Practical Challenge-Response for DNS*

Rami Al-Dalky
Case Western
Reserve University
rxa271@case.edu

Michael Rabinovich
Case Western
Reserve University
michael.rabinovich@case.edu

Mark Allman
International Computer
Science Institute
mallman@icir.org

ABSTRACT

Authoritative DNS servers are susceptible to being leveraged in denial of service attacks in which the attacker sends DNS queries while masquerading as a victim—and hence causing the DNS server to send the responses to the victim. This reflection off innocent DNS servers hides the attackers identity and often allows the attackers to amplify their traffic by employing small requests to elicit large responses. Several challenge-response techniques have been proposed to establish a requester's identity before sending a full answer. However, none of these are practical in that they do not work in the face of "resolver pools"—or groups of DNS resolvers that work in concert to lookup records in the DNS. In these cases a challenge transmitted to some resolver R_1 may be handled by a resolver R_2, hence leaving an authoritative DNS server wondering whether R_2 is in fact another resolver in the pool or a victim. We offer a practical challenge-response mechanism that uses challenge chains to establish identity in the face of resolver pools. We illustrate that the practical cost of our scheme in terms of added delay is small.

CCS Concepts

•**Networks** → **Network protocol design; Naming and addressing;**

Keywords

DNS; network security; performance

1. INTRODUCTION

The Domain Name System (DNS) is an essential component of the Internet infrastructure that plays a vital role in most Internet transactions. DNS most commonly uses UDP's connectionless transport to facilitate quick transactions. However, this opens the system to abuse as a conduit of denial-of-service (DoS) attacks. In particular, consider an attacker A that spoofs a DNS request from a victim V to some DNS server S. This will cause S to send the response to V achieving both (i) *reflection*—effectively hiding the true source of the attack, A—and (ii) *amplification*—given that DNS responses are generally larger than DNS requests [26, 21]. DNS was the second largest attack vector in DDoS attacks in the first quarter of 2017 [6].

There are multiple entry points into the DNS for an attacker to leverage—from open recursive resolvers to home routers that forward DNS requests. These are *accidental* security problems in that there is wide agreement that most of these should be closed off from all but their own local client populations. Our work concentrates on the fundamental entry point that cannot simply be closed down: authoritative servers. Previous work shows that the natural disparity between request and response sizes provides an attacker that coaxes authoritative servers to help with an attack a significant amount of amplification potential [30, 23]. To defend against DNS amplification attacks, several challenge-response schemes have been proposed (see § 2). These call for an authoritative DNS (ADNS) server to send a challenge to requester before sending the ultimate response. When the requester successfully answers this challenge the ADNS will furnish the response. This mechanism works much like TCP's three-way handshake in that it establishes the requester's identity. Further, the size of the challenge is similar to the size of the request, therefore offering an attacker no advantage if they coax the ADNS to send the challenge to a victim. Unfortunately, prior work [10, 28, 8] shows that modern DNS resolvers (RDNS) are sometimes not single machines, but organized as *pools* of resolvers (RDNS pools). Therefore, without any malice a response to a challenge may come from a different resolver than an ADNS challenged.

In this paper we first show the problem posed by RDNS pools is non-trivial and renders simple challenge-response practically useless. Consequently, we design a challenge-response scheme that works in the presence of RDNS pools. Our basic mechanism is a *challenge chain*—i.e., we challenge an RDNS pool until we get an acceptable response. We encode the challenge history in the challenge itself such that the scheme is stateless at the ADNS. Since a long challenge chain can add delay to transactions we also provide a number of ways an ADNS can *safely* reduce the chain length. We show that our techniques are feasible and their impact on the traffic and DNS resolution process is slight. To our knowledge, ours is the first *practical* challenge-response mechanism that (i) allows ADNS operators to nearly ensure their servers cannot be used as an instrument in an amplification attack, (ii) works in the presence of RDNS pools and (iii) can be employed by ADNS unilaterally, without any modifications to other components of the DNS ecosystem.

2. RELATED WORK

Previous work includes a number of proposals to apply the concept of challenge-response to DNS traffic.

The DNS protocol provides two mechanisms for authoritative servers to redirect querying resolvers instead of answering a query. A "canonical name" (CNAME) response indicates a name that the querying RDNS must resolve to determine the ultimate IP address. E.g., a lookup for "www.foo.com" may return a CNAME for "www-server-14.foo.com" which then must be resolved to an IP address before a client can initiate communication with the host behind the "www.foo.com" name. A second redirection method involves the NS record—which informs the querying resolver to contact a different nameserver for the given record. Previous DNS challenge-response schemes leverage both CNAME records [29] and NS records [16, 17]. The crucial problem with the previous work is that none of the

*This work is supported in part by NSF grants CNS-1237265, CNS-1647126 and CNS-1647145.

mechanisms deal with RDNS pools—which we find to be pervasive (see § 3). While [17] attempts to handle RDNS pools, the mechanism makes an unrealistic assumption that "quiet periods" without any spoofing-based attacks can be leveraged to build (and refresh) an ADNS' understanding of RDNS pools.

DNS-over-TCP [31] has been proposed to solve some of the same issues we address. In this case TCP's three-way handshake becomes the challenge-response mechanism. A limitation of this scheme is that some resolvers cannot use TCP—often due to middlebox blocking—and therefore results in more resolution failures compared to UDP [22, 13, 18].

The "DNS cookie" scheme calls for resolvers and authoritative servers to exchange IP address-based cookies within optional records in DNS messages [14]. This allows all parties to a transaction to gain confidence that the various actors are legitimate parties to the lookup process (as opposed to arbitrary victims of a reflection attack). Further, cookies protect an entire zone, whereas our challenge-response scheme operates on individual names (see § 4). DNS cookies require adoption from both hosts in a transaction and therefore an ADNS must rely on help from clients to thwart amplification attacks. An ADNS could force resolvers that do not support cookies[1] to fallback to TCP, but this would bring DNS-over-TCP limitations mentioned earlier. Alternatively, an ADNS could fall back to challenge-response when cookies are not present in requests.

Finally, another scheme aiming to limit ADNS's culpability in DoS attacks is Response Rate Limiting (RRL) [20], which attempts to discover patterns in arriving queries—e.g., a high-volume stream of requests for a single name from a single IP address—and, when problematic, limit the rate of replies. While RRL can reduce the effectiveness of reflection and amplification attacks, challenge-response offers three key advantages over RRL. First, challenge-response does not require a high-rate query stream and/or a savvy pattern discovery process to detect problematic requests. In particular, an attacker could spread spoofed queries across a large number of ADNS servers to defeat RRL. Second, RRL uses a coarse-grained mechanism to limit an ADNS' contribution to an attack, which can impact legitimate requests. However, challenge-response precisely adjudicates each DNS query and therefore prevents collateral damage to legitimate queries. Finally, while RRL offers a coping mechanism for large attacks, it also opens a new attack vector whereby an attacker can spoof a high-rate stream of requests to try to coax an ADNS to reduce the response rate to a legitimate RDNS server.

3. ASSESSING THE RDNS POOL ISSUE

We first aim to roughly understand the prevalence of RDNS pools. We deploy our own ADNS and scan the IPv4 address space with lookup requests for names within our DNS zone. Our ADNS responds to queries with a CNAME that includes the requesters' IP address. We then detect RDNS pools when a request to resolve the CNAME arrives from a different IP address.[2] We conducted our scan via PlanetLab nodes from March 20 to April 2, 2016. We received answers from 10.3 million open DNS resolvers across 20K ASes and 221 countries. As we know from previous work, most of these are simple forwarders—generally cheap home routers—which blindly forward requests on to an actual recursive resolver (RDNS) rather than conducting recursive lookup themselves [28]. The 10.3 million open resolvers forwarded the lookups to roughly 85K RDNSes across 13K ASes and 210 countries, which then sent queries to our ADNS. We detect RDNS pools in use for 24% of the open forwarders. Further, we find over 900 organizations deploying RDNS pools in some fashion. While we do not know whether the open resolver population gives us a biased view of the relative prevalence of RDNS pools, we do know that the absolute number of pools represents millions of homes and this is significant enough to indicate that any challenge-response scheme must be able to work correctly in the presence of resolver pools.

[1]We find 0.15% of the requests arriving at one instance of A root over 24 hours in April 2018 contain cookies [5].

[2]We do not claim to detect all RDNS pools, but only those that spread requests to our ADNS across multiple hosts within the pool. However, these are precisely the kinds of pools that are problematic for challenge-response schemes.

4. THE BASIC MECHANISM

Previous simple challenge-response mechanisms are left in a quandary when faced with RDNS pools. Consider challenge-response mechanisms faced with two situations:

Case 1: A benign RDNS pool sends a query from R_1 to ADNS Z for a name N. Z then responds with a challenge C_N. The RDNS pool then sends Z a query for C_N from a second resolver R_2.

Case 2: An attacker sends ADNS Z a request for N from its own IP address A. Z responds with a challenge C_N. The attacker then queries Z for C_N from a spoofed victim address V.

The only difference in these two cases is intent—which Z cannot determine. Thus, in this situation, (i) Z has established that the original requests for N—from R_1 and A—are not spoofed because they received C_N and responded; (ii) if Z decides to return the ultimate response at this point, this process roughly halves the attack's amplification factor because A must now send two requests instead of one to coax Z into sending a response to a victim; and (iii) Z is left unable to determine whether the query for C_N comes from a benign resolver in an RDNS pool (i.e., R_2 in case 1) or from an attacker spoofing a victim's address (i.e., V in case 2). We address the threat posed by (iii) through the following two strategies.

Challenge Chains: Our first strategy involves multiple rounds of challenge-response when the responses return from unexpected IP addresses—which we refer to as a *challenge chain*. For instance, the two cases above would continue with a second challenge, C_{N2}, sent to R_2 and V, respectively. Further, each challenge includes all IP addresses observed in the chain thus far. Therefore, in case 1, C_N would contain R_1 and C_{N2} would contain R_1 and R_2. We can then determine that the challenge response comes from a valid, non-spoofed address if its source address is among any of the IP addresses encoded in the challenge. Indeed, since all these addresses have been challenged they all must have explicitly played a part in the arrival of the current challenge response. Although this approach ensures that the challenge chain will eventually terminate for any finite-size RDNS pool, large pools may lead to long chains and high delays in obtaining the ultimate answer, hence our second strategy.

Traffic Nullification: Our second strategy stems from the original problem we are addressing: preventing amplification. Since each challenge requires the attacker to send another query, the amount of amplification an attacker can obtain is inversely proportional to the length of the challenge chain. There is a point when an attacker has been forced to answer enough challenges such that these effectively *nullify* any amplification possible by coaxing an ADNS to send the ultimate—and potentially large—DNS response to a victim. Therefore, an ADNS can terminate a challenge chain when amplification is no longer possible even though the challenge responses have all been from unexpected IP addresses. In calculating nullification, one can count the attacker's outbound traffic due to queries or bidirectional traffic due to both queries and replies (challenges). There are arguments to be made for either accounting method; to be conservative, we only account for attacker's outbound traffic throughout the experiments in this paper.

We have designed and implemented a challenge-response mecha-

nism that utilizes both challenge chains and nullification. We use CNAME resource records as a conduit for challenges. While our changes are consistent with the operational DNS ecosystem, they deviate from the letter of the DNS specifications. For instance, RFC 2181 [15] prohibits returning CNAMEs at the apex of a zone. However, we note that these sorts of deviations already happen. As an example, CNAME, NS, MX and A records simultaneously exist at the apex of the *adpop-1.com* zone. Further, we configured our own ADNS to return challenges per our mechanism and we were able to retrieve correct responses via a local *bind* [19] installation and eleven public DNS resolvers,[3] suggesting that our approach is not problematic for recursive resolvers in the wild. However, a 2014 Cloudflare trial shows that CNAME records at a zone apex cause issues in corner cases [12]. While our experiments show no current issues, operators wishing to avoid using a CNAME at the apex of a zone could instead aim to keep the records at the apex small to avoid being used in amplification attacks.

Another practical issue is ensuring a CNAME can encode a sufficiently long challenge chain to nullify large DNS responses in the face of large RDNS pools. We find that seven challenges are sufficient to nullify nearly all DNS responses (see § 5.3). Consider forming a challenge from (i) the 99^{th} percentile query string length (51 bytes) from our ICSI dataset (see § 5), (ii) an HMAC to ensure challenge integrity as we discuss below (five bytes) and (iii) seven IP addresses encoded as eight character strings of hexadecimal digits (56 bytes). This will result in a challenge with a length of 112 bytes (without considering separators)—or, less than half the allowable query string size (255 bytes). Therefore, even if this precise encoding is not used and/or a longer challenge chain is necessary, the query string limit does not present a practical barrier for including enough IP addresses to meet the nullification threshold—even in the face of large RDNS pools.

Our approach includes two additional facets:

Statelessness: Our design requires no ADNS state. We encode all required state to implement both challenge chains and nullification in the challenges themselves. We include a list of RDNSes observed thus far in the current chain. This implicitly conveys the length of the chain so the ADNS can determine when nullification happens. Further, each challenge includes a short HMAC[4] that covers the challenge and is salted with a secret key.[5] This prevents an attacker from fabricating a valid challenge response. By making the entire mechanism stateless we not only keep the ADNS complexity to a minimum, but also do not open the ADNS to state holding attacks.

Replay Prevention: Attackers may gain an advantage by reusing challenges. Consider an attacker that coaxes an ADNS to issue enough challenges to terminate the chain via nullification (e.g., using a botnet). Now, the attacker possesses a challenge which will trigger a large response and can be repeatedly presented to the ADNS from any spoofed address. Using only the state in this challenge, the ADNS will conclude nullification has happened and send the ultimate—likely large—DNS response to the victim even though the victim's address is not in the list of valid addresses in the challenge itself. Unfortunately, in a stateless system replay cannot be eliminated because the ADNS does not know how many times a challenge response has arrived. To mitigate the replay attack without accruing state about each transaction, we change the secret key used to calculate the HMAC every three seconds.[6] A challenge response with an expired HMAC triggers a fresh challenge chain. We additionally retain the two previous secret keys to validate incoming requests. This process confines the replay vulnerability to a small time window and therefore mitigates the possible impact.

An ADNS can further limit the impact of replay at the expense of relaxing the requirement to operate stateless. Specifically, an ADNS can track the challenges issued within the time window we sketch above. At this point, replay can be prevented by answering a query with a given challenge only once, and starting a fresh challenge chain for any replayed query. Such a policy would lengthen transactions that experience loss and therefore must rely on retransmitting a query. Therefore, an ADNS may tune the number of times a specific challenge will be answered to balance the additional delay imposed on retransmissions with the susceptibility to replay attacks. Note, since the blocking would be enacted individually for each specific challenge, this approach does not expose the ADNS to some general DoS attack whereby an attacker can coax the ADNS into wholesale blocking of a legitimate RDNS (pool). We also note that the attack signal generated by challenge replay is quite strong: once a specific challenge has been received more than a few times the purpose is clearly an attack and the IP addresses encoded in the challenge—but not the potentially spoofed source IP address from which the challenge arrived—are complicit and can be blocked by the ADNS or an associated IDS/IPS.

Finally, we note that our scheme does not directly address reflection attacks. We do not prevent a DNS request with a spoofed source of V from triggering a DNS response to V (a challenge in our case). Rather, our scheme aims to remove the amplification advantage by making it more costly for an attacker to coax an ADNS into sending a large number of bytes to a victim than it would be for the attacker to send those bytes itself (using spoofing to hide its identity). This means that DNSSEC poses a problem for our scheme. Since responses protected by DNSSEC are large relative to the queries, an attacker could use the amplification available in a DNSSEC-protected challenge to gain an advantage. As currently defined, DNSSEC and our challenge-response scheme are incompatible and an operator must choose between the two schemes. We discuss DNSSEC further in a longer version of this paper [7].

4.1 Performance Impact

We now turn to assessing the cost of our challenge-response mechanism via trace-driven simulation. Our analysis is based on one week—September 15–21, 2016—of passive observation at the shared access link of Case Connection Zone (CCZ), an experimental network that connects roughly 100 homes to the Internet via bi-directional 1 Gbps fiber-optic links [1]. The *Bro* traffic monitor [4] records all DNS transactions between the homes and the CCZ's RDNS servers, as well as summaries for all TCP connections. We aim to study the added delay TCP connections initiated by CCZ users would experience if all authoritative DNS servers involved in every DNS lookup adopted our challenge-response mechanism.[7]

[3]Google, Verisign, AT&T, OpenDNS, Quad9, UUNET, Level3, DNSWatch, Norton ConnectSafe, DYN, and SafeDNS.

[4]We use a 5 byte HMAC in our implementation as it need not be strong due to frequent key rotations.

[5]Since the contents of the challenge need only be understood by the ADNS that issues the challenge, the precise encoding of the challenges is immaterial to our findings. However, as an example, a challenge CNAME for "www.foo.com" may look like "c0011c5ic001ca5e.hmac.www.foo.com", where the first 16 characters encode two IP addresses, the second label is the protecting HMAC and the remainder is the actual query string. Note, exceeding the maximum label length results in adding a label.

[6]Note, for ADNSes with multiple instances—e.g., anycast replicas—the secret key will need to be aligned via synchronized clocks and keys generated from a pseudorandom function with a common starting point.

[7]While our challenge-response mechanism certainly could impact non-TCP transactions, we focus on TCP in this initial assessment because we find that TCP carries 95% of the traffic volume that

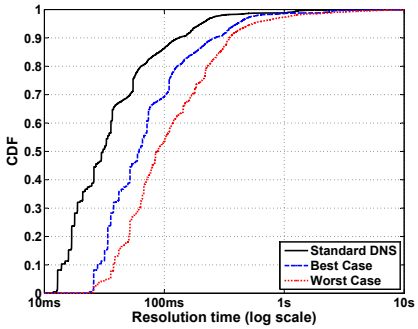

Figure 1: Response time distribution for cache misses

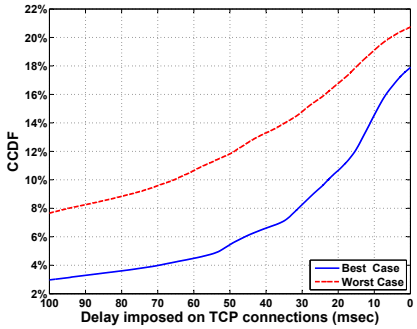

Figure 2: Distribution of delay imposed on TCP connections

The DNS logs contain 8.8M DNS transactions. We exclude 750K transactions that contain no resource records in the response and 50K transactions related to DNSSEC, which cannot benefit from our scheme. The remaining 8M DNS transactions resolve 137K unique hostnames. Our TCP logs contain summaries of 13.3M connections. We exclude 4.4M connections that likely stem from scanning or backscatter as they do not complete TCP's three-way handshake to establish a connection.[8] We couple each TCP connection with the most recent non-expired DNS response that includes the remote IP address used in the TCP connection to the given local CCZ host. Of the 8.9M valid TCP connections, we find 7.8M TCP connections leverage 3.7M DNS lookups, which is similar to our previous findings in this environment [27].

Since our vantage point is between the homes and the RDNS, we do not observe the iterative resolution process the RDNS performs for each query. Therefore, to understand the delays in performing each component of the lookup process we conducted an active measurement study whereby we iteratively looked up each hostname found in our logs via *dig* from a host within the CCZ—i.e., the same vantage point where we collect passive data—from October 10–12, 2016. These measurements yield: (i) the iterative steps involved in looking up each hostname, (ii) the time required to perform each step of the lookup process, (iii) the size of all responses and (iv) the TTL of all involved resource records.

We then simulate the RDNS behavior, including all iterative steps involved in each lookup, based on the workload observed from the actual CCZ clients. Our trace-driven simulation technique is similar to that used in our previous work [27]. To bound the estimated impact of our mechanism, we simulate a best and worst case, as follows. The best case is when a single RDNS handles all lookups (i.e., there is no RDNS pool). In this case, our challenge-response scheme turns each request to an ADNS server into two requests: one for the desired resource record and one to respond to the ADNS' challenge. The worst case involves simulating an RDNS that is implemented as an infinite pool of resolvers such that challenges are never answered from any of the previously observed RDNS servers and the chain is ultimately terminated via nullification. In both cases we simulate an RDNS cache with entries expiring after the TTL obtained in our active measurements. In the simulation of the worst case, we assume the pool uses a shared DNS cache—a la Google's public DNS platform [2] (although there is evidence that Google's cache is not global). In both the best and worst cases, the simulated resolution time for each iterative step is the number of required rounds of challenge-responses required for nullification multiplied by the resolution time obtained during our active measurements.

Figure 1 shows the duration of DNS resolutions with a baseline that does not include challenge-response, as well as our challenge-response scheme, under the best- and worst-case scenarios we sketch above. We include resolutions that are both (i) subsequently used by a TCP connection (3.7M) and (ii) not fully resolvable from the RDNS' cache (i.e., with a cache miss for at least one iterative step). With the hit rate of 54% we observe in our simulation, this encompasses 1.7M DNS transactions. Both challenge-response simulations show longer lookup times than the base case—which we clearly expected because the challenges add extra transactions to complete a lookup. The best case shows an increase of roughly 31 msec at the median point (a 2x increase over current DNS). The worst case adds 57 msec to current DNS at the median point.

We next aim to understand how meaningful this increase is to TCP connections that ultimately use DNS results. In this initial work we focus on individual TCP connections. In cases where TCP connections are dependent on one another—which is non-trivial to determine—any added delays imposed by challenge-response may accumulate across connections. Given our results we believe this effect is likely low; however, addressing this case will be the subject of future work. We calculate the delay imposed on each TCP connection as: $D = DNS_{start} + DNS_{lookup} - TCP_{start}$, where DNS_{start} is the time the client started the DNS lookup found in our passive logs, DNS_{lookup} is the simulated time required for the DNS resolution and TCP_{start} is the time from our passive logs when the client initiates the given TCP connection. When D is less than zero, the simulated DNS transaction concludes before the TCP connection begins, hence the TCP connection is not delayed. Our simulations begin with an empty cache, leading to overestimating the delay and therefore a conservative impact assessment.

Figure 2 shows the distribution of D—the amount each TCP connection is delayed in our simulations. We show a CCDF with reversed x-axis: for a given delay value on x-axis, the y-axis shows the fraction of connections with higher delay. We truncated the plot such that TCP connections that are not delayed by our scheme are not included. The plot shows that in the worst case 79% of the TCP connections are not delayed by our scheme and only 10% of the connections are delayed by more than 65 msec. The best case is strictly better with just over 4% of the connections being delayed by more than 65 msec. Our results illustrate that clients often make DNS requests well in advance of actually using the DNS results to establish TCP connections. This prefetching behavior has been noted in previous work (e.g., [11]). We conclude that the increased DNS lookup time we introduce in our scheme is modest in terms of its overall impact on end-user communication and therefore does not present a hindrance to deployment. However, there are cases where the challenge-response mechanism impedes subsequent transactions. Hence, in the next section we introduce extensions to our basic scheme to safely reduce the DNS lookup costs.

5. COPING WITH RDNS POOLS

As we discuss in § 3, our experiments show that 24% of open resolvers leverage RDNS pools. To understand the length of the

leverages DNS responses.

[8]The connection failure rate agrees with previous work (e.g., [9]).

Chain length	Basic scheme (%)	Explicit RDNS pool tracking (%)	Implicit RDNS pool tracking (%)
1	76.2	99.7	92.7
2	81.0	99.9	98.9
3	84.1	99.9	∼100
4	86.4	∼100	
5	88.2		
6	90.4		
7	91.5		
8+	100		

Table 1: Cumulative distribution of challenge chain lengths

challenge chains these pools cause, we extend the probing of open DNS resolvers we sketch in § 3. In particular, we implemented our challenge-response mechanism in the MaraDNS server [3]. Then, from several PlanetLab nodes we scan the entire Internet with lookups for names within our own zone—which is served by our challenge-response-enabled ADNS. Our implementation uses a single DNS label for the challenge and given our encoding scheme, this is creates a limit on the chain length of seven challenges—which is less than DNS' query string would ideally support (see § 4). If our ADNS has not received a challenge response from a previously seen IP address after seven challenges, we return the requested DNS resource record.[9] Our goal is to assess the possible size of challenge chains and therefore we did not use nullification in this experiment.

The first two columns in Table 1 show the cumulative distribution of the challenge chain lengths we find in our Internet scan. We find that 76% of the queries require a single challenge as the response to the challenge arrives from the same RDNS as the original request. On the other hand, 24% of the queries involve 5.3K RDNS pools (we describe the method of determining pools in § 5.1). We find that nearly 8.5% of the queries require more than seven challenges—indicating these queries come from large RDNS pools. Further, roughly 98% of these long chains come from Google RDNSes. Google is known to have a complex DNS infrastructure [2] and therefore this finding is unsurprising. While Google is responsible for the longest challenge chains, we find only 14% of the 5.3K RDNS pools to involve only Google resolvers. We find over 900 organizations are leveraging RDNS pools in some fashion.

An obvious way ADNSes can cope with RDNS pools is to set policy limits on the challenge-response process to trade protection for imposing lower delays on users' DNS lookups. Examples of such policies might include: (i) limiting the time the process can last, (ii) opting to reduce possible amplification rather than eliminating it via a lower threshold for nullification (e.g., the resolver may pay for only two-thirds of the response bytes with requests bytes) or (iii) imposing limits based on previous behavior or knowledge (e.g., requiring a successful challenge chain to be completed only once per resolver per time period). These policy knobs give ADNS operators latitude to shape the tradeoff between user experience and being co-opted into an attack. The rest of this section sketches ways to reduce challenge-response delays without weakening protection.

5.1 Developing Understanding of RDNS Pools

Our first extension calls for relaxing the requirement that our scheme be completely stateless for the ADNSes. Rather, we investigate the efficacy of allowing each ADNS to use the challenge chains to develop an understanding of RDNS pools. This knowledge is then used when future requests arrive to determine whether the challenge response is valid. For instance, the arrival of a challenge response encoding a challenge chain with two resolvers—R_1 and R_2—indicates a legitimate RDNS pool that includes both resolvers. Indeed, this query signifies that both R_1 and R_2 have acted as expected upon receiving their respective challenges, and therefore neither R_1 nor R_2 has been spoofed. Consider the case when the ADNS subsequently receives a request from R_1 and so issues a challenge to R_1. If the response to the challenge comes from R_2, the ADNS can accept the challenge immediately without continuing the chain because the ADNS has previously established that R_2 is in the same RDNS pool as R_1.[10] As the ADNS interacts with RDNS infrastructure, the ADNS will build an increasingly accurate understanding of RDNS pools and therefore will be able to reduce the challenge chain length for normal DNS lookups. When a challenge response arrives and carries a proper HMAC, all IP addresses encoded in the query have been verified to belong to the same RDNS pool, and we refer to the enclosed chain as a *validated chain*.

A first challenge is how to group resolvers into RDNS pools. We have explored several variants of the process but all perform similarly (and well, per results below). Therefore, we focus on a simple algorithm that groups RDNSes from validated chains with a common first resolver into an RDNS pool. E.g., consider encountering the following sets of RDNSes in three validated challenge chains: (R_a,R_b), (R_a,R_c) and (R_b,R_d). The ADNS would conclude there are two RDNS pools: (i) (R_a,R_b,R_c) based on the first two chains that both begin with R_a, and (ii) (R_b,R_d) based on the third chain that starts with R_b. Since both pools share R_b it would seem natural to consider all four resolvers to be within the same pool. We instead use a strategy that keys on the first resolver in the chain because this makes finding the pool candidate for a given chain efficient.[11]

To assess the benefits of this enhancement, we run a trace-driven simulation using the logs of challenge chains from our Internet scan collected at our ADNS. Initially, the ADNS has no understanding of RDNS pools and therefore challenges the requesters using our basic mechanism from § 4. As these challenge chains proceed, we update our understanding of the RDNS pools based on the validated challenge responses. This RDNS pool understanding is then used to shorten future challenge chains. We randomly shuffle the queries from our original scan to factor out the specific order of the queries in our results. The simulation run we report is consistent with four additional runs we conducted.

The cumulative distribution of the resulting challenge lengths is shown in the third column of Table 1. Virtually all (99.7%) of the queries require only a single challenge when leveraging the previously developed understanding of RDNS pools. This leaves 0.3% of the queries requiring more than one challenge, in contrast to the basic scheme (§ 4) where 24% of the queries need more than one challenge. We thus conclude that an ADNS that retains the understanding of RDNS pools observed through the challenge-response process can significantly reduce the fraction of queries that experience more than one challenge and the attendant delay.

The reduction in challenge chain length is gained at the cost of developing and retaining state about RDNS pools. We now use a back-of-the-envelope calculation to understand the rough magnitude of the state requirement. First, an analysis of requests to the authoritative servers for the `com` and `net` TLDs shows that 90% of the requests come from 40K resolvers [25]. We make a pessimistic assumption that each of these 40K resolvers represents an RDNS pool.

[9]Seven challenges is enough to nullify over 97% of the responses that arrived at ICSI's DNS resolvers on September 26, 2016.

[10]Similar to the DNS cookies and all prior challenge-response approaches, we assume an off-path attacker that is unable to sniff ADNS responses to the victim.

[11]Since we do not have a realistic traffic pattern we are unable to soundly explore expiring information about RDNS pools. Therefore, our results are the best case in that in a realistic setting information would be expunged periodically. We implemented a crude expiration threshold of 1–72 hours and find our results are affected by only a few percent. Therefore, we do not believe the simulation results we show are dramatically better than could be achieved in reality.

Further, [8] shows that 85% of RDNS pools contain at most ten resolvers and that the largest pool contains 317 resolvers. Therefore, assume that 85% of the 40K pools will require 26 bytes of state: IP address, timestamp, and a Bloom filter large enough to track ten IP addresses. The remaining 15% of the pools will consume 346 bytes of state to hold the same state as the small pools plus an additional Bloom filter that can hold up to 320 IP addresses. Therefore, the approximate amount of state a busy ADNS would use to store an understanding of RDNS pools in less than 3 MB—which is not an onerous requirement in modern servers.

5.2 Implicit Understanding of RDNS Pools

An alternative to explicitly retaining an understanding of RDNS pools is to implicitly assume that if two RDNS servers fall within the same address block they are likely working together. As an example, we assume that if two RDNS servers are located within the same /24 they are working together and therefore receiving a challenge response from any IP address within any of the /24 blocks that we challenged in the preceding rounds is acceptable. In the last column of Table 1, our scan data shows that making this assumption increases the instances where we need only a single challenge from 76% (the base case) to almost 93%. Further, 98.9% of the cases are handled after two challenges when using implicit RDNS pools.

The downside of treating all hosts within an IP address block as equivalents is that one host within the block can easily attack another host within the block. The larger the block the more problematic this issue becomes because there are simply more hosts to attack and likely a more complicated topology within the block. For instance, treating all RDNS servers within the same AS as equivalents would make the potential for problems much larger than using /24 blocks.

We argue that at the level of /24 blocks our policy is relatively safe. First, arbitrary attacks are not possible since an attacker needs control of a host within the victim's /24. Second, if an attacker controls a host within a /24 then there are at most 255 victims that can be attacked. Third, since a /24 is the smallest block that can reliably be routed in the Internet we know that hosts within a /24 are likely to be geographically and topologically close and hence likely *share fate*.[12] That is, if an attacker controls 1.1.1.1 and attacks 1.1.1.2 then the attack is likely to impact the Internet connectivity of the attacker as well as the victim. Further, this sort of shared-fate attack is possible regardless of whether we leverage implicit RDNS pools. Indeed, since 1.1.1.1 and 1.1.1.2 share fate, an attacker that controls 1.1.1.1 will be able to impact 1.1.1.2 without spoofing the host but by simply saturating its own Internet link. Thus, using implicit RDNS pools does not increase vulnerability in this respect. Therefore, we conclude that treating all hosts within a /24 as equivalents offers tangible benefit and little cost. We briefly investigate shorter prefixes but find roughly the same benefits (within 1% of /24) until we get to /16 blocks. The /16 prefixes handle roughly 99% of the cases with a single challenge (6% more than the /24 case), bringing implicit pools roughly on par with explicit pool tracking—but, broadens the attack surface to 64K hosts.

5.3 Probabilistic Responses

As with the other extensions in this section, our goal is to safely reduce the challenge chain length in the face of RDNS pools. Our next extension *randomly* ends the challenge chain by returning the ultimate answer before either meeting the nullification threshold or receiving an acceptable challenge response. Once an attacker knows that an ADNS is returning the ultimate answer randomly, a game of sorts ensues whereby at some point the attacker spoofs a challenge response such that the ADNS will send the next response—either the ultimate answer or another challenge—to the victim. The attacker wins the game when the ADNS randomly chooses to return the ultimate (large) response at the same point the attacker chooses to spoof the source address. In this case, since the nullification threshold has not been met, the attacker has sent fewer bytes to the ADNS than the ADNS sends to the victim. On the other hand, the attacker loses the game when the ADNS randomly chooses to return a (small) challenge at the same point the attacker chooses to spoof the source address. In this case, the attacker has sent the ADNS more traffic across multiple requests and challenge-responses than the ADNS sends to the victim in the form of a single challenge. In both cases the challenge chain ends because the victim will not respond. Our approach is to set the probability of returning the ultimate answer such that the attacker's wins are offset by losses and therefore there is no amplification benefit over the large number of transactions required to mount a DoS attack.

n	Nullification (%)	Probabilistic Responses (%)
1	44.5	72.3
2	72.8	92.2
3	92.0	97.8
4	96.6	98.9
5	97.8	99.2
6	98.5	99.4
7	∼100	∼100

Table 2: Cumulative distribution of challenge chains with probabilistic responses

Consider the case when an attacker is simulating an RDNS pool that uses distinct IP addresses to respond to challenges from an ADNS. Let n be the number of challenges sent by the ADNS and answered by the attacker in a given challenge chain. Also, let S_q and S_a be the size of the original DNS query and the ultimate answer, respectively. Including the original query, the attacker has sent a total of $T = S_q \cdot (n+1)$ bytes.[13] At this point, assume the attacker has decided to spoof a challenge response such that the ADNS will send the subsequent response to the victim of the attack. The ADNS will send the ultimate answer (S_a bytes) with probability P and another challenge (S_q bytes) with probability $1-P$. To balance wins and losses, we must choose a P that satisfies equation 1, which we subsequently re-arrange (after substituting T with its expression above) as a calculation of P in equation 2.

$$T = P \cdot S_a + (1-P) \cdot S_q \tag{1}$$

$$P = \frac{S_q \cdot n}{S_a - S_q} \tag{2}$$

Note that the ADNS always sends at least one challenge as $P = 0$ when no challenges have been completed (i.e., when $n = 0$).

To evaluate the probabilistic approach we use trace-driven simulation that leverages all DNS queries from ICSI's recursive resolvers to remote ADNSes on September 26, 2016. Our simulations make the worst case assumption that ICSI uses infinite RDNS pools that never query from the same IP address twice. Table 2 shows the cumulative percentage of queries that complete after a given challenge chain length, for both basic nullification (column 2) and our probabilistic approach (column 3). The table shows that the prob-

[12] A natural extension is for an ADNS to leverage routing table information to more precisely determine address blocks. While this allows for more resolvers to be treated as equivalents, it also requires the ADNS to keep additional state. Ultimately, the granularity each ADNS uses is a policy decision.

[13] In our implementation, the size of the challenges are larger than the original DNS request as they slowly grow with the length of the challenge chain. We performed an analysis taking this in account and the results are very close to those using the simplifying assumption that all requests are of size S_q bytes.

abilistic approach significantly increases the fraction of queries that complete after only short challenge chains. The probabilistic scheme increases the percentage of queries that require only a single challenge by nearly 28% (from 44.5% to 72.3%) and at most two challenges by over 19% (from 72.8% to 92.2%). While the probabilistic approach provides further benefits on longer chains, the benefits become smaller as most queries are terminated via nullification before reaching these chain lengths. E.g., 92% of the queries are answered after at most three challenges via nullification, leaving relatively few queries to optimize as the chain length grows.

5.4 Request Padding

The extensions we discuss above are designed to allow an ADNS to safely and unilaterally shorten challenge chains. In this section we take a different approach and also involve the resolvers in the process. In particular, we note that by padding requests a resolver can reduce the time required to nullify a response. As an illustrative example, consider an idealized resolver that (somehow) knows a request of 30 bytes to a given ADNS will yield a 100 byte response. By including 70 bytes of padding in the request (e.g., using the EDNS(0) Padding Option [24]), the resolver will meet the nullification threshold without any challenges—and hence negate the delay imposed by challenge-response on the resolver's users.[14] Of course, the size of a response is generally unknown to the resolver and therefore we offer two padding variants to cope:

Explicit Padding: Using this variant, the ADNS returns the size of the ultimate response—in an OPT record—in the first challenge. A resolver answering from a different IP address than sent the original request can meet the nullification test in the first challenge response by padding the request accordingly. This scheme therefore bounds to penalty imposed by the challenge-response to one RTT.

Implicit Padding: The second variant calls for resolvers to pad requests without concrete knowledge of the size of the ultimate response. This removes the need for explicit coordination between resolvers and ADNSes, but also makes the benefit more nebulous since we can no longer bound the length of the challenge chains as we can with the explicit approach. Still, the resolvers do not have to operate completely blind. One approach is for resolvers to simply pad requests to the size of the current MTU to "pay for" as large responses as possible. Alternatively, resolvers can use the history in their logs to gain a rough understanding of how much padding to use. For instance, analyzing one day of DNS transactions between ICSI's resolvers and the queried ADNSes shows that padding requests by 70 bytes would immediately nullify 50% of the responses. Each resolver could adopt a padding policy that strikes a balance between the amount of padding and the fraction of transactions the resolver wishes to nullify without being challenged.

A final note is that the resolvers (and hence their users) incurring the most penalty from the basic challenge-response mechanism are those within complex resolver pools. We expect that since these resolvers already have highly customized behavior they will be in a better position to implement padding than singular local resolvers.

6. CONCLUSION

DNS has been widely exploited in DDoS attacks largely because (i) the connectionless nature allows attackers to readily spoof the query source and (ii) the disparity between request and response sizes offers attackers a way to amplify their attacks. We offer a practical challenge-response mechanism that ADNSes can use to establish the identity of requesters and nearly ensure the ADNS is not coaxed into being an instrument in DNS amplification attacks. To our knowledge, our mechanism is the first such defense measure that (i) requires no changes to the DNS protocol or resolvers and clients and (ii) is capable of working in the face of RDNS pools—a common way to organize DNS resolver infrastructure. Further, we show that the basic cost of our scheme—increasing delay for legitimate lookups—can be managed and is ultimately small in most cases. We therefore conclude that our mechanism adds a valuable and practical tool to ADNS operators' arsenal.

[14]Unlike DNS cookies [14], which *require* resolvers' participation to mitigate amplification attacks, request padding is an optional optimization that does not affect the defense efficacy of our scheme.

7. REFERENCES

[1] Case Connection Zone. http://www.caseconnectionzone.org/.

[2] Google Public DNS. https://developers.google.com/speed/public-dns/docs/performance.

[3] MaraDNS. http://maradns.samiam.org/.

[4] The Bro Network Security Monitor. https://www.bro.org/.

[5] A-Root DITL Data, submitted to DNS-OARC by Verisign, Apr. 2018. https://www.dns-oarc.net/oarc/data/ditl/2018.

[6] Akamai. [State of the Internet] / Security. Q1 2017.

[7] R. Al-Dalky, M. Rabinovich, and M. Allman. Practical Challenge-Response for DNS. Technical Report TR-18-001, International Computer Science Institute, June 2018.

[8] R. Al-Dalky and K. Schomp. Characterization of Collaborative Resolution in Recursive DNS Resolvers. In *Passive and Active Measurement Conference*, 2018. To appear.

[9] M. Allman, V. Paxson, and J. Terrell. A Brief History of Scanning. In *ACM/USENIX Internet Measurement Conference*, Oct. 2007.

[10] H. A. Alzoubi, M. Rabinovich, and O. Spatscheck. The Anatomy of LDNS Clusters: Findings and Implications for Web Content Delivery. In *Proc. of the Intl. Conf on World Wide Web*, 2013.

[11] T. Callahan, M. Allman, and M. Rabinovich. On Modern DNS Behavior and Properties. *ACM Computer Comm. Review*, July 2013.

[12] Cloudflare. Introducing CNAME Flattening: RFC-Compliant CNAMEs at a Domain's Root, Apr. 2014. https://blog.cloudflare.com/introducing-cname-flattening-rfc-compliant-cnames-at-a-domains-root/.

[13] J. Dickinson, S. Dickinson, R. Bellis, A. Mankin, and D. Wessels. DNS Transport Over TCP - Implementation Requirements. RFC 7766, Mar. 2016.

[14] D. Eastlake 3rd and M. Andrews. Domain Name System (DNS) Cookies. RFC 7873, May 2016.

[15] R. Elz and R. Bush. Clarifications To the DNS Specification. RFC 2181, July 1997.

[16] F. Guo, J. Chen, and T.-c. Chiueh. Spoof Detection for Preventing DoS Attacks Against DNS Servers. In *IEEE Intl. Conf. on Distributed Computing Systems*, 2006.

[17] A. Herzberg and H. Shulman. DNS Authentication as a Service: Preventing Amplification Attacks. In *Proc. of the Annual Computer Security Applications Conference*, 2014.

[18] G. Huston. A question of DNS protocols. http://www.potaroo.net/ispcol/2013-09/dnstcp.html, 2013.

[19] Internet Systems Consortium. BIND. https://www.isc.org/downloads/bind/.

[20] ISC. A Quick Introduction to Response Rate Limiting. ISC Knowledge Base, AA-01000, June 2013.

[21] M. KÃijhrer, T. Hupperich, C. Rossow, and T. Holz. Exit from Hell? Reducing the Impact of Amplification DDoS Attacks. In *USENIX Security Symposium*, 2014.

[22] W. Lian, E. Rescorla, H. Shacham, and S. Savage. Measuring the Practical Impact of DNSSEC Deployment. In *USENIX Security Symposium*, 2013.

[23] D. C. MacFarland, C. A. Shue, and A. J. Kalafut. Characterizing Optimal DNS Amplification Attacks and Effective Mitigation. In *Passive and Active Measurement Conference*, 2015.

[24] A. Mayrhofer. The EDNS(0) Padding Option. RFC 7830, May 2016.

[25] E. Osterweil, D. McPherson, S. DiBenedetto, C. Papadopoulos, and D. Massey. Behavior of DNS' Top Talkers, a .com/.net View. In *Passive and Active Measurement Conference*, Mar. 2012.

[26] V. Paxson. An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks. *ACM Computer Comm. Review*, 2001.

[27] K. Schomp, M. Allman, and M. Rabinovich. DNS Resolvers Considered Harmful. In *ACM SIGCOMM HotNets*, Oct. 2014.

[28] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman. On Measuring the Client-Side DNS Infrastructure. In *ACM Internet Measurement Conference*, Oct. 2013.

[29] R. Tzakikario, D. Touitou, and G. Pazi. DNS Anti-Spoofing Using UDP, November 2009. US Patent 7,620,733.

[30] R. van Rijswijk-Deij, A. Sperotto, and A. Pras. DNSSEC and Its Potential for DDoS Attacks: A Comprehensive Measurement Study. In *ACM Internet Measurement Conference*, 2014.

[31] L. Zhu, Z. Hu, J. Heidemann, D. Wessels, A. Mankin, and N. Somaiya. Connection-Oriented DNS to Improve Privacy and Security. In *IEEE Symposium on Security and Privacy*, 2015.

Appendix: Aggressive Probabilistic Responses

As we show in § 5.3, an ADNS can randomly shorten challenge chains to benefit legitimate clients while providing no aggregate benefit to an attacker. We do this via carefully setting the probability of shortening the challenge chain. We now consider an extension to the scheme in § 5.3 that makes the probability setting more aggressive, while still ensuring an attacker garners no advantage across the large number of transactions required for a DoS attack. We describe this extension here because we find it to be too complex for the resulting benefits in the general case. E.g., the aggressive scheme we describe here increases the fraction of queries that complete with up to two rounds of challenge-responses by 3.5% (from 92.2 to 95.7%), and provides a less than 1% boost for queries requiring more rounds. However, as we will show, the more aggressive scheme provides larger benefits for large records and in the face of large RDNS pools.

The basic formulation in § 5.3 considers a game whereby an attacker wins by coaxing an ADNS to send a large record to a victim before nullification and loses when the ADNS sends a small challenge to a victim. In setting the probability of returning the ultimate (large) answer we carefully consider the traffic *within* a challenge chain such that an attacker's wins are offset by losses. However, we now take into account another way the attacker loses the game. Consider the case when n challenges have been answered by the attacker using addresses the attacker controls—i.e., no spoofing a victim to this point—and the ADNS randomly decides to return the ultimate answer. This terminates the challenge chain, representing a loss for the attacker in that the attacker sent $n+1$ queries to the ADNS and yet the ADNS sent *no traffic* to a victim. Our observation is that this challenge chain ends, but we can use the traffic volume the attacker expended in its futile effort to offset more frequent transmission of the actual DNS answers. Essentially, we can increase the traffic bank T and as a result P will also increase.

To get an intuition for the aggressive scheme, consider a DoS attack whereby an attacker controls a large number of hosts and never uses the same IP address twice within a given challenge chain. Initially the attacker sends Q_0 queries to an ADNS. These trigger Q_0 challenges from the ADNS. When the Q_0 challenge responses arrive we can leverage the basic probabilistic strategy to calculate P_1 via equation 2. The ADNS then returns the ultimate answer in response to $P_1 \cdot Q_0$ of the challenge responses—which also terminates the challenge chains. This leaves $Q_1 = (1-P_1) \cdot Q_0$ chains to further challenge. The ADNS will next receive challenge responses indicating two rounds of challenge response have been completed (i.e., $n=2$). We expect Q_1 of these challenge responses to arrive at the ADNS. On average, for each round 2 query, there must have been one round 1 query to which we responded with a challenge, but these only represent a $1-P_1$ fraction of all round 1 queries. The remainder of the round 1 queries were given the ultimate answer and hence did not issue queries i in round 2. Thus, there must have been on average $1/(1-P_1)$ total round 1 queries, costing the attacker $\frac{2S_q}{1-P_1}$ bytes. Therefore, for each round 2 query we can bank $S_q + \frac{2S_q}{1-P_1}$ bytes, which is greater than the $3S_q$ bytes the basic scheme uses. Further, we use equation 1 to compute the probability P_2 of answering the current challenge responses with the ultimate DNS answer.

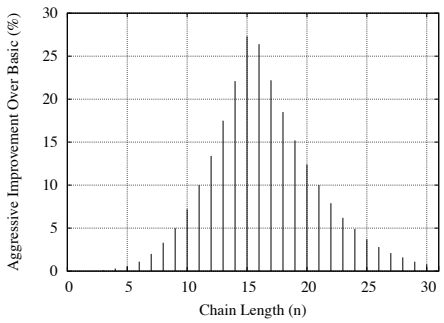

Figure 3: Improvement in cumulative probability of aggressive scheme compared over basic scheme

More generally, if we receive a challenge response that encodes a chain of n challenges, we can assume that on average, there must have been $1/1-P_{n-1}$ queries in round $n-1$, $1/\{(1-P_{n-2})(1-P_{n-1})\}$ queries in round $n-2$, and so on. This gives rise to the following iterative computation of probability of returning the ultimate answer, P_n:

$$P_n = \begin{cases} \frac{S_q}{S_a - S_q} & \text{if } n = 1 \\ \frac{S_q}{S_a - S_q}\Big[\frac{2}{(1-P_1)\cdot(1-P_2)\cdot\ldots\cdot(1-P_{n-1})} + \sum_{i=2}^{n-1} \frac{1}{(1-P_i)\cdot\ldots\cdot(1-P_{i+1})}\Big] & \text{if } n \geq 2 \end{cases} \quad (3)$$

While not of significant benefit in general, our aggressive approach provides relatively more help when dealing with large records and large RDNS pools compared to the basic variant. As an example, we use the DNS transaction with the biggest amplification factor from DNS transactions initiated by ICSI's RDNS servers on September 26, 2016. In this case, $S_q = 33$ bytes and $S_a = 3587$ bytes, hence nullification happens at $n = 109$. Assuming an infinite RDNS pool without resolver reuse, the cumulative probability of returning the actual answer reaches 100% by $n = 16$ for the aggressive scheme and $n = 33$ for the basic scheme. Figure 3 shows the increase in the cumulative probability of returning an answer within a given number of rounds n of the aggressive scheme over the basic scheme. The figure shows the two schemes perform similarly (within 1%) until $n = 6$. At this point the aggressive scheme shows growing improvement and reaches the greatest improvement at $n = 15$ rounds, when the aggressive scheme completes 27% more chains than the basic scheme. The aggressive scheme out paces the basic scheme by at least 10% for $n = [11, 21]$.

Our conclusion is that while the aggressive scheme can shorten the resolution process for some legitimate lookups, there are several caveats: (i) the probability calculation is more complex (as we move from a simple closed-form equation to an iterative calculation); (ii) the aggressive scheme only provides significant benefit for lookups of large records (the benefit of the aggressive scheme is at most 3.5% over all lookups in our ICSI log); and (iii) the benefits of the aggressive scheme only manifest when large RDNS pools are in use (the benefit is $< 1\%$ for challenge chains that end within six rounds). Therefore we conclude that at present the complexity is likely only worthwhile in specialized situations.