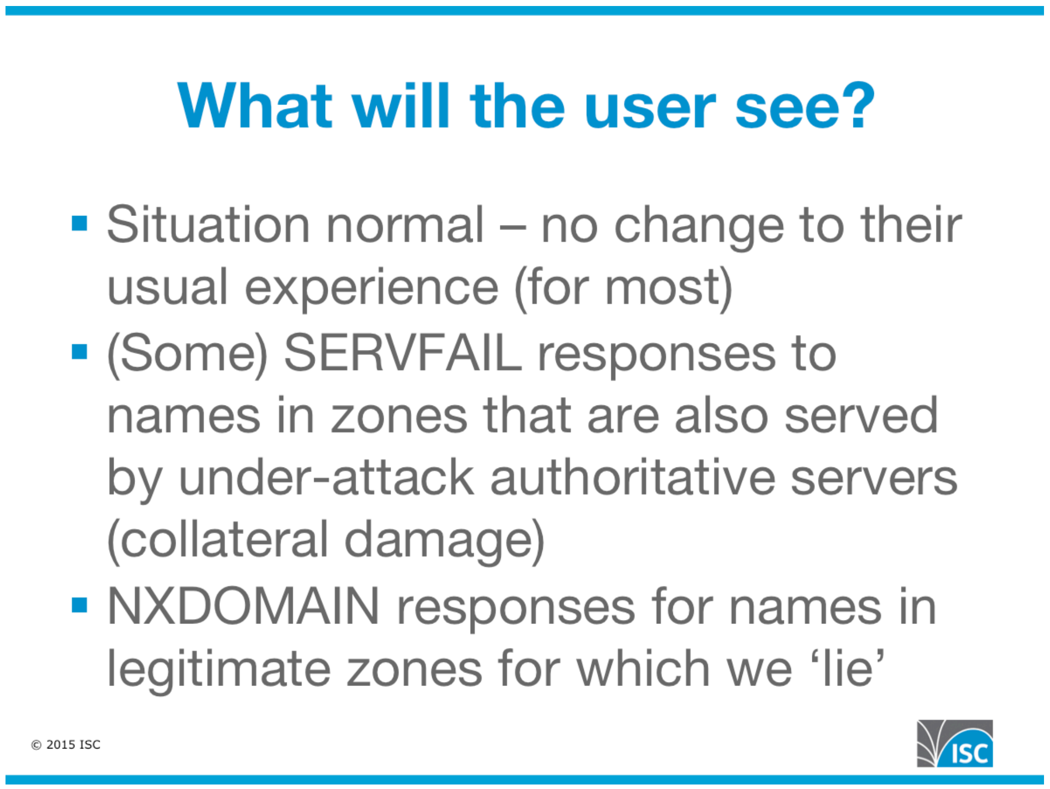
**Fetches Per Server**

- Rate-limits per server
- Impacts queries for all zones served by the same machine
- Dynamically re-sizes based on the ratio of timeouts to successful responses

**Fetches Per Zone**

- Rate-limits per zone
- Manually tuned
- Set to larger value on higher-performance machines

**HEADLINE: Use either or both (we think that deploying both is best).**



## What will the user see?

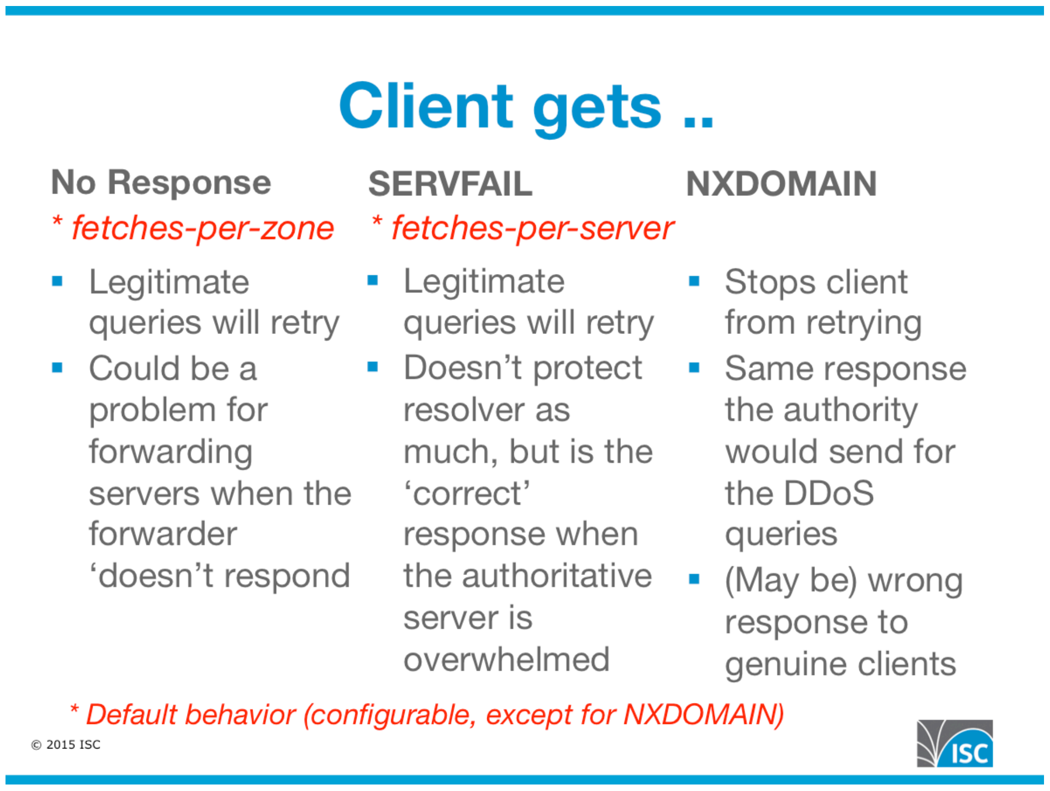
- Situation normal – no change to their usual experience (for most)
- (Some) SERVFAIL responses to names in zones that are also served by under-attack authoritative servers (collateral damage)
- NXDOMAIN responses for names in legitimate zones for which we 'lie'

**HEADLINE: Most end-users will be happily unaware of what is going on.**

But what about our DNS clients?

- We've demonstrated the protection provided to recursive servers by deploying these measures
- We know that by rate limiting the client queries, we're also helping the wider cause by limiting the attack traffic hitting the authoritative servers

Because our server is not starved of resources, it should handle most client queries as usual, but with two exceptions (noted on the slide).



## Client gets ..

| No Response | SERVFAIL | NXDOMAIN |
|---|---|---|
| * *fetches-per-zone* | * *fetches-per-server* | |
| ▪ Legitimate queries will retry<br>▪ Could be a problem for forwarding servers when the forwarder 'doesn't respond | ▪ Legitimate queries will retry<br>▪ Doesn't protect resolver as much, but is the 'correct' response when the authoritative server is overwhelmed | ▪ Stops client from retrying<br>▪ Same response the authority would send for the DDoS queries<br>▪ (May be) wrong response to genuine clients |

* *Default behavior (configurable, except for NXDOMAIN)*

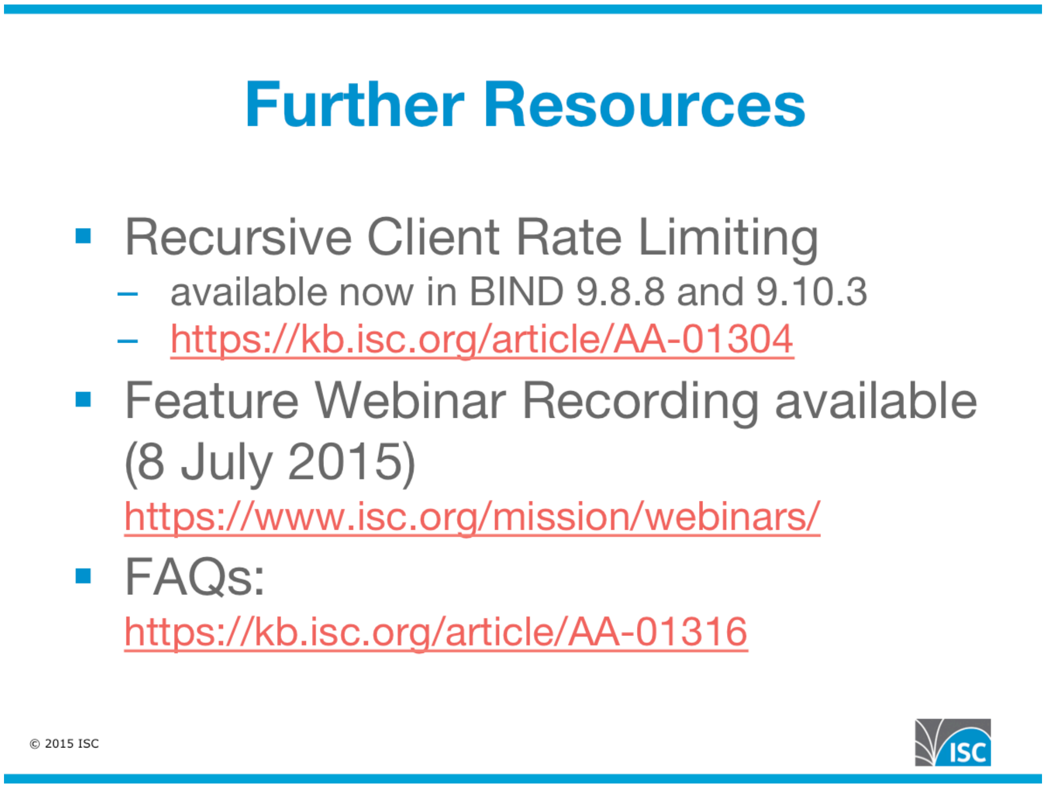
**HEADLINE: You can choose whether to DROP or to send back SERVFAIL for each of fetches-per-zone and fetches-per-server**

So – what IS the right response when rate-limiting client queries?

Fetches-per-zone and fetches-per-server can either drop or send back SERVFAIL. Either would be OK for ordinary clients, but it would be bad to fail to respond to another server that is forwarding, because effectively you'd be passing along to it, the bad impact of the DDoS.

SERVFAIL is the 'correct' response per the DNS protocol – but some administrators may prefer 'drop' as the client behaviour is going to be similar (will likely retry the query) anyway, so why send a response that will just be ignored, and which is using up your network bandwidth.

If you're using a local-authoritative-zone technique, then you'll be sending back NXDOMAIN to all clients (which is the answer the DDoS queries should be receiving anyway) – but unless you also have the legitimate answers for clients in your local zone (very unlikely!), you'll be sending back NXDOMAIN answers for


## Further Resources

- Recursive Client Rate Limiting
  - available now in BIND 9.8.8 and 9.10.3
  - https://kb.isc.org/article/AA-01304
- Feature Webinar Recording available (8 July 2015) https://www.isc.org/mission/webinars/
- FAQs: https://kb.isc.org/article/AA-01316

**HEADLINE: If you need more information, here is where to find it**

So in summary:

We've covered the attack traffic; how to recognize it and what impact it can have on recursive servers.

We've suggested how you can mitigate the attack impact by using local authoritative zones and DNS-RPZ, either manually or via automated scripts or dynamic feeds.

And then finally, we've introduced the new features of BIND that are designed to deal directly with this problem from within named, either instead of, or supplementing mitigation techniques that you already have in place.

## QUESTIONS

info@isc.org, bind-suggest@isc.org,
cathya@isc.org

https://kb.isc.org/article/AA-01304

Any questions?

If you've new ideas that you'd like us to consider, please submit them to bind-suggest@isc.org (we do have several of our own already that we're working on).