Vol.2021-IOT-52 No.12
2021/3/1

# 端末上DNSSEC検証システムにおける2種類のフルサービスリゾルバを併用した名前解決高速化

井口 和哉[1,a)]　山井 成良[1,b)]　金 勇[2,c)]

**概要**：DNS（Domain Name System）における応答の正当性と完全性を検証することによってDNSキャッシュポイズニングを防ぐDNSSEC（Domain Name System Security Extensions）には多くの課題が存在することが分かっている．過去の研究で，キャッシュDNSサーバで行われていたDNSSEC検証をクライアント端末上で行う手法の提案が行われ，その手法ではDNSSECのいくつかの課題を解決できた．しかし，この手法では自身の端末で検証した結果だけを利用するため，DNS問合せにおけるキャッシュヒット率の低下が起こり，DNSSEC検証の実行回数の増加による名前解決の低速化につながっている．本研究では，名前解決の低速化を改善するためにこのシステムが行った検証だけでなく，キャッシュDNSサーバでの検証結果を併用できるようにすることでキャッシュのヒット率を上げ，名前解決の高速化を実現する手法を提案する．キャッシュDNSサーバからの応答がタイムアウト或いはSERVFAILになった場合は自身の端末で検証し，検証に失敗する場合はポップアップをできるようにする．既存のDNSと端末上DNSSEC検証システムを組み合わせることで，DNSSECの課題を解決しながらも高速な名前解決を実現する．

# Acceleration of a Client Based DNSSEC Validation System with Parallel Queries to two Different Full-Service Resolvers

## 1. はじめに

現在，インターネットは世界中で広く利用されており，インターネットは人間の社会活動に欠かせないものとなった．DNS（Domain Name System）はインターネットを利用・運用する上で必要不可欠であるため，悪意ある攻撃の対象にされやすい．DNSへの攻撃の一つとしてDNSキャッシュポイズニングがある．DNSキャッシュポイズニングはDNSによってインターネットユーザをフィッシングサイト等に誘導させ，個人情報を不正に取得するなどの攻撃である．

DNSキャッシュポイズニングを防ぐ技術としてDNSSEC（Domain Name System Security Extensions）がある．DNSSECによりDNSキャッシュポイズニングは完全に防ぐことができるが，DNSSECには多くの課題が存在することが分かっている．

DNSSECの課題を解決するために通常キャッシュDNSサーバで行われるDNSSEC検証をクライアント端末で行うシステム（以降，端末上検証システムと呼ぶ）[1]が金らによって提案された．端末上検証システムにより，DNSSECの課題のいくつかを解消できたが，キャッシュのヒット率が低いため，名前解決の低速化が課題となっている．

本論文では端末上検証システムを利用して端末上DNSSEC検証による安全な名前解決を実現しながら，DNSSEC対応のキャッシュDNSサーバに並列して問い合わせることでキャッシュのヒット率を上げる提案を行う．これにより，端末上検証システムでキャッシュヒットしない場合でも，キャッシュDNSサーバからDNSSEC検証が完了している応答を取得できれば名前解決が終了するまでの時間を短縮できるかどうか検証する．

本論文では，まず2章でキーワードのまとめと関連研究について述べる．3章では関連研究の問題解決のための手法について検討する．4章では本研究における提案手法について述べる．5章では提案手法に基づいてシステムの実

[1] 東京農工大学
Tokyo University of Agriculture and Technology, Koganei, Tokyo, Japan
[2] 東京工業大学
Tokyo Institute of Technology, Meguro, Tokyo, Japan
[a)] kazuigu@net.cs.tuat.ac.jp
[b)] nyamai@cc.tuat.ac.jp
[c)] yongj@gsic.titech.ac.jp

装とその評価を行う．最後に 6 章で本研究のまとめを述べる．

## 2. 背景

### 2.1 DNS と DNSSEC

DNS はインターネット利用・運用において重要なプロトコルの一つで，人にとって扱うのが容易な「名前」（例えば，www.cs.tuat.ac.jp.）とコンピュータが通信で利用する IP アドレス（165.93.162.76）を変換する名前解決を行う．DNS では，まずクライアントのスタブリゾルバというプログラムがフルサービスリゾルバ（再帰的キャッシュ DNS サーバ）に問合せを行う．フルサービスリゾルバは権威 DNS サーバに問合せを行い，最終的には問合せの答えを含んだ応答を権威 DNS サーバから取得し，スタブリゾルバがフルサービスリゾルバから応答を受けとることで名前解決が終了する．このとき，キャッシュ DNS サーバは名前解決の過程で得た権威 DNS サーバからの応答をすべてキャッシュする．これにより，2 回目以降の問合せには，高速かつ権威 DNS サーバに負荷をかけることなく応答を返すことが可能となる．

DNS キャッシュポイズニングは高速化，負荷分散のために設置されたキャッシュ DNS サーバを利用して行われる．権威 DNS サーバでは自身の管理するゾーンのリソースレコード（DNS 情報）を常に保持しているが，キャッシュ DNS サーバでは権威 DNS サーバから応答を受けて初めてリソースレコードを保持する．しかし，クライアントとキャッシュ DNS サーバでは権威 DNS サーバからの応答の正当性を検証することができない．そのため，攻撃者は権威 DNS サーバに成りすまして，キャッシュ DNS サーバに偽の応答を送り，保持させることができる．クライアントは攻撃されたキャッシュ DNS サーバにアクセスし，偽装された応答を取得・利用すると，攻撃者が用意したフィッシングサイト等に誘導される．

DNS キャッシュポイズニングへの対策としてソースポートランダマイゼーション [2] があげられるが，これは DNS キャッシュポイズニングの成功率を下げるだけの対処療法的な措置である．DNS キャッシュポイズニングへの根本的な対策として DNSSEC がある．DNSSEC は DNS のセキュリティ拡張で公開鍵暗号方式による電子署名を用いて，権威 DNS サーバからの応答の正当性を検証できる．具体的には，権威 DNS サーバで ZSK（Zone Signing Key）と呼ばれる公開鍵と秘密鍵を生成しておき，ZSK 秘密鍵で各リソースレコードの電子署名（RRSIG）を生成しておく．キャッシュ DNS サーバからの問合せに対し，リソースレコードと電子署名を応答し（必要があれば ZSK 公開鍵も応答），キャッシュ DNS サーバは電子署名を ZSK 公開鍵で検証する．検証に成功した場合，そのリソースレコードは信頼できるものとなる（図 1）．

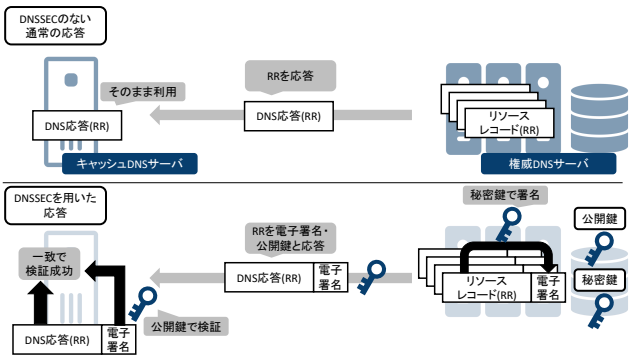


図 1　DNS のない通常の応答と DNSSEC を用いた応答

また，権威 DNS サーバの信頼性は信頼の連鎖によって検証される．権威 DNS サーバは ZSK の他に KSK（Key Signing Key）と呼ばれる公開鍵と秘密鍵を生成する．KSK 公開鍵のハッシュ値を親ゾーンの権威 DNS サーバに DS（Delegation Signer）レコードとして登録する．検証の際には自ゾーンの KSK 公開鍵と親ゾーンの DS を検証する．親ゾーンの DS を繰り返したどっていき，信頼できる DNSKEY レコードもしくは DS レコードを参照した場合に，権威 DNS サーバの信頼性が検証される．

### 2.2 関連研究

DNSSEC には多くの課題が存在する．関連研究 [1] と本論文では次の課題に着目した．

- キャッシュ DNS サーバで電子署名の検証が行われるため，キャッシュ DNS サーバの負荷が増加する．それにより，キャッシュ DNS サーバにおいてタイムアウトが発生しやすくなる．
- キャッシュ DNS サーバが DNSSEC 検証に失敗した場合，クライアントに「SERVFAIL」と返し，リソースレコードは何も返さない．しかし，「SERVFAIL」は様々な状況で発生するため，ユーザは「SERVFAIL」の原因を特定できない．
  - DNSSEC は管理工数が多く，鍵の更新や設定が複雑である [3]．正しい権威 DNS サーバであっても，管理者の設定ミスで「SERVFAIL」が発生する場合がある．
- 利用しているキャッシュ DNS サーバが DNSSEC に対応していない場合は，検証された応答を取得できない．
- DNSSEC は，キャッシュ DNS サーバと権威 DNS サーバの間で DNS キャッシュポイズニングが行われていないことを検証するもので，直接クライアントを標的とした DNS キャッシュポイズニングは防げない．

これらの課題を解決するために通常キャッシュ DNS サーバで行われる DNSSEC 検証を各クライアント端末で行う手法が提案された [1]．図 2 で名前解決の流れを簡単に説明する．なお，本論文ではシステム簡単化のため，DNSSEC

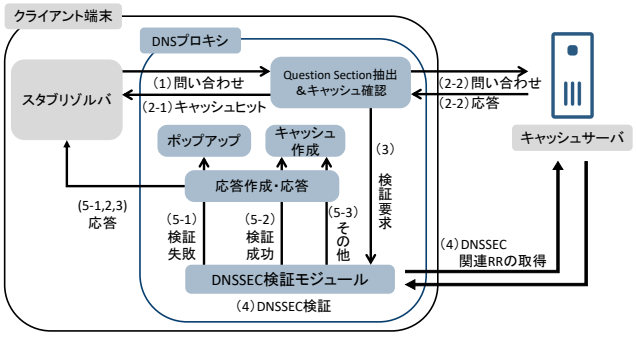


図 2　端末上検証システムにおける名前解決の流れ

検証モジュールの問合せ先は，クライアント端末の外部のキャッシュ DNS サーバを指定した．

**step 1**　スタブリゾルバは端末上検証システム（DNS プロキシ）に問合せを行う．

**step 2**　端末上検証システムはパケットを解析する．

**2-1**　端末内キャッシュに存在すれば，スタブリゾルバに応答を返す．（終了）

**2-2**　存在しなければ CD フラグ（DNSSEC 検証禁止フラグ）を 1 にしてキャッシュ DNS サーバに転送し，名前解決を行う．（**step 3** へ）

**step 3**　DNSSEC 検証モジュールにクエリ名，クエリタイプ，クラスと検証実行の種類を渡す．

**step 4**　DNSSEC 検証モジュールは DNSSEC 検証に必要なリソースレコードをキャッシュ DNS サーバから取得し，DNSSEC 検証を開始する．

**step 5**　DNSSEC 検証が時間内（DNS のタイムアウト直前）に終了すれば，次の条件に従ってスタブリゾルバに応答する．時間内に終了しなければ，**step 2-2** で得た応答をスタブリゾルバに送信して，図 3 のポップアップする．

**5-1**　DNSSEC 検証が失敗した場合は，検証過程で取得した応答をスタブリゾルバに送信する．その後，図 3 のポップアップによりユーザにフィッシングサイト等の可能性があることを警告する．

**5-2**　検証が成功した場合は，DNS 応答を作成し，スタブリゾルバに応答する．その後，結果を端末にキャッシュする．

**5-3**　クエリ名，クエリタイプが DNSSEC 検証に対応していないなどで DNSSEC 検証が最後まで行われない場合は，応答をスタブリゾルバに送信する．その後，結果を端末にキャッシュする．

キャッシュ DNS サーバで DNSSEC 検証を行わず，自身の端末上で行うため，キャッシュ DNS サーバに負荷をかけずに済む．また，キャッシュ DNS サーバの DNSSEC 検証が対応しているかどうかに関わらず，DNSSEC 検証を行う名前解決をすることができる．DNSSEC 検証が失敗もしくはタイムアウトした場合にも，必要なリソースレコードを取得しているため，それらを利用して名前解決を行うことができる．その際，図 3 に示すようなポップアップをすることでユーザにフィッシングサイト等に誘導されている可能性があることを警告するようになっている．

クライアント端末で行われた DNSSEC 検証の結果をキャッシュする機能 [4] も備えている．これにより，同一のクエリ名，クエリタイプ，クラスの名前解決の場合，キャッシュがある場合は DNSSEC 検証をせず，キャッシュを利用して名前解決を終了させることができる．

## 3. 検証済みリソースレコードの共有

端末上検証システムで DNSSEC 検証を行った場合，100-200ms ほど名前解決に時間がかかる．また，端末の状態（メモリや CPU の利用率）や端末の性能，ネットワークの性能が原因でさ名前解決にさらに時間がかかり，1000ms を超える場合もある．端末上検証システムのキャッシュ機能により，2 回目以降の同じ内容の問い合わせは 20-40ms ほどで高速に行われるようになり，名前解決の高速化を実現できるようになった．しかし，このキャッシュ機能は各クライアント端末でのみ有効なため，各クライアントの初回の問合せでは必ず端末上検証システムで DNSSEC 検証が行われる．通常のキャッシュ DNS サーバではアクセスしているクライアントの一人が問合せを行えば，別ユーザはすべてキャッシュを利用して，高速に名前解決を終了させることができる．そのため，端末上検証システムではキャッシュ DNS サーバに比べ，キャッシュのヒット率が低く，名前解決にかかる時間が平均的に長くなることが分かっている．

そこで，DNSSEC 検証が行われたリソースレコードを複数ユーザで利用できる方法をいくつか検討した．

### 3.1 共有用サーバの検討

一つ目は検証済みリソースレコードを共有用サーバへ登録し，各クライアント端末はその共有用サーバへ問合せを行う手法を考えた．共有用サーバへの更新は DNS プロトコルをそのまま使えるようにするため，動的 DNS を利用してプッシュすることで更新することを検討した．そのため，共有用サーバは再帰問合せ無効の DNS サーバを構築して，動的 DNS 用の TSIG キーを生成，各クライアント端末の端末上検証システムに TSIG キーを配布した．

名前解決の流れを図 4 に示す．基本的な名前解決の流れは 2.2 節の端末上検証システムと同じだが，キャッシュ DNS サーバに DNS 問合せを行う前に，共有用サーバへ DNS 問合せを行う．答えの応答があればそれをスタブリゾルバに送信するようにしている．また，DNSSEC 検証に成功した場合，TSIG キーを利用して共有用サーバへリソースレコードの更新を行うようにしている．

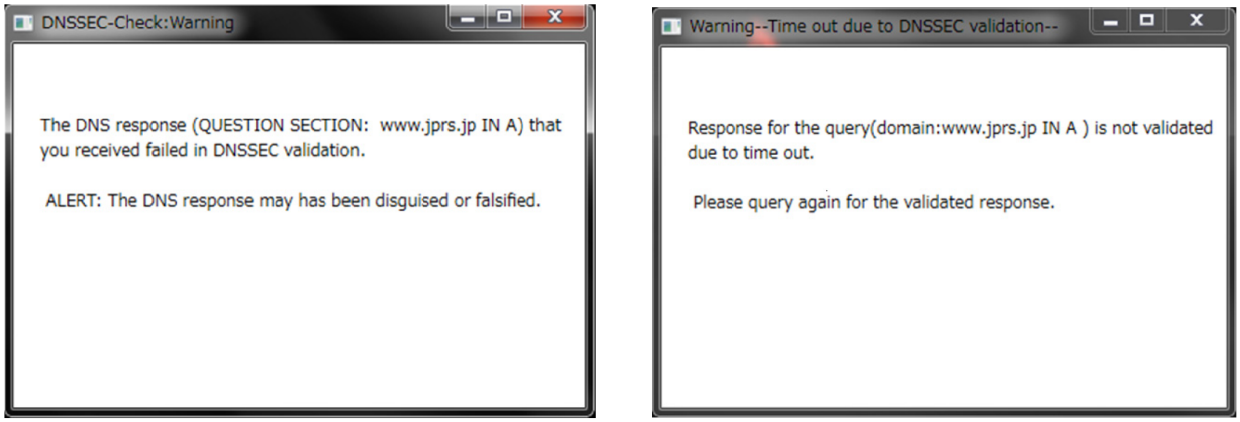


図 3　端末上 DNSSEC 検証システムにおけるポップアップ（左：検証失敗，右：タイムアウト）

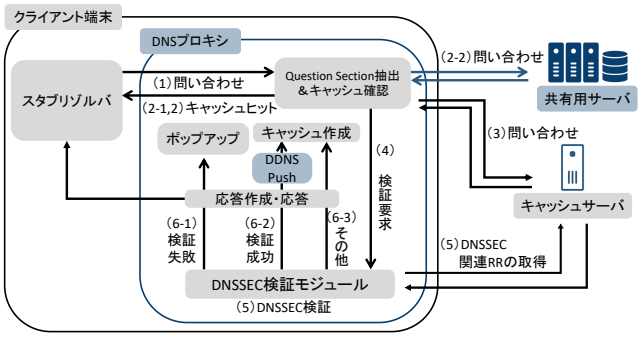


図 4　共有用サーバを利用した名前解決の流れ

これにより，クライアントが複数いる場合に名前解決の高速化を期待することができるが，いくつか課題が挙げられる．

- クライアント端末側から共有するリソースレコードを更新するため，不正アクセスされている端末や悪意あるユーザに攻撃される可能性がある．例えば，DNS キャッシュポイズニングと同様に偽のリソースレコードを登録させる攻撃が想定される．
- 動的 DNS では権威 DNS サーバを利用するため，リソースレコードの TTL が減らない．そのため，リソースレコードの削除をする必要がある．
- TSIG キーの生成・配布・更新の手間がかかる．

### 3.2　DNSSEC 有効共有用サーバの検討

共有用サーバでは課題がいくつか挙げられた．特にセキュリティの問題は無視できないため，共有用サーバにはリソースレコードの DNSSEC 検証を行う別プロセスを起動し，登録されたリソースレコードの正当性を検証する手法を検討した．しかし，この手法は設計の段階でいくつか課題があげられた．

- 更新されたリソースレコードが DNSSEC 検証を終え，検証結果を反映するまでには時間がかかる．検証が終わったリソースレコードと終わっていないリソースレコードの判別方法が決まっていない．
- 共有用サーバはインターネットに接続する必要がなかったが，共有用サーバはインターネットに接続する必要があるため，3.1 節の共有用サーバよりセキュリティが低下する．

### 3.3　2 台の DNS サーバに問い合わせを行う DNS プロキシサーバの検討

次に，端末上検証システムが利用するキャッシュ DNS サーバと共有用サーバに問い合わせを行う DNS プロキシサーバを検討した．共有用サーバに再帰問合せ無効の DNS サーバと動的 DNS を利用するのをやめ，共有用サーバを再帰問合せ・DNSSEC 有効の DNS サーバ（以降，DNSSEC 対応キャッシュサーバと呼ぶ）に変更することにした．これにより，ユーザ側からリソースレコードを更新することがなくなり，共有用サーバが攻撃される可能性を下げることができる．クライアントは DNS プロキシサーバに問い合わせを行うと，DNS プロキシサーバは端末上検証システムで使うキャッシュ DNS サーバ（以降，区別のため，端末上検証用キャッシュサーバと呼ぶ）と DNSSEC 対応キャッシュサーバに問い合わせを行う．DNS プロキシサーバは 2 台のキャッシュ DNS サーバから応答を受けると DNSSEC 対応キャッシュサーバからの応答を優先して返す．端末上検証システムは DNSSEC 対応キャッシュサーバから応答を受けた場合はそのまま名前解決を完了させ，端末上検証用キャッシュサーバから応答を受けた場合は端末上検証システムで DNSSEC 検証を行う．応答の判別には DNS パケットのヘッダセクションの CD フラグと AD フラグの組み合わせを利用することを検討した．

しかし，この手法では次のような課題が発生し，高速な

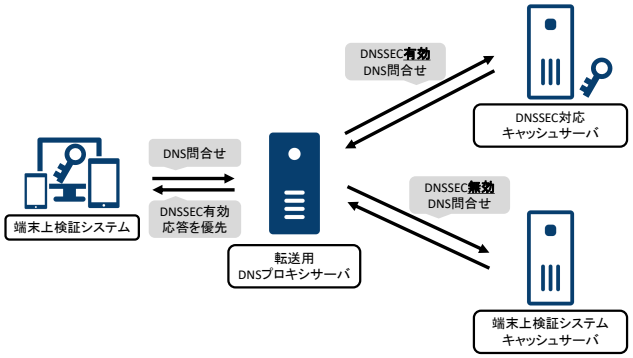


図 5　2 台の DNS サーバに問い合わせを行う DNS プロキシサーバ

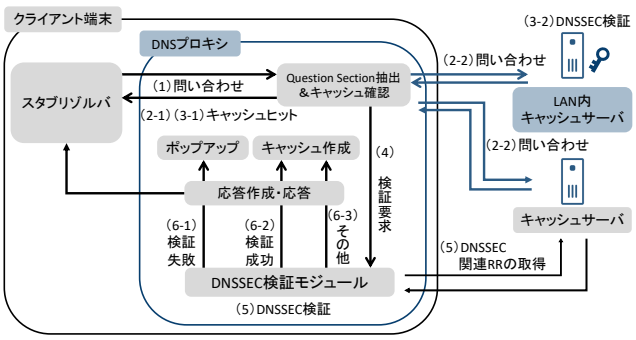


図 6　提案手法における名前解決の流れ

名前解決に成功しなかった．

- 端末上検証用キャッシュサーバと DNSSEC 対応キャッシュサーバから応答が返ってくる時間がほぼ同じである場合が多かった．そのため，DNSSEC 対応キャッシュサーバの結果を利用するには若干の遅延が必要になる．
  - DNSSEC 対応キャッシュサーバがキャッシュを持っていない場合は，応答が遅い，もしくは，返されないことがあるため遅延が無駄になる．
  - パケット受信から転送までの間でオーバヘッドが生じる．
- 端末上検証用キャッシュサーバからの応答と DNSSEC 対応キャッシュサーバからの応答の識別にはヘッダフラグを利用していたが，利用する DNS のプログラムや DNS プロトコルの変更によっては識別が困難になる可能性がある．

## 4. 2 種類のキャッシュ DNS サーバへの問合せ利用した高速化

2 台のキャッシュ DNS サーバに問い合わせを行う DNS プロキシサーバでは高速化のために，DNSSEC 対応キャッシュサーバからの応答を待たなければならなかった．そこで，端末上検証システムが直接 2 台のキャッシュ DNS サーバに問合せを行う手法を提案する．端末上検証システムによる名前解決と DNSSEC 対応キャッシュサーバへの問合せによる名前解決の 2 種類が行われ，スタブリゾルバに先に返された方の応答を利用する．後に返された応答はスタブリゾルバには利用されず，破棄される．また，2 台のキャッシュ DNS サーバへの問合せは順番に行うのではなく，並列に問合せを行う．並列化の手法に関しては次章で述べる．

全クライアントで初めての問合せのときには DNSSEC 対応キャッシュサーバからの応答は遅く，端末上検証システムによる応答の方が先に返される．しかし，2 台目以降のクライアントで同じ内容の問合せが発生した場合は，端末上検証システムより DNSSEC 対応キャッシュサーバからの応答の方が先に返される．また，2 台の DNS サーバを利用することで一方でタイムアウトが発生した場合でも，もう一方で名前解決を完了させることができる．この手法では，高速化だけでなく，名前解決に冗長性を持たせることも期待できる．

提案手法における名前解決の流れを次に示す（図 6）．

**step 1**　スタブリゾルバは端末上検証システム（DNS プロキシ）に問合せを行う．

**step 2**　端末上検証システムはパケットを解析する．

**2-1**　端末内キャッシュに存在すれば，スタブリゾルバに応答を返す．（終了）

**2-2**　存在しなければ二つのキャッシュ DNS サーバに並列に問合せを行う．端末上検証用キャッシュサーバには CD フラグを 1 にして問合せを行い，DNSSEC 対応キャッシュサーバには CD フラグを 0 にして問合せを行う．（**step 3** へ）

**step 3**　二つのキャッシュ DNS サーバから応答を受けとる．

**3-1**　DNSSEC 対応キャッシュサーバから応答を受けた場合は，そのままスタブリゾルバに応答する．端末上検証システムでのキャッシュ機能を有効にするため，端末上検証システムでの DNSSEC 検証は継続する．（**step 4** へ）

**3-2**　DNSSEC 対応キャッシュサーバから応答が遅い，もしくは，応答がない場合は，端末上検証システムでの DNSSEC 検証を開始する．DNSSEC 対応キャッシュサーバは次の問合せに備えて，DNSSEC 検証有効の名前解決を完了させておく．（**step 4** へ）

**step 4**　DNSSEC 検証モジュールにクエリ名，クエリタイプ，クラスと検証実行の種類を渡す．

**step 5**　DNSSEC 検証モジュールは DNSSEC 検証に必要なリソースレコードをキャッシュ DNS サーバ（本論文では端末上検証用キャッシュサーバ指定）から取得し，DNSSEC 検証を開始する．

**step 6**　DNSSEC 検証が時間内（DNS のタイムアウト直前）に終了すれば，次の条件に従ってスタブリゾルバ

に応答する．時間内に終了しなければ，**step 2-2** で得た応答をスタブリゾルバに送信して，図 3 のポップアップする．端末上検証システムでの DNSSEC 検証完了前に DNSSEC 対応キャッシュサーバから応答があれば，そちらを端末上での DNSSEC 検証なしで利用する．

**6-1** DNSSEC 検証が失敗した場合は，検証過程で取得した応答をスタブリゾルバに送信する．その後，図 3 のポップアップによりユーザにフィッシングサイト等の可能性があることを警告する．

**6-2** 検証が成功した場合は，DNS 応答を作成し，スタブリゾルバに応答する．その後，結果を端末にキャッシュする．

**6-3** クエリ名，クエリタイプが DNSSEC 検証に対応していないなどで DNSSEC 検証が最後まで行われない場合は，応答をスタブリゾルバに送信する．その後，結果を端末にキャッシュする．

## 5. 実装と性能評価

### 5.1 システム設計・実装

本研究におけるシステムは，2.2 節の端末上検証システムに機能追加して実装を行った．基盤となる DNS プロキシには Perl モジュールの Net::DNSServer::Proxy[5] を，DNSSEC 検証モジュールには Net::DNS::SEC::Validator[6] を利用した．DNS プロキシの制御には，Net::DNSServer::Proxy を制御している Net::DNSServer[7] を変更して端末上検証システムの実装を行った．

端末上検証システムの処理フローを図 7 に示す．図中に出てくる$status は DNSSEC 検証の結果を示している [1]．$status が 1,3,4 の場合は DNSSEC 検証が失敗したことを示し，128,133,134 の場合は DNSSEC 検証の成功を示している．それ以外の場合は，例えば，DNSSEC に対応していないクエリ名やクエリタイプのため，DNSSEC 検証を完了させられなかった場合などである．

**step 1** スタブリゾルバから DNS パケットを受け取る．

**step 2** クエスチョンセクションからクエリ名，クエリタイプ，クラスを取得する．

**step 3** DNS 問合せの内容が端末上検証システムのキャッシュにあるか確認する．キャッシュがあれば応答を作成し，スタブリゾルバに応答を送信する．（終了）

**step 4** キャッシュがなければ，CD フラグを確認する．本システムでは，CD フラグが 1 の場合に端末上検証システムでの DNDNSSEC 検証を行うように指定した．CD フラグが 0 の場合は，通常の名前解決を実行してスタブリゾルバに応答を送信する．（終了）

**step 5** CD フラグが 1 の場合は，端末上検証システムでの名前解決を始めると同時に，DNSSEC 対応キャッシュサーバに DNS 問合せを送る．

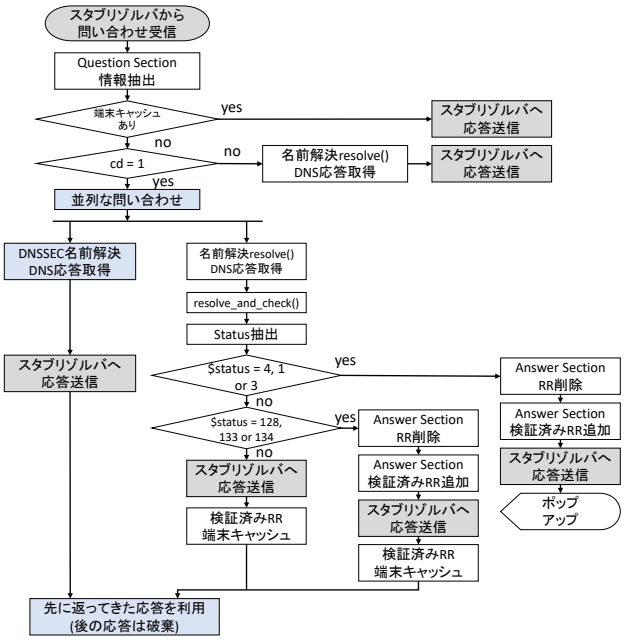


**図 7** 2 種類のキャッシュ DNS サーバへの問合せを利用した端末上検証システムのフローチャート

**step 6-a** （図 7 の左側フロー）DNSSEC 対応キャッシュサーバへ CD フラグを 0 にして，DNS 問合せを行い，応答を受け取ったらスタブリゾルバに応答を送信する．

**step 6-b-1** （図 7 の右側フロー）端末上検証システムでの DNSSEC 検証を開始する．端末上検証用キャッシュサーバに CD フラグを 0 にして DNS 問合せを行う．

**step 6-b-2** resolve_and_check で DNSSEC 検証を行う．

**step 6-b-3** $status で DNSSEC 検証の結果を取得する．

- DNSSEC 検証が失敗した場合は，検証過程で取得した応答をスタブリゾルバに送信する．その後，図 3 のポップアップによりユーザにフィッシングサイト等の可能性があることを警告する．
- 検証が成功した場合は，DNS 応答を作成し，スタブリゾルバに応答する．その後，結果を端末にキャッシュする．
- クエリ名，クエリタイプが DNSSEC 検証に対応していないなどで DNSSEC 検証が最後まで行われない場合は，応答をスタブリゾルバに送信する．その後，結果を端末にキャッシュする．

本論文では 2 種類の並列して問い合わせる方法を提案する．一つ目は Fork によって並列化し，一方で端末上検証システムで DNSSEC 検証行い，もう一方で DNSSEC 対応キャッシュサーバに名前解決を行う方法である．二つ目は DNSSEC 対応キャッシュサーバに DNS 問合せを行い，スタブリゾルバに応答を送信するプロセス（以降，名前解決専用プロセスと呼ぶ）を別で起動する方法である．端末上

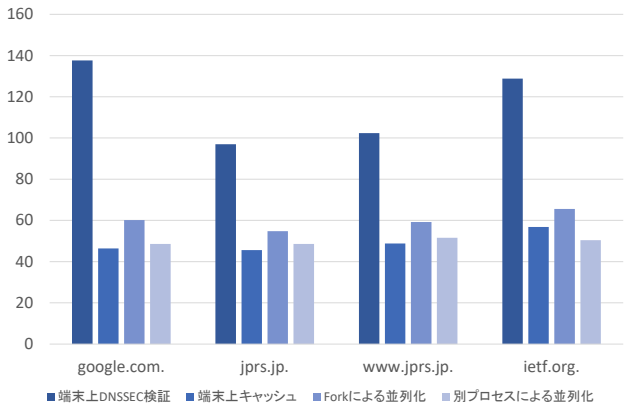


図 8　1 台目の名前解決にかかる時間 [ms]

表 1　1 台目の名前解決にかかる時間 [ms]

| | 端末上検証 | 端末キャッシュ | Fork | 別プロセス |
|---|---|---|---|---|
| (1) | 137.6 | 46.4 | 60.2 | 48.6 |
| (2) | 97.0 | 45.6 | 54.8 | 48.6 |
| (3) | 102.4 | 48.8 | 59.2 | 51.6 |
| (4) | 128.8 | 56.8 | 65.6 | 50.4 |

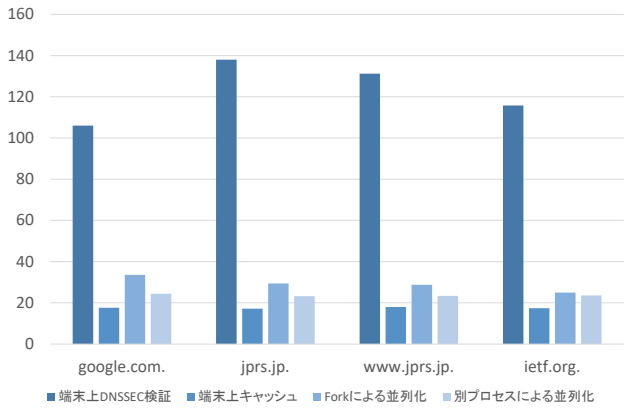


図 9　2 台目の名前解決にかかる時間 [ms]

表 2　2 台目の名前解決にかかる時間 [ms]

| | 端末上検証 | 端末キャッシュ | Fork | 別プロセス |
|---|---|---|---|---|
| (1) | 106.0 | 17.6 | 33.6 | 24.4 |
| (2) | 138.0 | 17.2 | 29.4 | 23.2 |
| (3) | 131.2 | 18.0 | 28.8 | 23.4 |
| (4) | 115.8 | 17.4 | 25.0 | 23.6 |

検証システムは DNSSEC 検証を開始する前にプロセス間通信で DNS パケットを名前解決専用プロセスに送信しておき，名前解決専用プロセスは端末上検証システムと独立してスタブリゾルバに応答を送信する．今回プロセス間通信には Perl モジュールの IO::Socket::INET[8] の UDP 通信を用いた．

### 5.2　性能評価

前節の設計に従って実装を行い，4 つのクエリ名で性能評価を行った．測定方法は dig コマンドを用いて名前解決を行い，出力結果の Query time で時間計測を行った．名前解決を行ったクエリ名は DNSSEC 検証に対応していないドメイン「google.com.」，DNSSEC 検証に対応した「jprs.jp.」，jprs.jp. のサブドメインである「www.jprs.jp.」，DNSSEC に対応して，かつ，外部名（ゾーン外のネームサーバを利用しているドメイン名）を利用している「ietf.org.」ですべて IN クラス，A レコードで問合せを行った．DNSSEC ではドメイン名の階層が増えるほど検証に時間がかかる傾向があるため，また，外部名を利用している場合の時間比較のために 3 つのクエリ名で計測している．

本実験では 2 台の端末を利用して，名前解決を行った．1 台目の結果を図 8 と表 1 に示す．なお，図表の簡略化のため，名前を省略しているが，左から端末上検証システムでの DNSSEC 検証による名前解決の時間，端末上検証システムのキャッシュ機能による名前解決の時間，Fork による並列な問合せを行った名前解決の時間，名前解決専用プロセスによる並列な問合せを行った名前解決の時間を表している．表は上から「google.com.」，「jprs.jp.」，「www.jprs.jp.」，「ietf.org.」を表している．

初回の問合せには 1 番左の端末上検証システムでの DNSSEC 検証となり，約 100-150ms の時間が必ずかかる．同一の内容の問合せの場合は，左から 2 番目の端末上検証システムのキャッシュが有効となり，約 40ms に短縮することができる．右二つの結果は 1 台目では端末上検証システムのキャッシュが無効になり，かつ，DNSSEC 対応キャッシュサーバのキャッシュが有効な場合に得られる．

1 台目の結果を図 9 と表 2 に示す．

2 台目では 1 台目がすでに問合せを行っているので，初回の問合せでも右二つの結果を利用することができる．そのため，約 100-150ms の時間がかかる端末上検証システムでの DNSSEC 検証の結果を待つ必要がなく，初回の問合せでも 20-30ms 程度名前解決を終了させることができる．

## 6.　むすび

本研究では端末上検証システムで 2 種類のキャッシュ DNS サーバ（DNSSEC 対応キャッシュサーバと端末上検証用キャッシュサーバ）に問い合わせを行うことで検証回数の減少，名前解決の高速化を実現した．ユーザは並列した問い合わせをする端末上検証システムを利用することで安全かつ高速な名前解決を行うことができる．実際に利用する場合，web ページなどでは多くの名前解決が生じるため，大量の DNSSEC 検証を行う必要があるが，キャッシュのヒット率を上げることができれば，速やかな web ページの表示に繋がり，ユーザにストレスを与える機会を減らすことができる．

また，二つのキャッシュ DNS サーバに問い合わせを行うことで，名前解決に冗長性を持たせることができる．片方の名前解決でタイムアウトが生じても，もう片方の名

前解決で DNS 応答を取得することができる．今回の端末では端末上 DNSSEC 検証でタイムアウトを起こすことはなかったが，クライアント端末の性能や OS，CPU の利用状況，ネットワークの性能・利用状況等によっては端末上 DNSSEC 検証もタイムアウトを起こす可能性がある．その場合，本研究では近くにあるキャッシュ DNS サーバからの応答をすぐに利用することができる．

今後の展望としては，名前解決専用別プロセスによる並列化において動作の柔軟性を持たせていくことや DNSSEC 対応キャッシュサーバから応答を取得した場合に，端末上検証システムでの DNSSEC 検証回数を減らすために，端末上検証システムでの DNSSEC 検証を停止する手法の検討が挙げられる．

## 参考文献

[1] Jin, Y., Kakoi, K., Yamai, N., Kitagawa, N. and Tomoishi, M.: A Client Based DNSSEC Validation System with Adaptive Alert Mechanism Considering Minimal Client Timeout, *IEICE Transactions on Information and Systems*, Vol. E100-D, No. 8, pp. 1751–1761 (2017).
[2] 藤原和典：ソースポートランダマイゼーションとは何ですか？：DNS Tips - @ IT，ITmedia Inc.（オンライン），入手先 ⟨https://www.atmarkit.co.jp/ait/articles/1501/05/news015.html⟩（参照 2021-01-24）.
[3] Deccio, C.: Maintenance, mishaps and mending in deployments of the domain name system security extensions (DNSSEC), *International Journal of Critical Infrastructure Protection*, Vol. 5, No. 2, pp. 98–103 (2012).
[4] Kakoi, K., Jin, Y., Yamai, N., Kitagawa, N. and Tomoishi, M.: Cache Function Activation on A Client Based DNSSEC Validation and Alert System by Multithreading, *Proceedings of 2017 IEEE 41st International Conference on Computer Software and Applications (COMPSAC 2017) Workshops*, pp. 37–42 (2017).
[5] Brown, R.: Net::DNSServer::Proxy - Forwards requests to another DNS server - metacpan.org, CPAN (online), available from ⟨https://metacpan.org/pod/Net::DNSServer::Proxy⟩ (accessed 2021-01-05).
[6] Marzot, G. S.: Net::DNS::SEC::Validator - interface to libval(3) and related constants, structures and functions. - metacpan.org, CPAN (online), available from ⟨https://metacpan.org/pod/Net::DNS::SEC::Validator⟩ (accessed 2021-01-05).
[7] Brown, R.: Net::DNSServer - Perl module to be used as a name server - metacpan.org, CPAN (online), available from ⟨https://metacpan.org/pod/Net::DNSServer⟩ (accessed 2021-01-05).
[8] Rinaldo, T.: IO::Socket::INET - Object interface for AF_INET domain sockets - metacpan.org, CPAN (online), available from ⟨https://metacpan.org/pod/IO::Socket::INET⟩ (accessed 2021-01-05).