The ISP Column
A column on various things Internet

March 2018
Joao Damas,
Geoff Huston

Measuring ATR

The Problem

It's pretty clear that the Internet has a problem. If you want to include Facebook's misuse of personal information in ways that closely resemble unconstrained abandon, then the Internet probably has hundreds of millions of problems! More prosaically, lets confine our view of problems to the Internet Protocol itself, rather than the way it has been abused. One of the major issues here is the ossification of the network due to the constraining actions of various forms of active middleware. The original idea was that the Internet was built upon an end-to-end transport protocol (TCP and UDP) layered above a simple datagram Internet Protocol. The network's active switching elements would only look at the information contained in the IP packet header, and the contents if the "inner" transport packet hear was purely a matter for the two communicating end points.

But that was then, and today is different. Middle boxes that peer inside each and every packet are pervasively deployed. This has reached the point where it's now necessary to think of TCP and UDP as a network protocols rather than host-to-host protocols. Middleware has its own set of expectations of 'acceptable' packet formats and appear to silently drop errant packets. But what is "errant" can be a very narrow set of constraints that impose even greater conditions on protocol actions than the standard specifications would normally permit.

One of the more pressing and persistent problems today is the treatment of fragmented packets. We are seeing a very large number of end-to-end paths that no longer support the transmission of fragmented IP datagrams. This count of damaged paths appears to be getting larger not smaller. As with many network level issues, IPv6 appears to make things worse, not better, and the IPv6 fragmented packet drop rates are substantially greater than comparable IPv4 figures.

There are two middleware boxes that appear to play a major role here. These are firewalls and Network Address Translation units. In both cases these units look inside the transport header to collect the port addresses. The transport header is used by firewalls to associate a packet with an application, and this is then used to determine whether to accept of discard the packet. NATs use the port addresses to increase the address utilisation efficiency, by sharing a common IP address, but differentiating between distinct streams by the use of different port addresses.

The issue with IP packet fragmentation is compounded the use of an extension header to control fragmentation reassembly, inserted between the IP and transport headers. Middleware expecting to see a transport header at a fixed offset within a IPv6 packet are going to have to perform additional work to unravel the chain of extension headers. Some IPv6 devices appear to have been programmed to take a faster path and just drop the packet!

We either can't or don't want to clean up this middleware mess, so it's left to the applications to make use of transport protocols that steer around these network obstacles.

In theory, TCP should be able to avoid this problem. Through conservative choices of the session maximum segment size (MSS), reaction to ICMP Packet Too Big messages (when they are passed through) and as a last resort use of TCP Path MTU Discovery (RFC 4821) it is possible for TCP sessions to avoid wedging on size-related packet drop issues.

But what about UDP? And what about the major client application of UDP, the DNS? Here there is no simple answer.

The original specification of DNS adopted a very conservative position with respect to UDP packet sizes. Only small responses were passed using UDP, and if the response was larger than 512-bytes then the server was meant to truncate the answer and set a flag to show that the response has been truncated.

```
Messages carried by UDP are restricted to 512 bytes (not counting the IP or UDP
headers).  Longer messages are truncated and the TC bit is set in the header.
```
Section 4.2.1, RFC 1035

The DNS client is supposed to interpret this truncated response as a signal to re-query using TCP, and thus avoid the large size UDP packet issues. In any case, this treatment of large DNS responses was a largely esoteric issue right up until the introduction of DNSSEC. While it was possible to trigger large responses, the vast majority of DNS queries elicited responses lower than 512 bytes in size.

The introduction of DNSSEC saw the use of larger responses because of the attached signatures, and the use of this 512-byte maximum UDP response size was seen as an arbitrary limitation that imposed needless delay and inefficiencies on the DNS. An optional field was added to DNS queries, the client's UDP buffer size as part of the EDNS(0) specification. A client could signal its willingness to receive larger packets over UDP by specifying this buffer size.

```
EDNS(0) specifies a way to advertise additional features such as larger response size
capability, which is intended to help avoid truncated UDP responses, which in turn
cause retry over TCP. It therefore provides support for transporting these larger
packet sizes without needing to resort to TCP for transport.
```
Section 4.3, RFC 6891

Interestingly, the default UDP buffer size is not commonly set at 1,280, or 1,500, both of which would both be relatively conservative settings, but at the size proposed in the RFC, namely 4,096. We've managed to swing the pendulum all the way over and introduce a new default setting that signals that its quite acceptable to send large fragmented UDP responses to DNS queries.

```
A good compromise may be the use of an EDNS maximum payload size of 4096 octets as a
starting point.
```
Section 6.2.5, RFC 6891

As we've already noted, the delivery of large fragmented UDP packets is not very reliable over the Internet. What happens when the fragmented packets are silently dropped? From the querier's perspective, a lost large UDP response has no residual signal. All the querier experiences is a timeout waiting for an unforthcoming response. So how do DNS resolvers cope with this? RFC 6891 provides an answer:

```
A requestor MAY choose to implement a fallback to smaller advertised sizes to work
around firewall or other network limitations.  A requestor SHOULD choose to use a
fallback mechanism that begins with a large size, such as 4096.  If that fails, a
fallback around the range of 1280-1410 bytes SHOULD be tried, as it has a reasonable
chance to fit within a single Ethernet frame.  Failing that, a requestor MAY choose a
512-byte packet, which with large answers may cause a TCP retry.
```
Section 6.2.5, RFC 6891

This searching for a size appears to simply take more time in most cases, and it appears that the generic algorithm used by DNS resolvers is:

1. If the client does not receive a response within the locally defined timeout interval then it should re-send the query. These timeout intervals are variously set at 200ms, 370ms, 800ms and 1 second, depending on the resolver implementation.
2. Further timeouts should also trigger queries to any other potential DNS servers.
3. If these queries also experience a timeout then the client should try the queries using an EDNS(0) UDP buffer size of 512 bytes.
4. If the client receives a truncated response, then it should switch over to try TCP.

Even that truncated search for a response involves a lot of time and potentially a lot of packets. If large response packets were uncommon in the DNS, or fragmented packet drop was incredibly rare then maybe this overhead would be acceptable. However, that's not the case. Large responses are not uncommon when using DNSSEC. For example, a query for the signed keys of the .org domain elicits a 1,625 octet response. Inevitably that's a response that is formatted as a fragmented UDP response if the query contains a large EDNS(0) UDP buffer size.

Fragmented packet drop is also depressingly common. Earlier work in September 2017 showed a failure rate of 38% when attempting to deliver fragmented IPv6 UDP packets the DNS recursive resolvers.

As was noted when reporting on this alarmingly high drop rate:

> However, one conclusion looks starkly clear to me from these results. We can't just assume that the DNS as we know it today will just work in an all-IPv6 future Internet. We must make some changes in some parts of the protocol design to get around this current widespread problem of IPv6 Extension Header packet loss in the DNS, assuming that we want to have a DNS at all in this all-IPv6 future Internet.
> http://www.potaroo.net/ispcol/2017-08/xtn-hdrs.html

So how can we get around this drop rate for large DNS responses?

We could move the DNS away from UDP and use TCP instead. That move would certainly make a number of functions a lot easier, including encrypting DNS traffic on the wire as a means of assisting with aspects of personal privacy online as well as accommodating large DNS responses. However, the downside is that TCP imposes a far greater load overhead on servers, and while it is possible to conceive of an all-TCP DNS environment, it is more challenging to understand the economics of such a move and to understand in particular how name publishers and name consumers will share the costs of more expensive name resolution environment.

It we want to continue to UDP where it's feasible, and continue to use TCP only as the 'Plan B' protocol for name resolution, then can we improve the handling of large response in UDP? Specifically, can we make this hybrid approach of using UDP when we can and TCP only when we must faster and more robust?

The ATR Proposal

An approach to address this challenge is that of "Additional Truncated Response" (documented as an Internet Draft: draft-song-atr-large-resp-00, September 2017, by Linjian (Davey) Song of the Beijing Internet Institute, https://tools.ietf.org/html/draft-song-atr-large-resp-00).

The approach described in this draft is simple: If a DNS server provides a response that entails sending fragmented UDP packets, then the server should wait for a 10ms period and also back the original query as a truncated response. If the client receives and reassembles the fragmented UDP response, then the ensuing truncated response will be ignored by the client's DNS resolver as its outstanding query has already been answered. If the fragmented UDP response is dropped by the network, then the truncated response will be received (as it is far smaller), and reception of this truncated response will trigger the client to switch immediately to re-query using TCP. This behaviour is illustrated in Figure 1.

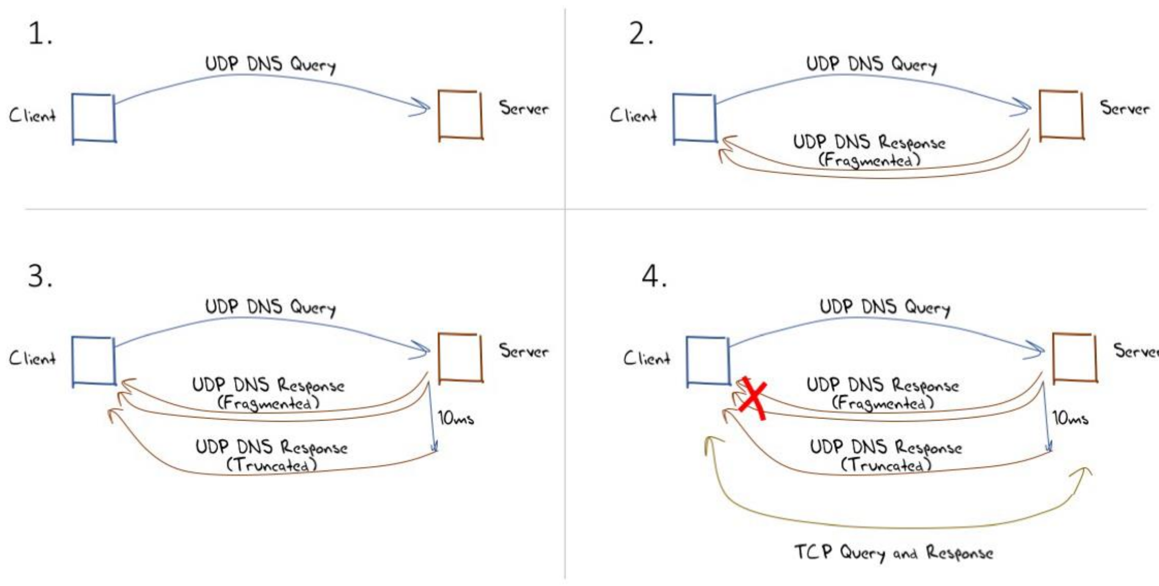
1.
UDP DNS Query
Client
Server
2.
UDP DNS Query
Client
Server
UDP DNS Response (Fragmented)
3.
UDP DNS Query
Client
Server
UDP DNS Response (Fragmented)
10ms
UDP DNS Response (Truncated)
4.
UDP DNS Query
Client
Server
UDP DNS Response (Fragmented)
10ms
UDP DNS Response (Truncated)
TCP Query and Response

Figure 1 – Operation of ATR

Measuring ATR

How well does ATR actually work? We've constructed an experiment to test this proposed mechanism.

The approach used here was that of "glueless delegation" (described by Florian Maury, ANSSI at the May 2015 DNS OARC Workshop: https://indico.dns-oarc.net/event/21/contribution/11/material/slides/0.pdf). When the authoritative server for the 'parent' zone is queried for a name that is defined in a delegated 'child' zone the parent zone server will respond with the name servers of the delegated child zone. But, contrary to conventional server behaviour, the parent zone server will not provide the IP addresses of these child zone name servers. The recursive resolver that is attempting to complete the name resolution task now must suspend this original resolution task and resolve this name server name into an IP address. Only when it has completed this task it will then be able to revert to the original task and pass a query to the child zone servers. What this means is that we can use the DNS itself to test the capabilities of those resolvers that query authoritative servers. If the response that provides the IP address of the child zone name servers is so constructed to exhibit the behaviour being measured, then we know if the resolver was able to receive the response if it subsequently queries the child zone name servers.

Using the glueless delegation technique we constructed 6 tests:

- The first pair of tests used ATR over IPv4 and IPv6. In this case we constructed a fragmented UDP response by appending a NULL Resource Record (RR) into the response as an additional record, generating a response of 1,600 octets. We configured the server to deliberately ignore the offered UDP buffer size (if any) and generated this UDP fragmented response in all cases. The server then queued up a truncated response that was fired off 10ms after the original response.

- The second pair of tests used just the large packet response in both IPv4 and IPv6. In this case the server was configured to send a large fragmented UDP response in all cases, and never generated a truncated response.

- The third pair of tests was just the truncated UDP response in IPv4 and IPv6. Irrespective of the offered UDP buffer size the server echoed the query with an empty response part and the truncated flag set.

This allowed us to measure the extent to which large fragmented UDP responses fail on the paths between our authoritative name servers and the resolvers that pose queries to these servers. It also allows us to measure the extent to which these resolvers are capable of using TCP when given a truncated response. We can also measure the extent to which ATR uses the trailing truncated response.

We performed these tests over 55 million end points, using an online ad distribution network to deliver the test script across the Internet.

Table 1 shows the results from this experiment looking at the behaviour of each IP resolver.

Protocol	Visible Resolvers	Fail Large UDP	Fail TCP	Fail ATR
IPv4	113,087	40%	21%	29%
IPv6	20,878	50%	45%	45%

Table 1 – Failure Rate of Visible Resolvers

Some 40% of the IPv4 resolvers failed to receive the large fragmented UDP response, which is a disturbingly high number. Perhaps even more disturbing is the observation IPv6 failure rate is an astounding 50% of these visible resolvers when a server is sending the resolver a fragmented UDP response.

The TCP failure numbers are not quite as large, but again they are surprisingly high. Some 21% of the IPv4 resolvers were incapable of completing the resolution task if they were forced to use TCP. The IPv6 number is more than double, with 45% of the IPv6 resolvers running into problems when attempting to use TCP.

The ATR approach was seen to assist resolvers, and in IPv4 the ATR loss rate was 29%, indicating that a little over 10% of resolvers that were incapable of receiving a fragmented UDP response were able to switch over the TCP and complete the task. The IPv6 ATR failure rate was 45%, a 5% improvement over the underlying fragmented UDP loss rate.

When looking at the DNS counting the behaviour of resolvers should not be used to infer the impact on users. In the DNS the most heavily used 10,000 resolvers by IP address are used by more than 90% of users. If we would like to understand the impact of ATR on the DNS resolution behaviours as experienced by users, then we need to look at this measurement from the user perspective.

For this user perspective measurement, we count as a "success" if any of the resolvers invoked by the user can complete the DNS resolution process, and "failure" otherwise. These user perspective results are shown in Table 2.

Protocol	Fail Large UDP	Fail TCP	Fail ATR
IPv4	13%	4%	4%
IPv6	21%	8%	6%

Table 2 – Failure Rate of Users

These results indicate that in some 9% of IPv4 cases the use of ATR by the server will improve the speed of resolution of a fragmented UDP response by signalling to the client an immediate switch to TCP to perform a re-query. The IPv6 behaviour would improve the resolution times in 15% of cases.

The residual ATR failure rates appear to be those cases where the DNS resolver lies behind configuration that discards both DNS responses using fragmented UDP and DNS responses using TCP.

Why Use ATR?

The case for ATR certainly looks attractive if the objective is to improve the speed of DNS resolution when passing large DNS responses.

In the majority of cases (some 87% in IPv4, and 79% in IPv6) the additional truncated response is not needed, and the fragmented UDP response will be successfully processed by the client. In those cases, the trailing truncated UDP response is not processed by the client, as the previously processed fragmented UDP response has cleared the outstanding request queue entry for this query.

In those cases where fragmented UDP responses are being filtered or blocked, then the ATR approach will eliminate the client performing a number of timeout and re-query cycles before they query using an effective ENDS(0) UDP buffer size of 512 and generating the truncated response that will trigger TCP. This will eliminate a number of query packets and reduce the resolution time.

ATR is incrementally deployable. ATR does not require any special functionality on the part of the DNS client, as it is exclusively a server-side function. And the decision for a server to use ATR can be made independently of the actions of any other server. It's each server's decision whether or not to use ATR.

Why NOT Use ATR?

It is not all positive news with ATR, and there are some negative factors to also include in the consideration of whether to use ATR.

ATR adds a further UDP packet to a large fragmented DNS response. This assists an attacker using a DNS DDOS attack vector, as the same initial query stream will generate more packets and a larger byte count when the ATR packet is added to the large DNS responses.

If the client resolver is using source port randomisation and is able to receive fragmented UDP responses, then the ATR packet will arrive at the client when there is no outstanding listen request for an incoming UDP packet addressed to this port number. Many resolvers use UDP 'black holing' and silently discard these additional packets, but some will generate an ICMP Port Unreachable message back to the server. In our measurement we found that the level of ICMP message generation was at a level equal to 20% of outbound ATR packets. This should not represent any significant load on the server, as in the context of a DNS server such ICMP port unreachable messages are discarded, but this packet activity does represent a small incremental load placed on the network.

The server's actions in queuing up an additional response to be sent some 10ms after the initial response represents additional load on the server, particularly as a potential memory drain. Implementations should use a maximum delayed response queue size and drop the ATR response when the delayed response queue is full.

The choice of the delay timer is critical. Inspection middleware often reassembled packet fragments in order to ensure that the middleware operation is applied to all packets in the fragmented IP packet sequence. At the destination host the fragmented packet is reassembled at the IP level before being passed to the UDP packet handler. This implies that the unfragmented ATR packet can 'overtake' the fragmented response and generate unnecessary TCP activity. To prevent this the delay needs to be long enough to take into account this potential additional delay in handling of fragmented packets. It also needs to take into account some normal level of packet jitter in the network. But too long a delay would allow for the client to timeout and re-query, negating the incremental benefit of ATR. That implies that the setting of the delay should be "long enough, but not too long".

ATR Assessment

The DNS is a vitally important component of the Internet, and a more robust DNS that can help in providing further security protections for users is a Good Thing. However, there is no free rides here, and among the costs of adding DNSSEC to the DNS is the issue of inflating the size of DNS responses, and larger DNS responses make the DNS slower and less reliable.

ATR does not completely fix the large response issue. If a resolver cannot receive fragmented UDP responses and cannot use TCP to perform DNS queries, then ATR is not going to help. But where there are issues with IP fragment filtering, ATR can make the inevitable shift of the query to TCP a lot faster than it is today. But it does so at a cost of additional packets and additional DNS functionality. If a faster DNS service is your highest priority, then ATR is worth considering.

Disclaimer

Author

Geoff Huston B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

www.potaroo.net