Post-Quantum Signatures in DNSSEC via Request-Based Fragmentation

Jason Goertzen
University of Waterloo
jgoertze@uwaterloo.ca

Douglas Stebila
University of Waterloo
dstebila@uwaterloo.ca

Abstract

The Domain Name System Security Extensions (DNSSEC) provide authentication of DNS responses using digital signatures. DNS operates primarily over UDP, which leads to several constraints: notably, DNS packets should be at most 1232 bytes long to avoid problems during transmission. Larger DNS responses would either need to be fragmented into several UDP responses or the request would need to be repeated over TCP, neither of which is sufficiently reliable in today's DNS ecosystem. While RSA or elliptic curve digital signatures are sufficiently small to avoid this problem, even for DNSSEC packets containing both a public key and a signature, this problem is unavoidable when considering the larger sizes of post-quantum schemes.

We propose ARRF, a method of fragmenting DNS resource records at the application layer (rather than the transport layer) that is *request-based*, meaning the initial response contains a truncated fragment and then the requester sends follow-up requests for the remaining fragments. Using request-based fragmentation avoids problems identified for several previously proposed—and rejected—application-level DNS fragmentation techniques. We implement our approach and evaluate its performance in a simulated network when used for the three post-quantum digital signature schemes selected by NIST for standardization (Falcon, Dilithium, and SPHINCS+) at the 128-bit security level. Our experiments show that our request-based fragmentation approach provides substantially lower resolution times compared to standard DNS over UDP with TCP fallback, for all the tested post-quantum algorithms, and with less data transmitted in the case of both Falcon and Dilithium. Furthermore, our request-based fragmentation design can be implemented relatively easily: our implementation is in fact a small daemon that can sit in front of a DNS name server or resolver to fragment/reassemble transparently. As well, our request-based application-level fragmentation over UDP may avoid problems that arise on poorly configured network devices with other approaches for handling large DNS responses.

1 Introduction

The Domain Name System (DNS) is a mission critical service for the Internet. DNS is responsible for translating human-readable domain names into machine-understandable IP addresses and is used by billions of devices daily. Ensuring that these translations are correct and not forged is critical to prevent users from being directed to malicious servers instead of their intended destination. The Domain Name System Security Extensions (DNSSEC) [15] provide data integrity by using digital signatures. DNSSEC ensures that the received DNS message is indeed from a server authorized to respond to the query, and that the message has not been modified in transit.

Today's DNSSEC uses digital signatures that rely on traditional security assumptions such as factoring and discrete logarithms, which would not resist attacks by a cryptographically relevant quantum computer. To continue to provide its intended security guarantees in the face of such threats, DNSSEC must be updated to accommodate quantum-resistant algorithms. The post-quantum cryptography standardization project of the United States National Institute of Standards and Technology (NIST) announced in July 2022 [2] three post-quantum digital signatures algorithms to be standardized: CRYSTALS-Dilithium [11], Falcon [14], and SPHINCS+ [9]. All of these selected algorithms have one thing in common: the amount of data transmission required in order to perform a verification is substantially larger than their non-post-quantum counterparts: both public keys and signatures. This increase in size can cause substantial issues for pre-existing network protocols; DNS and DNSSEC are particularly sensitive to this issue.

Constraints on DNS and DNSSEC. There is an extremely large quantity of DNS traffic, so DNSSEC must be sufficiently efficient to support this high volume, which leads to the need for highly performant signature verification and, to a somewhat lesser extent, signature generation (signatures are often done offline and then transferred to the servers). DNS relies

primarily on UDP for communicating between servers. UDP has the benefit of being very lightweight and data efficient, however it has limitations that impact DNS: namely any UDP packet that exceeds 1500 bytes must be fragmented. UDP fragmentation is fragile and is generally not considered a reliable method for delivering large messages. With this in mind, accounting for the size of IPv6 headers, it is recommended that the DNS message sizes should not exceed 1232 bytes [8, 12]. As we will note below, for all three of the post-quantum signature algorithms selected by NIST, 1232 bytes is not enough to send both a public key and a signature, as is needed in some parts of DNS.

Admittedly, this 1232 byte limit does not mean that large DNS message cannot in principle be sent. When a DNS response exceeds 1232 bytes, a truncated response is sent instead indicating to the requester that they should then switch to using TCP instead of UDP. Unfortunately, a non-trivial number of name servers are estimated to not support TCP communication, preventing them from sending and receiving large DNS messages [12].

There have been two proposed mechanisms to solve the large DNS message issue [16, 17], both of which ultimately failed at getting standardized for use. Both mechanisms moved message fragmentation from the transport layer into the application layer, thus removing concerns of UDP fragmentation fragility and the lack of support of TCP. If a large DNS message needed to be sent, both of these mechanisms would split the DNS message into chunks and send each chunk one after the other. Fundamentally, both these mechanisms sent many, potentially large, packets, in response to a single request. There were significant concerns about the impacts these mechanisms would have. First, sending a many, potentially large, packets in response to a single request increases the risk and impact of denial of server amplification attacks. Second, sending many UDP packets in response to a single UDP request is an unusual behaviour, and some networks are configured to only accept a single UDP response packet to a single UDP request; the rest would trigger ICMP 'destination unreachable' packets, leading to concerns about ICMP flooding (which could reduce the utility of ICMP packets in debugging network issues).

Application level fragmentation is not the only solution presented for delivering large messages. Beernink presented in his thesis the idea of delivering large DNSKEYs out-of-band from DNS. The idea is that when a large DNSKEY is required, such as when using the now defunct round 3 candidate Raindbow [7], for verification the requesting server would initiate a HTTP or FTP request to fetch the large key.

Implications for post-quantum DNSSEC. When considering which post-quantum algorithms to standardize for DNSSEC, we must consider both the algorithms' operation performance as well as the sizes of its signatures and public keys. Müller et al. [12] began this discussion by evaluating the NIST Round 3 candidates in the context of DNSSEC. They established several requirements for a scheme to fulfil if it were to be used for DNSSEC signatures. As noted above, fragmentation is a major concern for DNSSEC and the recommended maximum DNS response size, including any signatures and public keys, should not exceed 1232 bytes. However, due to public keys not needing to be transmitted as often as signatures, larger public keys may be acceptable. Müller et al. also noted the requirement that a resolver should be able to validate at least 1000 signatures per second. The final requirement noted by Müller et al. is that zones should be able to sign 100 records per second.

Müller et al. identified three of the NIST Round 3 candidate algorithms that had the potential to fulfill these requirements: Falcon-512 [14], Rainbow-I_a [7] and RedGeMSS128 [5]. On first inspection it would appear that Falcon-512 is the clear winner as it is the only scheme that completely meets the requirements set above, however, both Rainbow-I_a and RedGeMSS128 have significantly smaller signatures sizes which made them appealing: Falcon-512 has a signature size of 0.7kB whereas the other two schemes have signature sizes of 66 bytes and 35 bytes respectively. The requirement that both Rainbow-I_a and RedGeMSS128 failed was that their public keys are 158kB and 375kB respectively, versus Falcon-512's much smaller size of 0.9kB. (Since the 2020 study of Müller et al., both Rainbow and GeMSS have succumbed to cryptanalysis that substantially undermines their claimed security [3,4], and they were not selected by NIST to advance beyond Round 3.) A conclusion of Müller et al. was that they expect that DNSSEC specification changes will be required before quantum safe cryptography can be deployed in order to support larger key sizes.

1.1 Our contributions

Given the inherent conflict between the larger public key and signature sizes of post-quantum algorithms and the practical 1232-byte limit on DNS packet size, we revisit fragmentation in hopes of finding a practical way forward. In this work we propose A Resource Record Fragmentation mechanism, or ARRF for short. ARRF is a *request-based* lightweight DNS fragmentation solution which removes the fragility of large DNS messages over UDP while being designed with backwards compatibility in mind. Similarly to previously proposed mechanisms, fragmentation is moved from the transport layer to the application layer, thus avoiding the fragility of UDP fragmentation. Whereas previously proposed mechanisms sent several response fragments for a single request, ARRF requires that fragments of specific resource records be explicitly requested. In particular, for large responses, the first response packet is truncated but includes sufficient information to allow the requester to make separate requests for each additional fragment, either in sequential or in parallel (the latter of which we called "batched ARRF"). Our fragmentation

Table 1: Resolution times and data transfer sizes for standard DNS (over UDP using TCP fallback) and parallel ARRF in one network scenario.

Algorithm	**Standard DNS**	**Parallel ARRF**
Resolution time (ms) with 10ms latency and 50 megabytes per second bandwidth		
Falcon-512	82.11	61.96
Dilithium2	82.24	62.52
SPHINCS+-SHA256-128S	82.59	63.45
RSA 2048 with SHA256	41.50	—
ECDSA P256	47.78	—
Data transfer (bytes)		
Falcon-512	3,112	2,557
Dilithium2	8,623	8,367
SPHINCS+-SHA256-128S	26,073	26,140
RSA 2048 with SHA256	1,081	—
ECDSA P256	504	—

approach based on explicit requests for fragments improves both backwards compatibility and addresses the concern over ICMP flooding. ARRF is also designed in such a way that it can be implemented with low impact on existing servers; in fact we were able to implement it as a transparent daemon sitting in front of an ARRF-unaware requester and resolver at both ends of a DNS lookup request, reducing the burden of deployment.

To evaluate our approach, we implemented the three post-quantum digital signature algorithms selected by NIST – specifically, parameter sets Falcon-512, Dilithium2, and SPHINCS+-SHA256-128S – in BIND using liboqs [18], as well as a daemon implementing ARRF sitting in front of the requester and resolver, transparently carrying out the ARRF fragmentation/reassembly. We were then able to carry out a variety of experiments on a simulated network with different latencies and bandwidth and different fragmentation sizes to evaluate the performance of ARRF compared to DNS over UDP with TCP fallback, measuring the total resolution time and the amount of data transmitted.

Detailed results across all the various scenarios can be found in Section 4. Table 1 shows the results for a low-latency (10ms) network scenario, when restricting DNS messages to be at most 1232 bytes. In this scenario, ARRF in batched mode (meaning with additional fragments requested in parallel) yields resolution times of approximately 62–63ms for our three post-quantum algorithms, compared to approximately 82ms when using standard DNS over UDP with TCP fallback. ARRF is also more data efficient for Falcon-512 and Dilithium2, with the small additional overhead on each ARRF fragment packet being outweighed by the cost of falling back to TCP and retransmitting the first fragment.

In all our tested scenarios, we found that Falcon-512 performs better than Dilithium2 due to Falcon-512's smaller signatures, suggesting that Falcon-512 may be the most suitable option currently available to be standardized for DNSSEC. We did however find that even with the improved performance of post-quantum algorithms in ARRF compared to standard DNS over UDP with TCP fallback, post-quantum algorithms incurred a performance penalty compared to non-post-quantum algorithms currently in use with DNSSEC (RSA and ECDSA) due to the unavoidable cost of transmitting more data. Overall, we conclude that ARRF is a promising option for transitioning to post-quantum DNSSEC: it has less performance degradation compared to standard DNS over UDP with TCP fallback.

It remains to evaluate the backwards compatibility of ARRF in real-world deployments, where there may be misconfigured network devices or poorly written software that incorrectly handles unrecognized fields. We did design ARRF to avoid some known problems by using EDNS(0) pseudo resource records and using request-based fragmentation rather than responder fragmentation. Assessing the success of this approach in real-world network scenarios is an important next step.

2 The Domain Name System

The Domain Name System is a distributed database primarily responsible for translating human readable domain names to machine understandable IP addresses. The DNS is broken up into *zones*, each responsible for a specific level of granularity of the translation process. Each zone is contains various types of *resource records* which correspond to *labels*. Resource records can be used to look up IP addresses associated to domain names, name servers of a zone, as well as many other types of data.

To assist with explaining how DNS translations are performed, we will suppose there is a client which wants the IP address for `example.com`. The client will generally send a query to a caching resolver to handle the rest of the translation on behalf of the client. Assuming the resolver does not have the answer to the `example.com` query, it will then query the root name servers for the name servers responsible for `.com` domain names. Once the resolver receives a reply from the root name servers, it will then query the name servers responsible for `.com` for the name servers responsible for `example.com`. Finally, once the resolver learns of the name servers responsible for `example.com`, it will query those servers for the IP address associated with `example.com`, and finally receive and forward the response to the client. The responses to each of the intermediate queries can be cached to reduce the resolution time and reduce load on name servers.

DNSSEC adds digital signatures to DNS to maintain data integrity. Resource record labels are not required to be unique,

so all resource records of a specified type and a specified label are grouped together as a RRSet. These RRSets are then signed by a specified digital signature algorithm, and the signature is stored inside of an RRSIG resource record. The public key is published to the zone inside of a DNSKEY resource record. There are generally two types of key pairs generated: Zone Signing Keys (ZSK), and Key Signing Keys (KSK). The ZSKs are responsible for signing and verifying the resources records in the zone, and the KSKs are responsible for signing the ZSKs and are what allows the chain of trust to be constructed.

As queries are made from the root servers to its children, and its children's children, eventually reaching the appropriate name server to answer the query, a chain of trust is constructed. Each zone that is queried must have a digest of the public KSK being used stored in a delegate signer (DS) record in its parent's zone, otherwise the public ZSK which is transmitted by the name server cannot be trusted. The one zone which does not publish a DS record is the root zone, due to its lack of parent. The public KSK of the root zone must be retrieved out-of-band from DNS; most modern operating systems have the root zone's public KSK pre-installed, removing the need for the user to fetch and configure the key themselves.

DNS as original specified only allows for DNS messages of at most 512 bytes over UDP, which quickly became too small to transport DNS messages, especially with DNSSEC being deployed. Extension Mechanisms for DNS (EDNS(0)) [6] introduced a way for resolvers to advertise the maximum sized UDP message they can receive, with a theoretical maximum of 2^{16} bytes. In reality, however, UDP/IP fragmentation can pose a significant issue for reliable delivery and thus the maximum recommended DNS message size over UDP is 1232 bytes [8].

3 Request-based fragmentation

As DNS is most reliable with limited size, single packets running over UDP, and given that post-quantum digital signature schemes have public key and signature sizes larger than can be accommodated in that limited size, something must change in order to reliably support post-quantum cryptography in DNSSEC. In a perfect world, we could simply send the larger DNS messages with little to no concern of them arriving. However, UDP fragmentation can cause significant problems for delivering large DNS message via UDP. The current solution to solving this problem is falling back to TCP; however, a non-trivial number of DNS name servers do not support TCP, and fallback to TCP can also incur a performance penalty. We look to solve this problem by moving DNS message fragmentation from UDP (transport layer) to DNS itself (application layer), while addressing concerns raised to previously proposed mechanisms. In this section we present our solution, A Resource Record Fragmentation mechanism, or ARRF for short.

3.1 Resource Record Fragments

When a DNS message is too large to fit into the maximum advertised UDP size, some of the message must be omitted while still containing meaningful information to the requester. We introduce a new type of pseudo-resource record: Resource Record Fragments (RRFRAGs). Like OPT [6], another pseudo-resource record, RRFRAGs are not explicitly in DNS zones. Rather they are created only when they are needed. RRFRAGs are designed similarly to the OPT pseudo-resource record; they use the standard resource record wire format but repurpose some of the fields. An RRFRAG contains the following fields:

- **NAME**: Must always be root (.) to reduce the amount of overhead required to send a RRFRAG while respecting the generic resource record format.
- **TYPE**: Used to identify that this pseudo-resource record is an RRFRAG.
- **RRID**: Used to indicate the particular resource record that is being fragmented. Since labels do not necessarily have distinct resource records attached to them, this allows a requester to be explicit in its request while not requiring the responder to remember which particular resource record it fragmented. The RRID of a particular resource record can be arbitrarily assigned, but must not change.
- **CURIDX**: The current index in the byte array of the original resource record which is being fragmented.
- **FRAGSIZE**: The total number of bytes contained in FRAGDATA plus two bytes to account for the extra space needed for the RRSIZE field. FRAGSIZE has two different meanings depending on the context. If the RRFRAG is part of a query, then this indicates how large the responding server should make this particular fragment. If the RRFRAG is part of a response, this field indicates how much data was sent in this particular fragment.
- **RRSIZE**: The size of the original non-fragmented resource record. This is used by the requester to determine how much data it still needs to request from the responder in order to reassemble that particular resource record.
- **RAGDATA**: The raw bytes of the fragment of the original resource record. In queries this is always empty. In responses this will contain FRAGSIZE bytes starting at CURIDX. It is possible for FRAGDATA to contain zero bytes in responses, which we will elaborate on later.

Figure 1 depicts how an RRFRAG maps onto the generic resource record format. Similar to a DNSKEY resource record where the extra fields reuqired are inside RDATA, an RRFRAG stores the RRSIZE alongside FRAGDATA inside RDATA. This was done to handle the case where an implementation which does not support ARRF blindly copies RDLENGTH, or in our case FRAGSIZE, bytes into a buffer prior to branching based on resource record type.

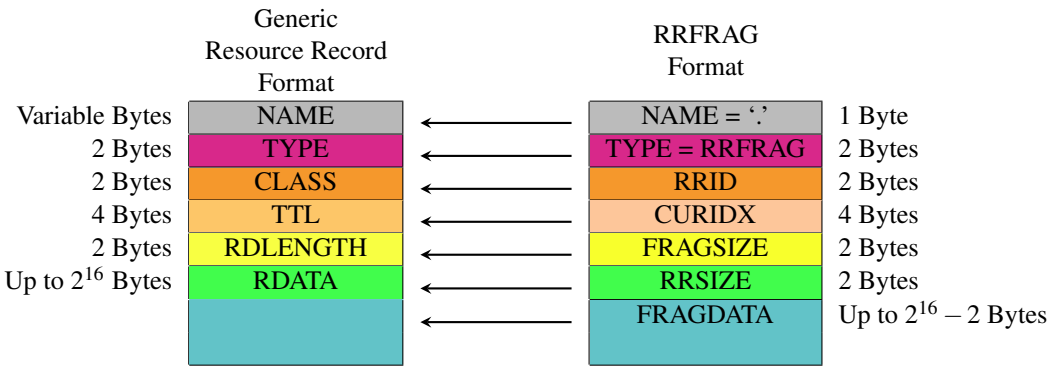

Figure 1: The mapping of the RRFRAG format onto the generic resource record format.

3.2 Using RRFRAGs

When a DNS response is too large to fit in the maximum advertised UDP size, RRFRAGs are used to split the data across multiple queries with each response's size below the advertised threshold. Resource records are replaced with RRFRAGs in place. That is to say, that if a resource record being fragmented is in a particular section of the DNS message, the RRFRAG replacing the resource record will be inserted into the same section. This is essential so that the original message format, once all resource records are assembled, will remain intact. It is important to note that the OPT pseudo-resource record must not be fragmented as it contains important meta data about the response, such as the DNS cookie. DNS messages that contain RRFRAGs should send as much data as they are able without surpassing the advertised threshold.

The initial response containing at least one RRFRAG can be considered a "map" of the non-fragmented message. This map is used by the requester to determine what the non-fragmented DNS message will look like upon reassembly. The requester can now determine what fragments it is missing in order to complete the original large DNS message, and can now send a new query for the missing RRFRAGs. It is the responsibility of the requester to specify which resource records it desires, how large the fragments should be, and where the fragments start. This is done by adding a RRFRAG for each distinct RRID the requester is requesting a fragment for in the query's additional section. If the response contains any non-RRFRAG resource records, it should store them until it is possible to reassemble the entire DNS message.

When the responder sees a query containing a RRFRAG, it just has to construct a standard DNS response by inserting the corresponding RRFRAGs into the answers section. The Fragdata being sent is a simple copy of the bytes of the desired resource record starting at CURIDX and ending at CURIDX + FRAGSIZE. This request/response cycle continues until the requester is able to reassemble the original large non-fragmented message. Note that, after receiving the initial response containing the map, nothing prevents the requester from making the subsequent RRFRAG requests in parallel.

For backwards compatibility reasons, whenever a response is sent which contains an RRFRAG, the truncated flag (TC) must be set in the DNS message header.

If a requester asks for a fragment which cannot be constructed, such as an RRID which does not map to a specific resource record, the responder should respond with a return code of FORMERR to indicate that the query was malformed.

3.3 Example execution of ARRF

To better solidify how ARRF works, we will now work through an example DNS query whose response is larger than the MTU. This example has had some details abstracted away and should not be used in place of the above specification when implementing ARRF. Figure 2 illustrates our example execution. This example begins at the last stage of name resolution for the query "example.com". We have two parties: the resolver making the DNSSEC-enabled query for example.com., and the example.com. name server.

First the resolver makes a standard request for the A record and its associated RRSIG. Upon receiving the request, the resolver observes that the DNS response is too large to fit within the confines of the MTU, and thus replaces the large RRSIG with an RRFRAG. This RRFRAG will contain as much of the original RRSIG as possible, and will inform the resolver how much of the original RRSIG is missing. Once the resolver receives the DNS response, it will copy both the entire A record as well as the RRFRAG and allocating enough space for the rest of the missing record. The resolver will then send another DNS query, but this time asking for an RRFRAG and sending its own RRFRAG indicating the next range of data it needs. Once the name server receives the RRFRAG query, it will use the RRFRAG in the additional section to determine the starting position and size of the fragment of the original RRSIG is being requested. The name server will construct a new DNS response containing the rest of our missing RRSIG inside of an RRFRAG and send the new response to

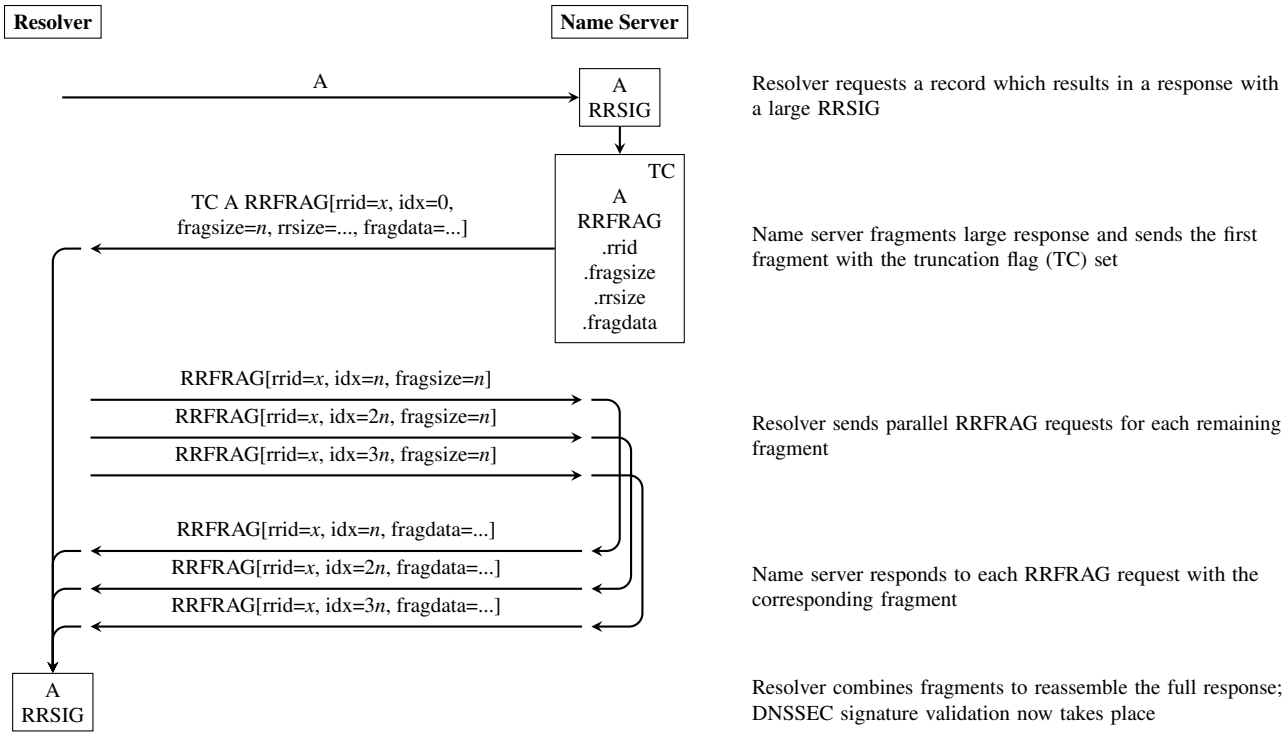

Figure 2: An example execution of ARRF

the resolver. Finally the resolver will copy the newly received RRFRAG into its state, reassemble the original RRSIG, and finally reconstruct the original large DNS response. DNSSEC validation now takes place, and if verification is successful the records are cached by the resolver.

3.4 Caching and DNSSEC Considerations

RRFRAGs themselves should never be cached. Once a DNS message is reassembled, and its DNSSEC authenticaiton validated if appropriate, then then non-fragmented resource records may be cached. If RRFRAGs could be cached, this would allow for malicious data to be accepted prior to validation. Caching complete resource records as opposed to RRFRAGs also allows for intermediate resolvers to send different fragment sizes than they originally received which allows for more flexibility to handle varying advertised UDP sizes.

4 Evaluation

In this section we evaluate the performance of post-quantum signature algorithms in DNSSEC without and with our request-based fragmentation technique ARRF.

4.1 Experiment setup

Algorithms. The algorithms we selected for the experiment are level-1 (128-bit-security) parameter sets of the three algorithms selected for standardization by NIST at the end of round 3: Falcon-512, Dilithium2-AES, and SPHINCS+-SHA256-128S. We also include results for RSA 2048 with SHA256 and ECDSA P256 for the sake of comparison.

Adding post-quantum algorithms to BIND. We evaluate these algorithms both using DNSSEC as defined today, as well as with ARRF. To perform this evaluation we used Internet Systems Consortium's BIND9 9.17.9 [10] as our DNS server software. We then added support for the three selected algorithms to BIND9 using Open Quantum Safe's liboqs 0.7.1 and OpenSSL 1.1.1l fork [1, 18]. To construct a test network environment, we used Docker and Docker's built in networking as well as Linux's '`tc`' (traffic control) to simulate network bandwidth and latency.

Daemon implementing ARRF. Rather than implementing ARRF directly into BIND9, we constructed a daemon which intercepts all incoming and outgoing network traffic and implements ARRF transparently for both the resolver and all name servers. We used libnetfilter-queue 1.0.3-1 to intercept packets.

We will now describe how the daemon behaves. When the machine acting as the name server receives a DNS query,

the daemon on the name server's side will modify the maximum advertised UDP message size to the maximum value of 65355 bytes.[1] The daemon then sends the message to the DNS software, which responds with a UDP message up to 65355 bytes. The daemon on the name server side receives this response and copies the entire message into its state. It outputs a response that is either the original message, if it fits within the requester's maximum UDP message size, or the first fragment if fragmentation is required. Whenever a fragment is requested in the future the daemon will use its state if possible rather than sending the request to the DNS software.

On the side of the DNS resolver, there is another copy of the daemon which intercepts incoming DNS responses and processes them before passing them on to the DNS resolver. When the resolver-side daemon receives a DNS response containing an RRFRAG, the daemon will intercept the message. The daemon will create a state for that individual transaction containing the metadata provided by the initial response's map and copy any data included into the state. The daemon will then execute ARRF and request the remaining fragments until the entire message can be reconstructed, at which point in time the daemon transparently sends the reconstructed message to the DNS resolver software.

DNS network design. We construct a simple DNS network consisting of a client, a resolver, and a name server each running in their own Docker container on the same machine. The name server zone contains 1000 'A' records, each with a unique label and signature. We query for each of these A records and measure the total resolution time for each one. The zone also contains 1 'primer' name record. We first query for this primer resource record so that our resolver has the DNSKEYs and NS records of our test domain, which means that we can evaluate ARRF's effect on an individual query. To model the worst case response size, we disabled 'minimal responses', and as such each response will contain 1 question, 1 A record, 1 NS record, 1 SOA record, and 3 RRSIGs. We use '`dig`' to issue each query and measure the total resolution time of said query.

We evaluated using the following four network conditions:

- low bandwidth, low delay: 10ms of delay and 128 kilobytes per second bandwidth;
- high bandwidth, low delay: 10ms of delay and 50 megabytes per second bandwidth;
- moderate bandwidth, high delay: 100ms of delay and 50 megabytes per second bandwidth; and
- ideal network: no delay, unlimited bandwidth (the only cost being processing the messages).

All experiments were run on a c5.2xlarge Amazon Web Services instance which provides 8 cores of a 3GHz Intel Xeon Platinum 8124M and 16 gigabytes of RAM.

[1] Modifications to BIND9 were required as the maximum DNS message size BIND9 supports is 4096

Table 2: Algorithm runtime measured using OQS-OpenSSL Speed

Algorithm	Sign (ms)	Verify (ms)
Falcon-512	0.2810	0.0438
Dilithium2	0.0753	0.0268
SPHINCS+-SHA256-128S	373.1	1.36
RSA 2048 with SHA256	0.6019	0.1772
ECDSA P256	0.0219	0.0677

4.2 Algorithm performance

To put the network results in context, it is important to understand the performance of the verification function of each of the algorithms. We use the Open Quantum Safe OpenSSL fork's `speed` command to measure each algorithm's signing and verification performance and report the results in Table 2.

4.3 Post-quantum with standard DNSSEC

In this section we measured how the post-quantum algorithms perform if they are deployed in DNSSEC as it is currently specified, under two scenarios and five different network conditions. We first measured how the algorithms would perform with a maximum UDP size of 1232. For messages larger than 1232 bytes, the DNS servers will fall back to TCP. The second scenario is the exclusive use of UDP for DNS communication, which provides an idealized view of the best case performance we can achieve using a particular algorithm; in this scenario, responses larger than the maximum advertised UDP message size will be fragmented *by the responder*, resulting in multiple UDP packets being sent in response to a single UDP packet request. Table 3 shows the average resolution times with standard deviation for the various network conditions. RSA 2048 with SHA256 and ECDSA P256 only have results recorded for standard DNS as the signatures of these algorithms are small enough to ensure they can fit in a single DNS message without fragmentation.

4.4 Post-quantum with ARRF

In this section we evaluate how each of the algorithms perform when using two different flavours of ARRF. First, we consider a "sequential" version. This version sends a request, receives a response, then looks what it needs to request and sends another request. This process is repeated until the entire message is received. Next, we consider a "parallel" version where once the first response is received the name server sends all of the requests for the remaining fragments at once, essentially parallelizing the ARRF mechanism. We consider several scenarios where the maximum DNS message size varies across all of the various network conditions described above.

Table 3: Mean resolution times (and standard deviation) in milliseconds for DNS without ARRF

Algorithm	Standard DNS	DNS using only UDP
10ms of latency and 128 kilobytes per second bandwidth		
Falcon-512	107.3 ± 1.786	61.52 ± 2241
Dilithium2	147.9 ± 1.478	102.0 ± 1.898
SPHINCS+	275.4 ± 2.114	229.4 ± 2.040
RSA 2048	52.20 ± 1.242	—
ECDSA P256	47.78 ± 1.949	—
10ms of latency and 50 megabytes per second bandwidth		
Falcon-512	82.11 ± 2.331	40.56 ± 2.115
Dilithium2	82.24 ± 2.216	40.77 ± 2.251
SPHINCS+	82.59 ± 2.096	41.16 ± 2.192
RSA 2048	41.50 ± 2.157	—
ECDSA P256	47.49 ± 1.919	—
100ms of latency and 50 megabytes per second bandwidth		
Falcon-512	802.1 ± 2.115	401.6 ± 1.991
Dilithium2	802.4 ± 2.032	401.5 ± 1.962
SPHINCS+	802.5 ± 1.940	401.9 ± 2.021
RSA 2048	401.3 ± 2.022	—
ECDSA P256	401.2 ± 2.176	—
0ms of latency and unlimited bandwidth		
Falcon-512	2.480 ± 3.884	1.1222 ± 2.034
Dilithium2	2.282 ± 3.318	1.240 ± 2.156
SPHINCS+	2.38 ± 3.500	1.176 ± 1.935
RSA 2048	1.672 ± 3.046	—
ECDSA P256	1.567 ± 2.711	—

Our daemon implementation is a prototype; with that in mind, it is important to understand the raw overhead that the daemon incurs. By setting the maximum DNS message size to be larger than any response (say, 65355 bytes), we can see how much of a cost we are paying just by having the proof of concept daemon involved. We then also evaluate what we would expect most operators would use as their maximum DNS message size of 1232 bytes. In order to see how ARRF scales, we also provide some smaller maximum DNS message sizes of 512 (the minimum DNS message size that must be supported) and 256 bytes. Table 4 shows the measured mean resolution time in milliseconds for the daemon running in sequential mode for the various network conditions measured, and Table 5 contains the results for the parallel daemon. Figures 3, 4, 5, and 6 illustrate all measured resolution times for standard DNS and DNS using ARRF for all network conditions.

4.5 Data transmission

In order to understand the full implications of deploying ARRF, we must also consider the amount of data transmitted compared to that of the DNS as it is currently standardized. Table 6 shows the total number of bytes required to transmit a complete DNS message signed with Falcon-512, Dilithium2, and SPHINCS-SHA256-128S both with and without ARRF deployed.

4.6 Results

Resolution times for standard DNS without ARRF. When considering standard DNS, RSA and ECDSA have the shortest resolution times with the best performing post-quantum algorithm being twice as slow across all network conditions. This is due to the response sizes being too large for a single UDP packet, causing it to be truncated and thus effectively making the initial query a wasted trip. The resolver must then fall back to the less performant TCP protocol to complete the lookup. When standard DNS using only UDP (with name-server-based fragmentation) is used, ECDSA and RSA only beat Falcon-512 and Dilithium2 when bandwidth was restricted to 128 kilobytes per second; this is likely due to the verification functions of Falcon-512 and Dilithium2 being more efficient than ECDSA and RSA.

Basic overhead of ARRF daemon. When considering the cases where the ARRF daemon is running, but not actively fragmenting resource records, we see comparable performance to standard DNS using only UDP. When comparing the post-quantum algorithms on standard DNS using only UDP versus the ARRF daemon using a maximum message size of 65355 bytes, we see a minimal overhead never exceeding 1.25 ms. Given that this is the overhead for our prototype daemon running as a separate process, we conclude that ARRF itself has very low overhead when fragmentation is not required.

Parallel versus sequential ARRF. When the ARRF daemon is fragmenting resource records, we see that the parallel daemon has a performance improvement of approximately 20% over TCP for all algorithms and all maximum messages sizes. This is due to the parallel nature of the parallel daemon effectively only paying the latency cost once after receiving the initial response, whereas TCP has a limited sized window restricting its parallelization, which cases the latency cost to be paid more times compared to the unlimited parallelization of parallel ARRF. The sequential daemon even outperforms TCP for Falcon-512 with a maximum messages size of 1232 bytes across all tested network conditions. This is due to the Falcon-512 signed response only requiring one additional round trip to reassemble the message, whereas the TCP fallback needs to receive the entire message from scratch

Table 4: Mean resolution times (with standard deviation) with ARRF using daemon in sequential mode

Algorithm	ARRF in sequential mode Resolution times (ms) for each maximum message size 65355 bytes	1232 bytes	512 bytes	256 bytes
10ms of latency and 128 kilobytes per second bandwidth				
Falcon-512	62.61 ± 2.052	84.414 ± 1.451	148.5 ± 1.587	275.8 ± 1.738
Dilithium2	103.2 ± 1.753	231.7 ± 1.841	422.7 ± 2.409	803.9 ± 1.344
SPHINCS+-SHA256-128S	230.7 ± 1.879	635.1 ± 2.088	1271 ± 1.963	2480 ± 1.916
10ms of latency and 50 megabytes per second bandwidth				
Falcon-512	41.77 ± 2.135	62.07 ± 2.278	122.5 ± 2.197	243.0 ± 2.269
Dilithium2	41.91 ± 2.108	162.9 ± 2.240	343.8 ± 1.899	705.6 ± 2.379
SPHINCS+-SHA256-128S	42.45 ± 2.160	424.7 ± 1.811	1028 ± 2.465	2173 ± 2.123
100ms of latency and 50 megabytes per second bandwidth				
Falcon-512	401.97 ± 2.060	601.1 ± 2.865	1203 ± 1.912	2404 ± 1.123
Dilithm2	402.1 ± 2.005	1604 ± 1.754	3405 ± 2.113	7008 ± 1.708
SPHINCS+-SHA256-128S	402.7 ± 1.957	4207 ± 2.166	10210 ± 1.843	21620 ± 1.440
0ms of latency and unlimited bandwidth				
Falcon-512	1.644 ± 2.334	1.992 ± 2.594	2.172 ± 2.361	2.668 ± 2.606
Dilithium2	1.804 ± 2.641	2.344 ± 2.495	2.932 ± 2.184	4.176 ± 1.291
SPHINCS+-SHA256-128S	1.992 ± 2.408	3.564 ± 1.460	5.692 ± 2.243	5.673 ± 2.389

(it cannot make use of the truncated response returned in the UDP response).

The sequential daemon performs worse in all other cases and is greatly affected by increased latency. This is due to the sequential daemon needing to wait for each request to be fulfilled before requesting the next piece, and TCP being able to achieve some parallelism due to its sliding window.

In the scenarios with latency and bandwidth restrictions, we see that, as the maximum message size is reduced, parallel ARRF scales very nicely due to parallelizing the requests, whereas sequential ARRF scales roughly by the factor that the maximum message size is reduced by.

Post-quantum versus non-post-quantum. When comparing post-quantum to non-post-quantum algorithms, Falcon-512 comes the closest to RSA and ECDSA in all constrained network scenarios, but is still slower despite the efficient verification function. Falcon-512 is affected primarily by bandwidth and is 60% slower than RSA and 76% slower than ECDSA in the 128 kilobytes per second scenario even when using parallel ARRF. If bandwidth is not a concern, then Falcon-512 performs better, but is still 49% slower than both RSA and ECDSA in both scenarios with 50 megabytes per second bandwidth. Unsurprisingly, Dilithium2 and SPHINCS+-SHA256-128S perform far worse than Falcon-512 and the non-post-quantum algorithms; roughly 1.5 and 3 times slower than Falcon-512 when using parallel ARRF, and even worse when using sequential ARRF.

Data overhead. When DNS messages sizes are at the recommended size of 1232 bytes, we can see that ARRF actually uses less data to transmit a DNSSEC response signed with Falcon-512 and Dilithium2. This is due to how DNS handles switching to TCP, essentially causing the three-way TCP handshake to turn into a five-way handshake, which we now explain. First the resolver sends a UDP request to the name server. The name server then sends a response identical to the request and marks the response as truncated. The resolver switches over to TCP and performs the standard TCP three-way handshake. TCP also sends an acknowledgement packet for each packet the requester receives, essentially offsetting the fragment requests in ARRF. With these factors, combined with UDP packet headers being 12 bytes smaller than those of TCP, ARRF allows efficient communication for both Falcon-512 and Dilithium2.

However, TCP becomes more data efficient compared to ARRF once many fragments are requested and sent, such as for SPHINCS+-SHA256-128S. Due to maintaining backwards compatibility, ARRF must surround all requests and responses inside of a DNS message and all fragments inside of an RRFRAG. TCP, on the other hand, is a stream which only sends a single DNS message header and sends the raw resource records themselves rather than sending the extra bytes that RRFRRAGs require. As mentioned earlier TCP sends ac-

Table 5: Mean resolution times (with standard deviation) with ARRF using daemon in parallel mode

Algorithm	ARRF in parallel mode Resolution times (ms) for each maximum message size			
	65355 bytes	**1232 bytes**	**512 bytes**	**256 bytes**
10ms of latency and 128 kilobytes per second bandwidth				
Falcon-512	62.80 ± 2.161	84.68 ± 1.765	86.15 ± 2.296	89.50 ± 2.120
Dilithium2	103.1 ± 1.855	127.9 ± 1.551	132.9 ± 2.038	142.7 ± 2.024
SPHINCS+-SHA256-128S	230.7 ± 1.908	262.9 ± 2.050	279.7 ± 1.720	311.6 ± 2.070
10ms of latency and 50 megabytes per second bandwidth				
Falcon-512	41.62 ± 2.060	61.96 ± 2.140	62.14 ± 2.343	62.16 ± 2.156
Dilithium2	41.02 ± 2.170	62.52 ± 2.240	62.96 ± 2.590	62.45 ± 2.590
SPHINCS+-SHA256-128S	42.35 ± 2.164	63.45 ± 2.241	64.44 ± 1.865	66.808 ± 2.247
100ms of latency and 50 megabytes per second bandwidth				
Falcon-512	400.6 ± 1.965	601.1 ± 2.212	601.2 ± 2.208	601.7 ± 2.168
Dilithm2	400.9 ± 2.044	601.7 ± 2.271	601.7 ± 2.209	602.4 ± 1.947
SPHINCS+-SHA256-128S	401.5 ± 2.145	602.4 ± 1.870	603.4 ± 1.638	605.5 ± 2.3638
0ms of latency and unlimited bandwidth				
Falcon-512	1.224 ± 2.428	1.471 ± 2.250	1.650 ± 2.310	1.769 ± 2.520
Dilithium2	1.185 ± 2.052	1.698 ± 2.365	1.875 ± 2.010	2.496 ± 1.871
SPHINCS+-SHA256-128S	1.436 ± 2.143	2.406 ± 1.876	3.461 ± 1.618	5.673 ± 2.389

Table 6: Total data transmitted when performing a DNS lookup

Algorithm	Bytes transmitted during DNS lookup			
	Standard DNS	**ARRF maximum message size**		
		1232 bytes	**512 bytes**	**256 bytes**
Falcon-512	3,112	2,557	2,947	3,637
Dilithium2	8,623	8,367	9,402	11,322
SPHINCS+	26,073	26,140	29,620	36,175

knowledgement packets for each TCP packet received. These acknowledgements are smaller than a UDP packet containing an ARRF request. The size difference depends on how many RRFRAGs are being requested, but the most common ARRF request in our experiments was 60 bytes including UDP, IP, and DNS message headers, and the largest request being 75 bytes, whereas TCP's acknowledgement packets are 52 bytes in size. If a DNS message is quite large, as is the case with SPHINCS+-SHA256-128S signed messages, these small savings end up making up for wasting the initial UDP request.

5 Discussion

Having seen the results of the experiments, we now discuss ARRF and consider whether if it is a viable solution for sending large DNS message.

5.1 Performance

Parallel ARRF is by far the most performant solution for larger responses, beating out TCP fallback in all cases despite how many requests and responses are required to transmit the original large DNS message. Sequential ARRF also outperforms TCP in cases where messages are only slightly larger than what can fit in a single UDP packet. However, parallel ARRF's performance does not come for free. On a busy resolver these parallel requests could eat up available bandwidth quite quickly and could potentially overwhelm middle boxes. We hypothesize that a production-ready version of ARRF would have a maximum window size similar to TCP in an effort to reduce request flooding, and therefore performance would lie somewhere between the ideal version of parallel ARRF and TCP. Despite there not being considerable differences between DNS with only UDP and the ARRF daemon running but not fragmenting, there are likely optimizations, such as multithreading, that can be made to the daemon. If ARRF was integrated directly into DNS software, it would also increase efficiency. We leave experimenting and evaluating these potential optimizations as well as evaluating window

sizes as future work.

5.2 Backwards Compatibility

As DNS is a distributed system managed by many different entities, in any deployment there will be requesters and name servers which do not understand ARRF. We now consider what happens in two such scenarios: when the requester implements ARRF but the responder does not, and when the requester does not implement ARRF but the responder does. We also discuss the impact ARRF has on middle boxes.

Requester implements ARRF but responder does not. When a requester which supports ARRF receives a response from a name server which does not support ARRF, it will, as per the current DNS specifications, receive a truncated DNS message with the TC flag set. It can then gracefully fallback to TCP and retry the query, therefore maintaining backwards compatibility.

Requester does not implement ARRF but responder does. Since the requester does not actually indicate its support of ARRF, it may appear at first glance that ARRF may cause issues when the requester receives a response containing an RRFRAG, as it will not be able to understand what an RRFRAG is, nor what it should do with it. Fortunately, older resolvers ignore unknown resource record types, so they will gracefully fallback to repeating the request over TCP as they will see that the TC flag is set. This results in no additional round trips compared to if ARRF was not being used.

Middle box support. By fragmenting at the DNS level, we should ensure that the majority of middle boxes will not cause issues for ARRF. From a middle box's perspective (even one unaware of ARRF), all messages sent using ARRF look like standard DNS messages which should not require any state to be properly routed. However, if there exist middle boxes which look inside DNS messages and view the types of the message's resource records, the new RRFRAG type could potentially cause those middle boxes to reject the message. Additional work would be required to determine if there are middle boxes with that behaviour, and how widespread they are.

5.3 Security Considerations

Denial of service attacks. ARRF is designed to not increase the scope of DoS attacks. Since fragments must be explicitly requested, a querier can reject any fragments it is not expecting (unlike responder-based fragmenting). When combined with DNS cookies, off-path attacks become infeasible. An adversary who is on-path could modify the values in responses which contain RRFRAGs, which could cause a querier to ask for fragments which do not exist. Middle boxes could also inject malicious data into individual RRFRAG's FRAGDATA fields. If DNSSEC is used, then this will cause the validation to eventually fail. This is acceptable as this validation failure, although denying service, is no worse than DNS without ARRF deployed (where a middle box adversary simply modifies the body or signature of a DNSSEC response). ARRF also limits the impact of amplification DoS attacks as it restricts the response sizes and each response needs a corresponding request. If a response arrives with the wrong id or DNS cookie, it should be discarded.

DNS cache poisoning. Since RRFRAGs themselves should not be cached, DNS cache poisoning is no more of a concern than it is in traditional DNS. If DNSSEC is used, then DNS cache poisoning is not a concern assuming a secure algorithm is used.

Memory exhaustion attacks. ARRF as specified is susceptible to memory exhaustion attacks. Although DNS cookies make this less of a concern for off-path adversaries, there is nothing stopping an on-path adversary from changing the RRSIZE fields in the initial response. Since the requester uses this initial response as a map without any validation thereof, an adversary could insert many RRFRAGs advertising they are fragments of extremely large resource records. The requester would likely then allocate enough memory to store the intermediate state until reassembly is possible, and could only detect the attack once trying to verify the signature. One potential solution to this would be to use some heuristics to determine if a RRFRAG map makes sense. Based on what the requester could expect to receive for a query of some form, the requester can check to see if the response it actually received fits within those expectations. For example, if the requester indicated that it only supported Falcon-512 signatures, it can check that the advertised sizes of the fragments are no larger than 690 bytes. We leave this issue for future exploration.

Unreliable networks. In this work we did not evaluate how ARRF performs when UDP packets do not reach their destination. BIND9 uses a default timeout of 800ms to determine whether it should try the request again or not, but it is unclear if that timeout duration would make sense for ARRF or not. This question must be answered before ARRF can be deployed and we leave this for future work.

5.4 Comparing ARRF Against Previous DNS Fragmentation Proposals

ARRF is not the first attempt at a DNS-level fragmentation mechanism. Since Sivaraman's draft "DNS message fragments" [16] was not as developed as Additional Truncated Response (ATR) [17], we will be primarily focusing on ATR

in this section. ATR, Sivaraman's draft, and ARRF, all rely on DNS-level fragmentation. The DNS servers are required to fragment messages and re-assemble them rather than relying on the transport layer to handle message fragmentation for them. All three mechanisms are transport layer agnostic and could therefore be used on both UDP and TCP. It may seem unclear why someone would want to run any of these mechanisms over TCP, however by doing so there is the potential for sending DNS messages larger than the 64 kilobyte maximum. ATR and Sivaraman's draft could in theory allow resource records of 64 kilobytes to be transmitted; whereas ARRF could allow for resource records of arbitrary length. This is due to the difference in granularity of fragmentation that the three mechanisms use. ATR and Sivaraman's draft fragment the DNS message as a whole, where as ARRF fragments individual resource records. Although there are no resource records that require an increase to the maximum DNS message size, and therefore maximum resource record size, it is not entirely unrealistic to see this issue potentially arising.

Before being broken [4], the Rainbow [7] post-quantum signature scheme was quite appealing due to its relatively small signature sizes; however it had large public keys of 161600 bytes. Since DNSKEYS are sent much less frequently than signatures, this might have been a reasonable trade off had Rainbow not been broken. It is entirely possible that a new, secure post-quantum signature scheme is created which has similar signature and public key sizes. (In fact, this is specifically mentioned as a desirable design characteristic in NIST's September 2022 call for additional post-quantum digital signature schemes [13].) In order to fully support arbitrary-sized resource records, the resource record format would need to be modified to support larger RDATA regions, and RRSIZE would need to be updated to the proper integer width.

One of the major criticisms of ATR [17] was that, since the mechanism would blindly send its additional message as part of its response, it would cause a flood of ICMP 'destination unreachable' packets to be created by resolvers which did not support ATR. Many implementations close their sockets immediately after receiving a response, so by the time the additional message is received the socket would no longer be accessible. This would make debugging considerably more challenging and reduce the usefulness of ICMP messages as a whole. Another issue arises with firewalls that have the policy of only receiving a single DNS message per query, and thus compounding the ICMP flood issue. ARRF does not suffer from these issues. Firstly, responses are only sent when they are explicitly queried for. A DNS server implementing ARRF will never send an additional response blindly and will never send additional messages to resolvers that don't support ARRF as they will never ask for them. Similarly, all DNS messages containing RRFrags will have an associated query and will therefore not get dropped by firewalls implementing the above policy. As ARRF does not suffer from either of those issues, there will not be a flood of ICMP packets that caused so much concern.

ATR also requires a slight delay between the first message being sent and the trailing messages being sent in order to maintain message ordering. Receiving messages out of order is not an issue for ARRF as the requesting server will know what to expect after receiving the first message containing the RRFRAG map of the whole DNS message. All responses after the first one will have been explicitly asked for and are not dependent on any other responses.

Where as ATR is quite lightweight, ARRF does have some additional transportation costs. ATR costs a single round trip plus the delay required to maintain message ordering, whereas ARRF has $\left\lceil \frac{\text{Original response size}}{\text{Maximum message size}} \right\rceil$ round trips. With the exception of the initial round trip, these round trips can be performed in parallel, thus reducing the overall resolution time. ARRF also requires more data to be sent, specifically as part of requesting the additional fragments. RRFRAGs in requests are 15 bytes in size, and the number sent depends on the number of resource records, how large they are, and how much data can fit in the maximum message size.

Sivaraman's draft [16] was built off of EDNS(0)'s OPT resource record requiring three fragmentation related options support to be assigned by ICANN. ARRF does not use the OPT pseudo-resource record and therefore does not require any options to be defined by ICANN.

Finally both ARRF and ATR can be implemented as a daemon on the resolver side without any changes required to the DNS software being used. This would make deployment much simpler as it would not require a DNS operator to update their resolver software and potentially have version incompatibilities. The reassembly could be performed entirely transparently to the resolver.

6 Future Work

Although ARRF appears to be a viable solution to solving DNS message fragmentation and therefore opening the door for post-quantum DNSSEC, additional work needs to be done. The backwards compatibility of ARRF needs to be further explored and evaluated in real-world deployments, exploring if there are middle boxes which cause ARRF to fail. ARRF as specified in this work is susceptible to memory exhaustion attacks and additional work needs to be done to prevent these attacks. It also also likely that operators will want to limit the number of concurrent requests when using parallel ARRF and therefore research into selecting a reasonable limit must be done.

In this work we provide a proof of concept daemon which transparently implements ARRF. Directly integrating ARRF into DNS implementations may uncover unexpected surprises.

Our experiments only considered the case of lossless packet delivery. In reality, UDP packet delivery is not guaranteed,

so research is needed on how ARRF behaves in unreliable networks. Work also needs to be done to measure any additional processing/memory overhead introduced by ARRF and whether that overhead is reasonable.

Any future standardization of ARRF would depend both on ARRF itself being evaluated by the Internet Engineering Task Force as well as appropriate post-quantum algorithms being specified for use in DNSSEC.

7 Conclusion

Post-quantum cryptography will inevitably need to be integrated into the DNSSEC ecosystem, however it looks like it will not be as smooth of a transition as we would like. Of our current options, Falcon-512 is by far the most performant but even with parallel ARRF is still significantly slower than currently used classical signing algorithms. There has been recent work on shrinking Falcon-512 signatures significantly which would improve its performance. Dilithium2 is perhaps viable as an alternative option, but considering the DNSSEC community's previous stance of "we can avoid sending large message by shaping their contents better (smaller signatures, less additional data)" [19], Dilithium2 may receive significant resistance if proposed for use in DNSSEC. SPHINCS+-SHA256-128S is by far the worst performing of the three NIST post-quantum selections due to its slow verification and extremely large signatures which causes very large resolution times.

Message sizes are not the only thing to consider when discussing which post-quantum signing algorithm to standardize for DNSSEC, as the security of the algorithms must also be considered. So far major attacks have been found against several candidates fairly late in the NIST selection process. To make matters worse, those algorithms were broken with traditional computers, therefore making the attacks much more practical. Although the three selected algorithms are believed to be secure now, will they hold up to additional scrutiny? Only time will tell. It is likely that using a hybrid of a classical signing scheme and post-quantum scheme will be desirable for some time to ensure that the signatures are at least as strong as what are currently standardized. This will come at a further performance cost and also increase communication sizes, and we plan to evaluate this additional cost in the future.

A final option is to wait for new post-quantum signature schemes to be invented and hope that signature sizes become more reasonable. NIST has requested additional post-quantum signature schemes be submitted for consideration standardization [13]. However, waiting several years for a better scheme to emerge is eating into the valuable time needed to prepare for securing DNS against a quantum adversary. It is best that we plan for the worst case of signatures sizes not improving, and be pleasantly surprised if such a scheme arises. With that in mind, we recommend Falcon-512 as a suitable signature algorithm for use in DNSSEC with ARRF as its delivery mechanism to achieve reasonable resolution times.

Acknowledgments

We gratefully acknowledge helpful discussion with Roland van Rijswijk-Deij, Andrew Fregly and Burt Kaliski, Sofía Celi, and Michael Baentsch. D.S. was supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grants RGPIN-2016-05146 and RGPIN-2022-0318, and a donation from VeriSign, Inc.

Availability

The software implementing the daemon and experiment is available at https://github.com/Martyrshot/ARRF-experiments/.

References

[1] The Open Quantum Safe project, 2022. URL: https://openquantumsafe.org.

[2] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone. Status report on the third round of the NIST post-quantum cryptography standardization process, July 2022. doi:10.6028/NIST.IR.8413.

[3] W. Beullens. Improved cryptanalysis of UOV and Rainbow. In A. Canteaut and F.-X. Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 348–373. Springer, Heidelberg, Oct. 2021. doi:10.1007/978-3-030-77870-5_13.

[4] W. Beullens. Breaking Rainbow takes a weekend on a laptop. Cryptology ePrint Archive, Report 2022/214, 2022. https://eprint.iacr.org/2022/214.

[5] A. Casanova, J.-C. Faugère, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem. GeMSS. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[6] J. da Silva Damas, M. Graff, and P. A. Vixie. Extension Mechanisms for DNS (EDNS(0)). RFC 6891, Apr. 2013. URL: https://www.rfc-editor.org/info/rfc6891, doi:10.17487/RFC6891.

[7] J. Ding, M.-S. Chen, A. Petzoldt, D. Schmidt, B.-Y. Yang, M. Kannwischer, and J. Patarin. Rainbow. Technical report, National Institute of Standards and

Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[8] DNS-Violations. DNS flag day 2020, 2020. URL: https://dnsflagday.net/2020/.

[9] A. Hulsing, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, P. Kampanakis, S. Kolbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, J.-P. Aumasson, B. Westerbaan, and W. Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[10] Internet Systems Consortium. BIND 9. https://www.isc.org/bind, 2021.

[11] V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehlé, and S. Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[12] M. Müller, Jins de Jong, M. van Heesch, B. Overeinder, and Roland van Rijswijk-Deij. Retrofitting post-quantum cryptography in internet protocols: a case study of DNSSEC. *ACM SIGCOMM Computer Communication Review*, 50(4):49–57, 2020.

[13] National Institute of Standards and Technology. Call for additional digital signature schemes for the post-quantum cryptography standardization process, Sept. 2022. URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf.

[14] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[15] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends. DNS Security Introduction and Requirements. RFC 4033, RFC Editor, Mar. 2005. URL: https://rfc-editor.org/rfc/rfc4033.txt.

[16] M. Sivaraman, S. Kerr, and L. Song. DNS message fragments, July 2015. URL: https://datatracker.ietf.org/doc/draft-muks-dns-message-fragments/00/.

[17] L. Song and S. Wang. ATR: Additional Truncation Response for Large DNS Response, Mar. 2019. URL: https://datatracker.ietf.org/doc/draft-song-atr-large-resp/03/.

[18] D. Stebila and M. Mosca. Post-quantum key exchange for the internet and the open quantum safe project. In R. Avanzi and H. M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 14–37. Springer, Heidelberg, Aug. 2016. doi:10.1007/978-3-319-69453-5_2.

[19] P. Vixie. Re: [dnsop] call for adoption: draft-song-atr-large-resp. https://mailarchive.ietf.org/arch/msg/dnsop/JdhkwdWT2hGzIwfVx6CrX15KCfk/, 2019.

A Appendix – Performance graphs

Figures 3, 4, 5, and 6 visualize the performance of ARRF in batched and sequential mode in various network scenarios and at different maximum UDP packet sizes compared with standard DNS with TCP fallback or UDP only mode.

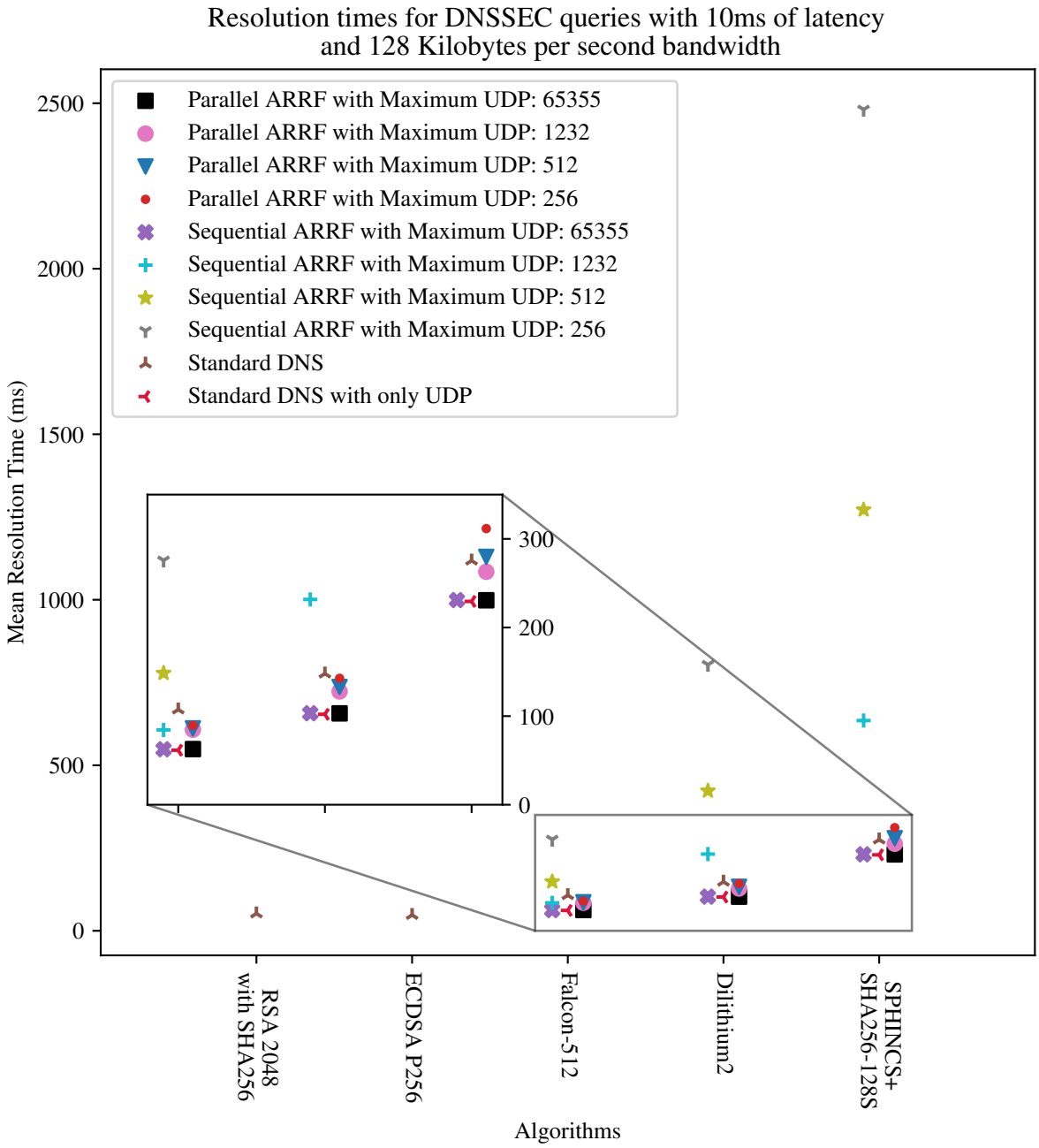

Figure 3: Mean resolution times in milliseconds with 10ms latency and 128 kilobytes per second bandwidth

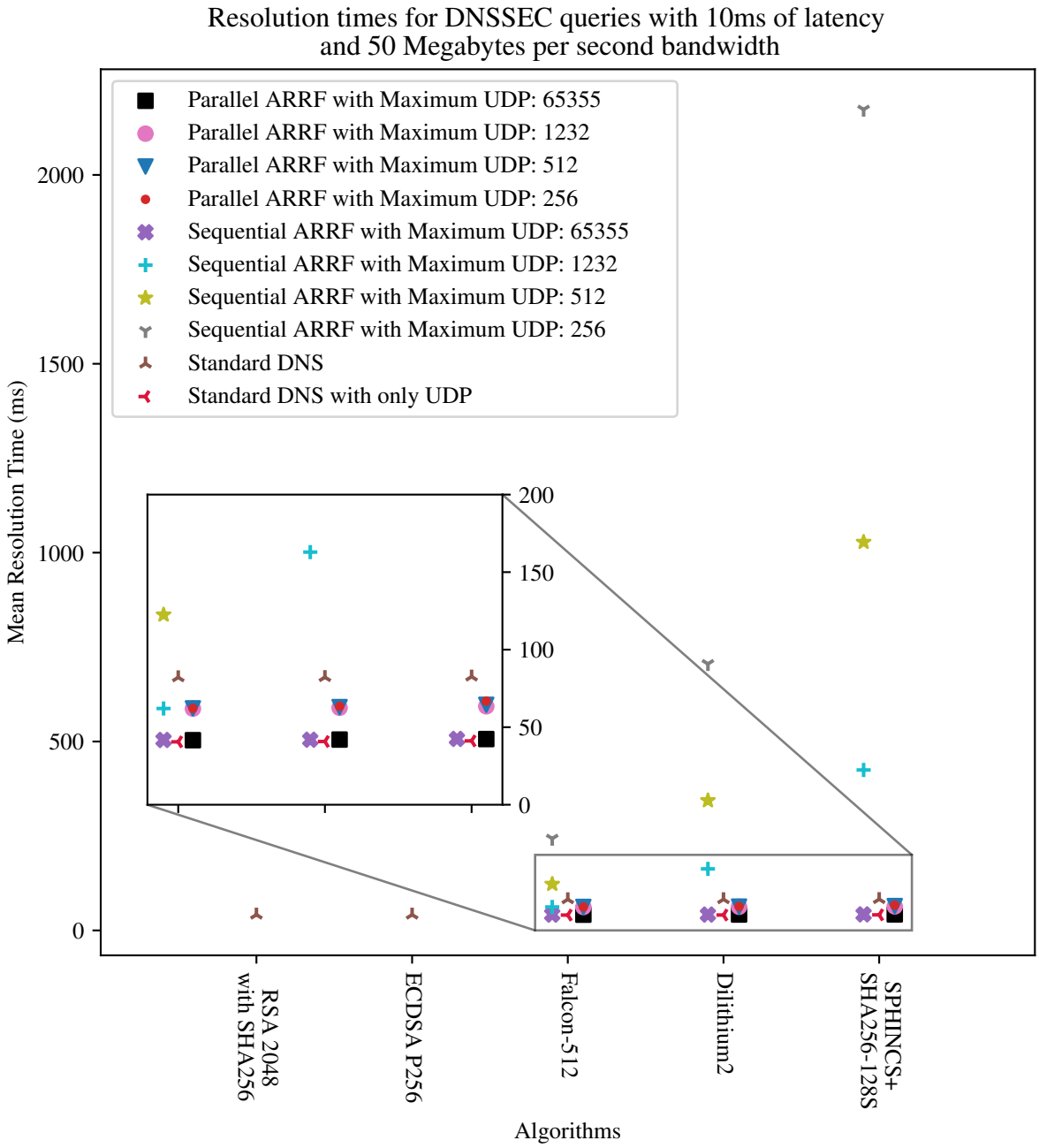

Figure 4: Mean resolution times in milliseconds with 10ms latency and 50 megabytes per second bandwidth

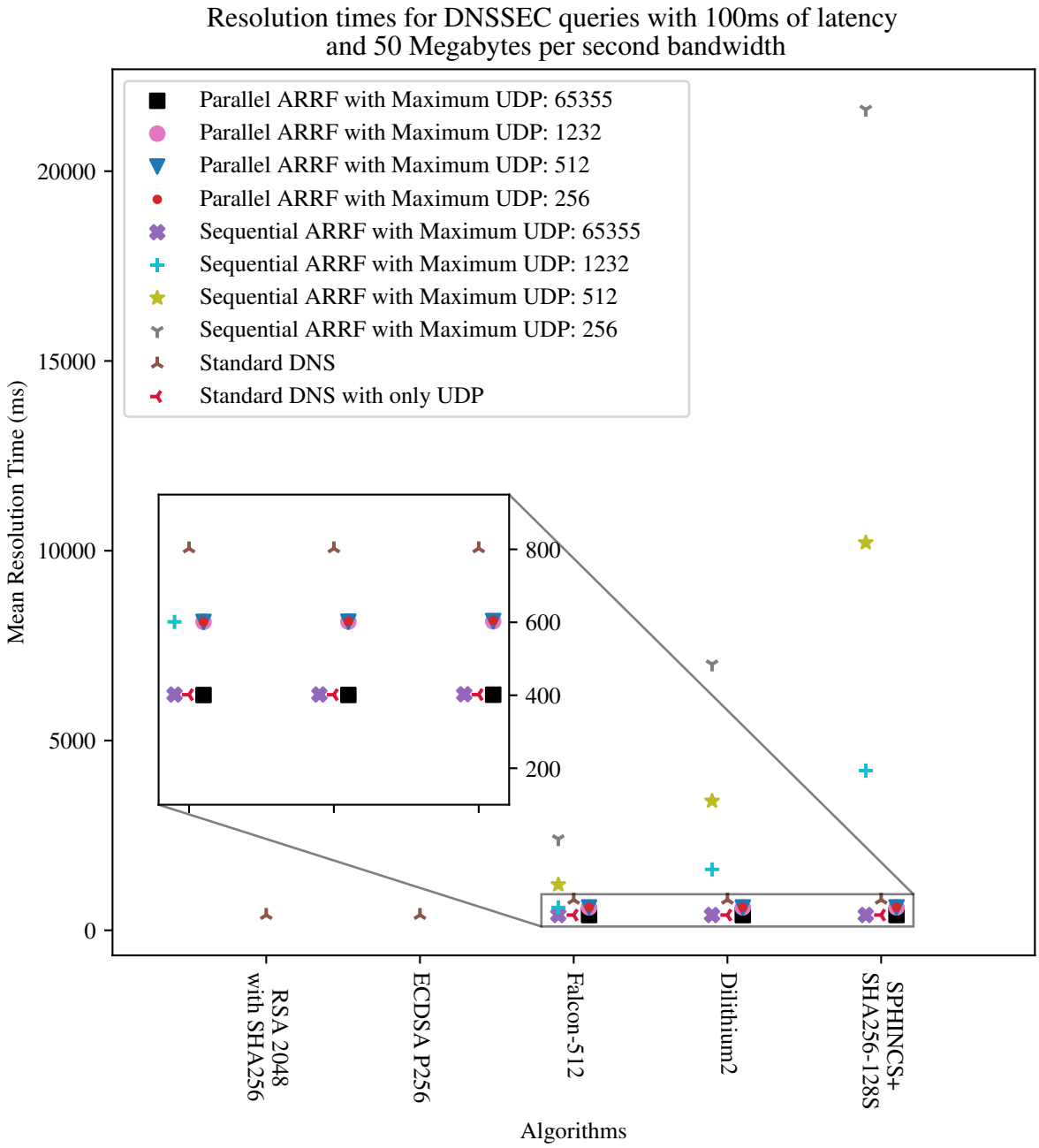

Figure 5: Mean resolution times in milliseconds with 10ms latency and 50 megabytes per second bandwidth

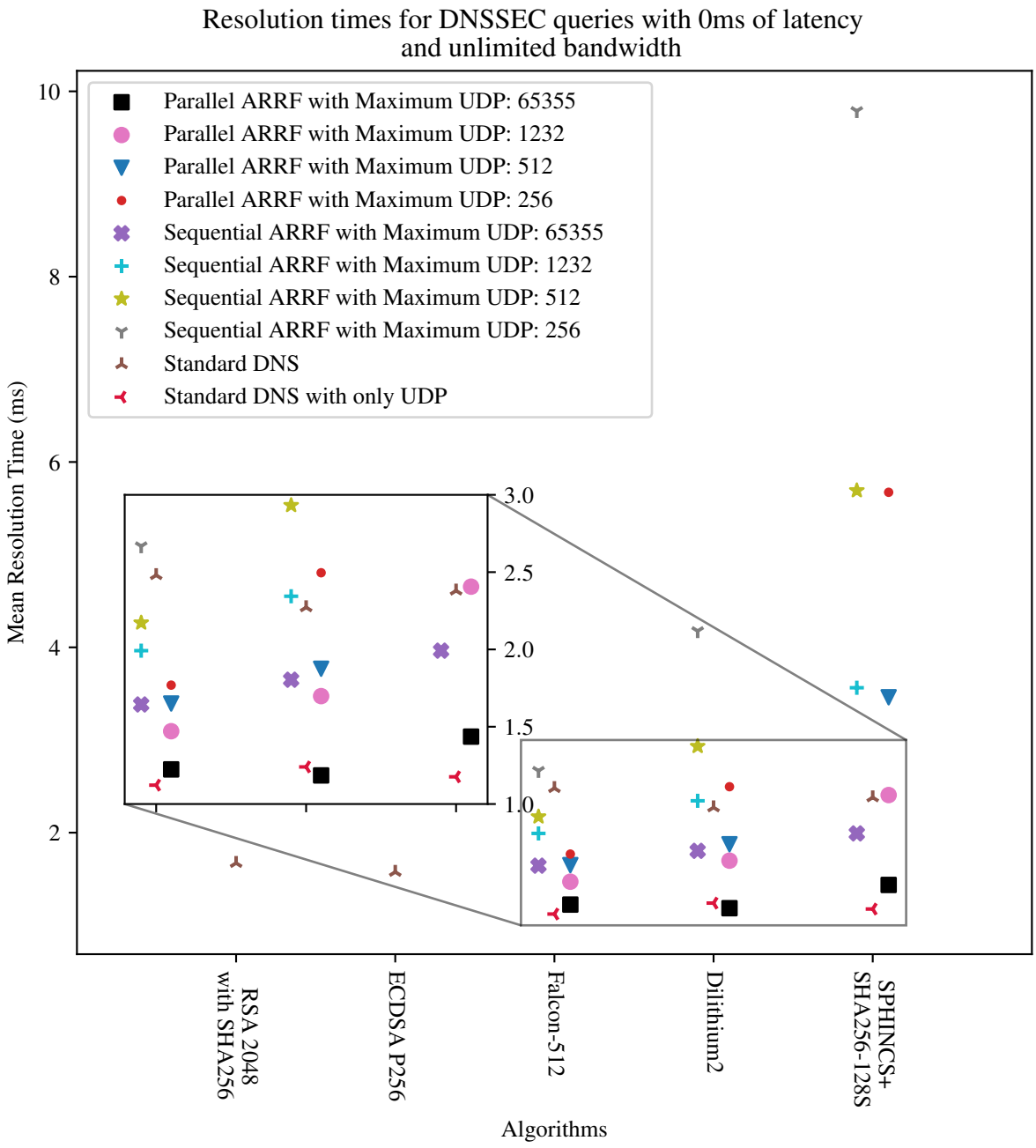

Figure 6: Mean resolution times in milliseconds with 0ms latency and unlimited bandwidth