DNSOPS.JP BoF 2015-11-19

nginxを利用した DNS over TLS 対応フルリゾルバの作り方

(株)ハートビーツ　滝澤 隆史

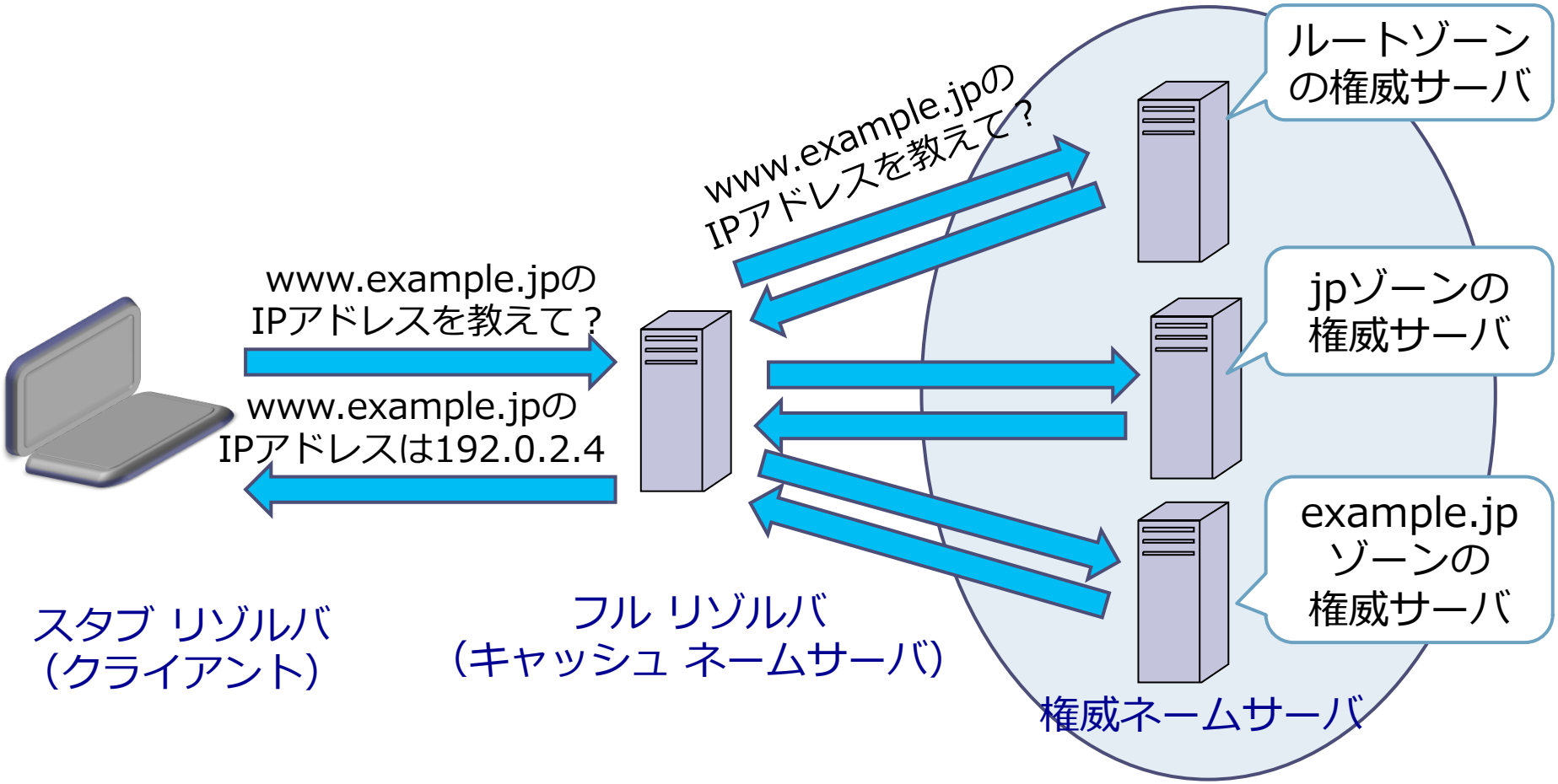
私は誰

- 氏名: 滝澤 隆史 @ttkzw
- 所属: 株式会社ハートビーツ 
  - ウェブ系のサーバの構築・運用や
    24時間365日の有人監視をやっている会社
  - いわゆるMSP(マネージド サービス プロバイダ)
- 日本Unboundユーザー会
  - Unbound/NSDの文書の翻訳
- DNSをネタとして遊んでいるおじさんです。
  - 「DNS RFC系統図」を作っています。

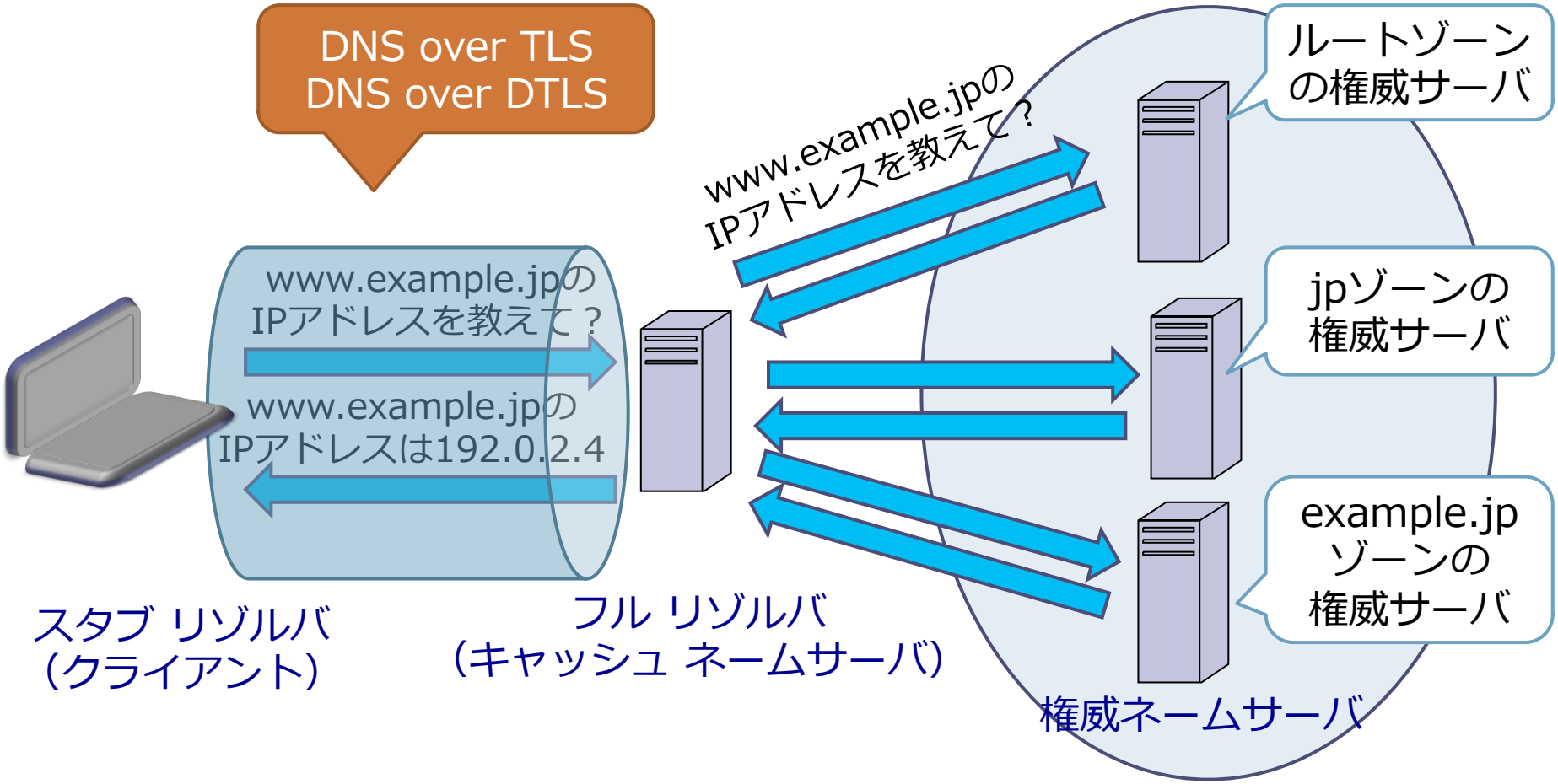
DNS Privacy

- IETFのdpriveワーキンググループ
  - DNS PRIVate Exchange (DPRIVE)
- DNSトランザクション中のプライバシーを守る仕組み
- RFC 7626: DNS Privacy Considerations
- I-D: DNS over TLS: Initiation and Performance Considerations
- I-D: DNS over DTLS (DNSoD)

DNSトランザクション



DNSトランザクションの暗号化



DNS over TLS対応リゾルバ

- draft-ietf-dprive-dns-over-tls-01
  - 8. Implementation Status
  - 8.1. Unbound
  - 8.2. ldns
  - 8.3. digit
  - 8.4. getdns

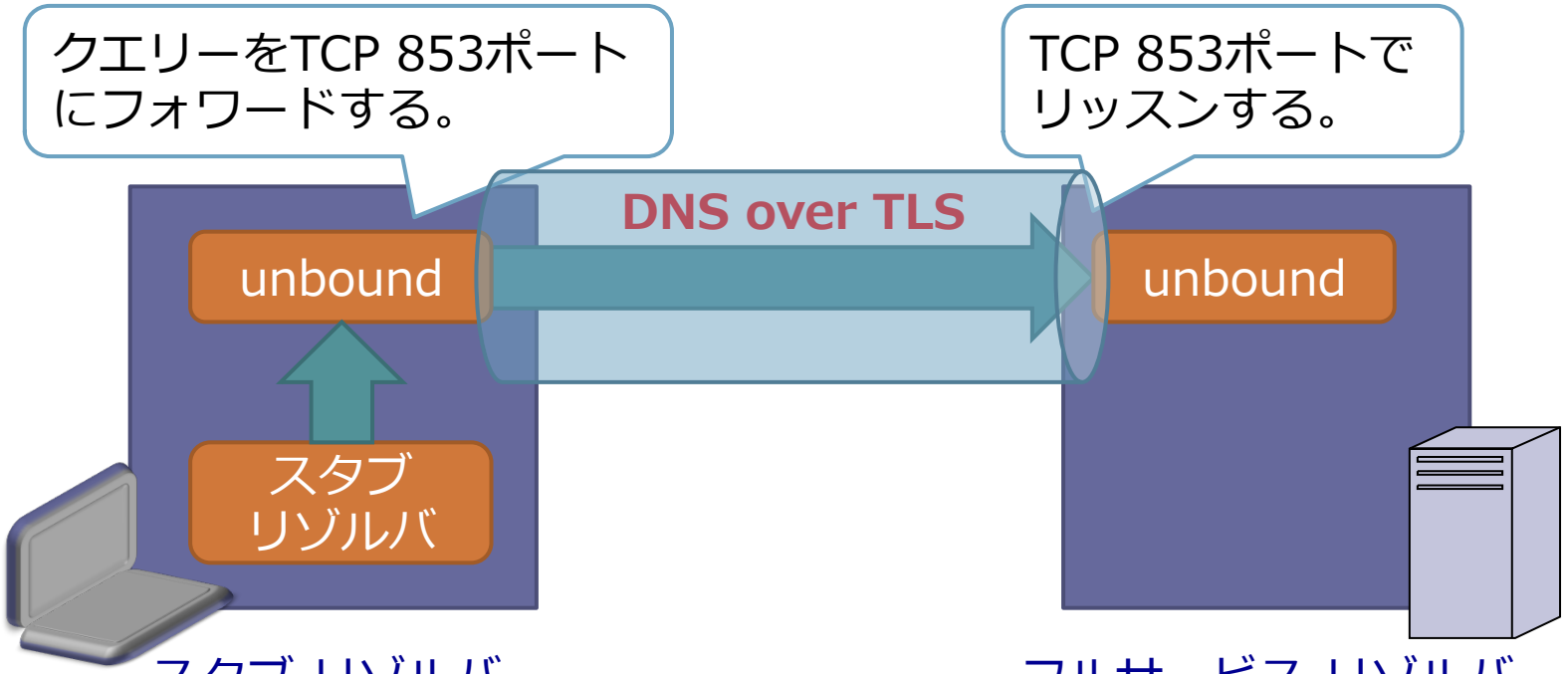
DNS over TLS対応リゾルバ

- draft-ietf-dprive-dns-over-tls-01
  - 8. Implementation Status
  - 8.1. Unbound
  - 8.2. ldns
  - 8.3. digit
  - 8.4. getdns

DNS over TLS対応リゾルバ

- Unbound
  - ssl-port, ssl-upstream機能
    - DNS over TLSに対応

Unboundのssl-upstream機能の利用例




```
server:
  ssl-upstream: yes
forward-zone:
  name: "."
  forward-addr: 192.0.2.1@853
```

```
server:
  interface: 0.0.0.0@853
  ssl-service-key: "/etc/unbound/unbound_server.key"
  ssl-service-pem: "/etc/unbound/unbound_server.pem"
  ssl-port: 853
```

Unboundならできるのは
わかった。
では、フルリゾルバ側で
Unbound以外を
利用したいときには？

そうだ。
nginx（ウェブサーバ）の
streamモジュールを
使おう。

nginx（えんじん えっくす）とは

- HTTP server
- reverse proxy server (HTTP)
- mail proxy server (SMTP, POP3, IMAP)
- TCP proxy server
- SSL/TLS termination対応
- HTTP/2対応
- ウェブサーバのシェアはApache HTTP Serverに続いて2位。（Netcraftの調査。Market share of active sites）
  - http://news.netcraft.com/archives/2015/10/16/october-2015-web-server-survey.html

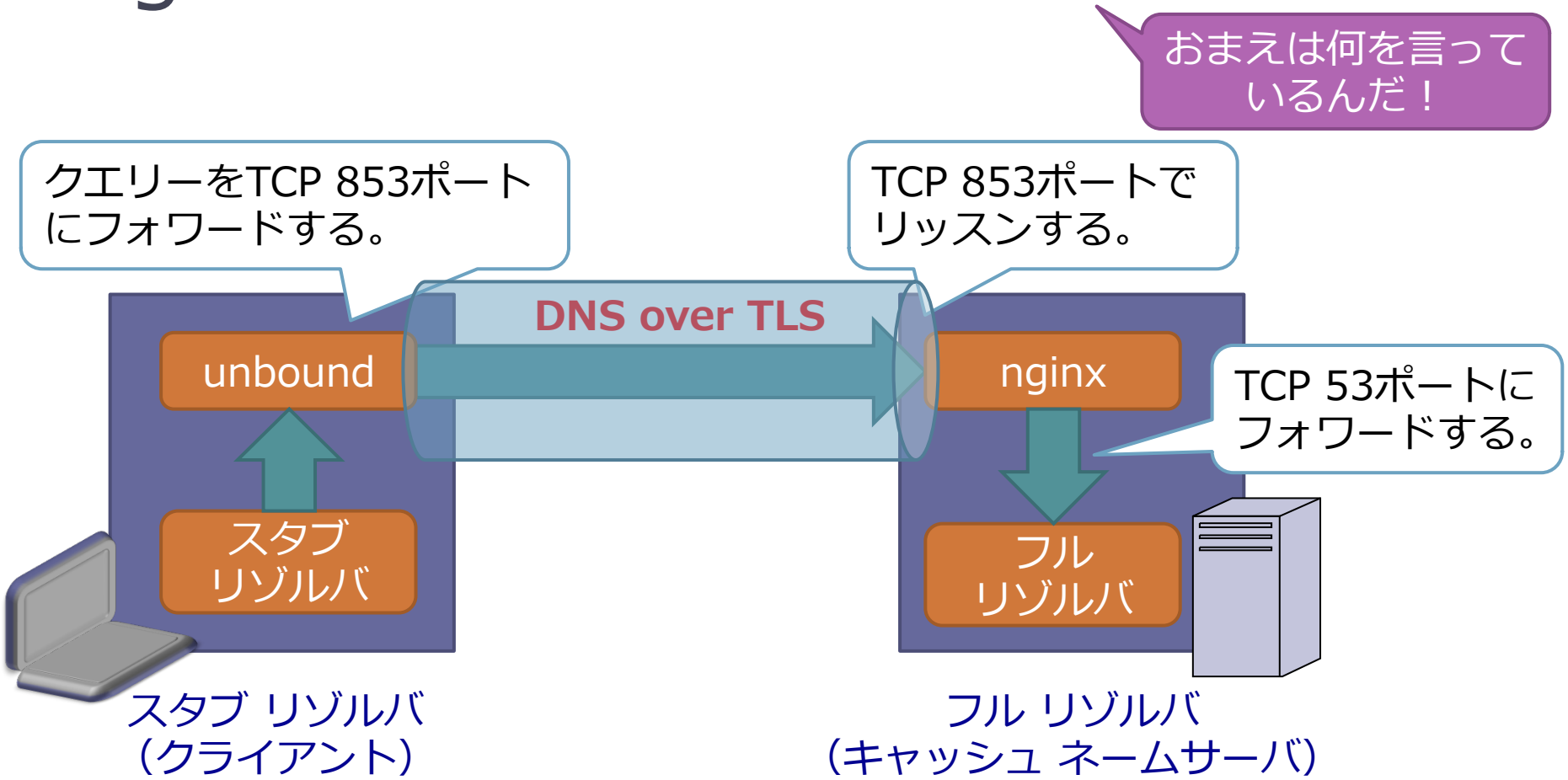
nginx（えんじん えっくす）とは

- HTTP server
- reverse proxy server (HTTP)
- mail proxy server (SMTP, POP3, IMAP)
- TCP proxy server
- SSL/TLS termination対応
- HTTP/2対応
- ウェブサーバのシェアはApache HTTP Serverに続いて2位。（Netcraftの調査。Market share of active sites）
  - http://news.netcraft.com/archives/2015/10/16/october-2015-web-server-survey.html

なぜ、nginxなのか

- streamモジュール(TCP proxy機能)は
  TLS termination対応
- そこに、nginxがあったから
- 今時の実装（イベント駆動、ノンブロッキング
  非同期I/O）

nginxでDNSパケットを転送



nginx.conf

```
stream {
    upstream backend {
        server 127.0.0.1:53;
    }

    server {
        listen *:853 ssl;
        listen [::]:853 ssl;

        ssl_prefer_server_ciphers on;
        ssl_protocols       TLSv1.2;
        ssl_ciphers         ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-
SHA:ECDHE-RSA-AES128-SHA;
        ssl_certificate     /etc/nginx/tls/server.crt;
        ssl_certificate_key /etc/nginx/tls/server.key;

        ssl_session_cache   shared:DNS:5m;
        ssl_session_timeout 5m;
        ssl_session_ticket_key /etc/nginx/tls/ticket.key;
        ssl_session_tickets on;

        proxy_pass backend;
    }
}
```

フルリゾルバを指定。
複数個記述することで
負荷分散できる。

リッスンするポートを
TCP 853に指定。
SSL/TLSを有効に。

アップストリーム先の
指定。

ベンチマーク

ベンチマーク環境

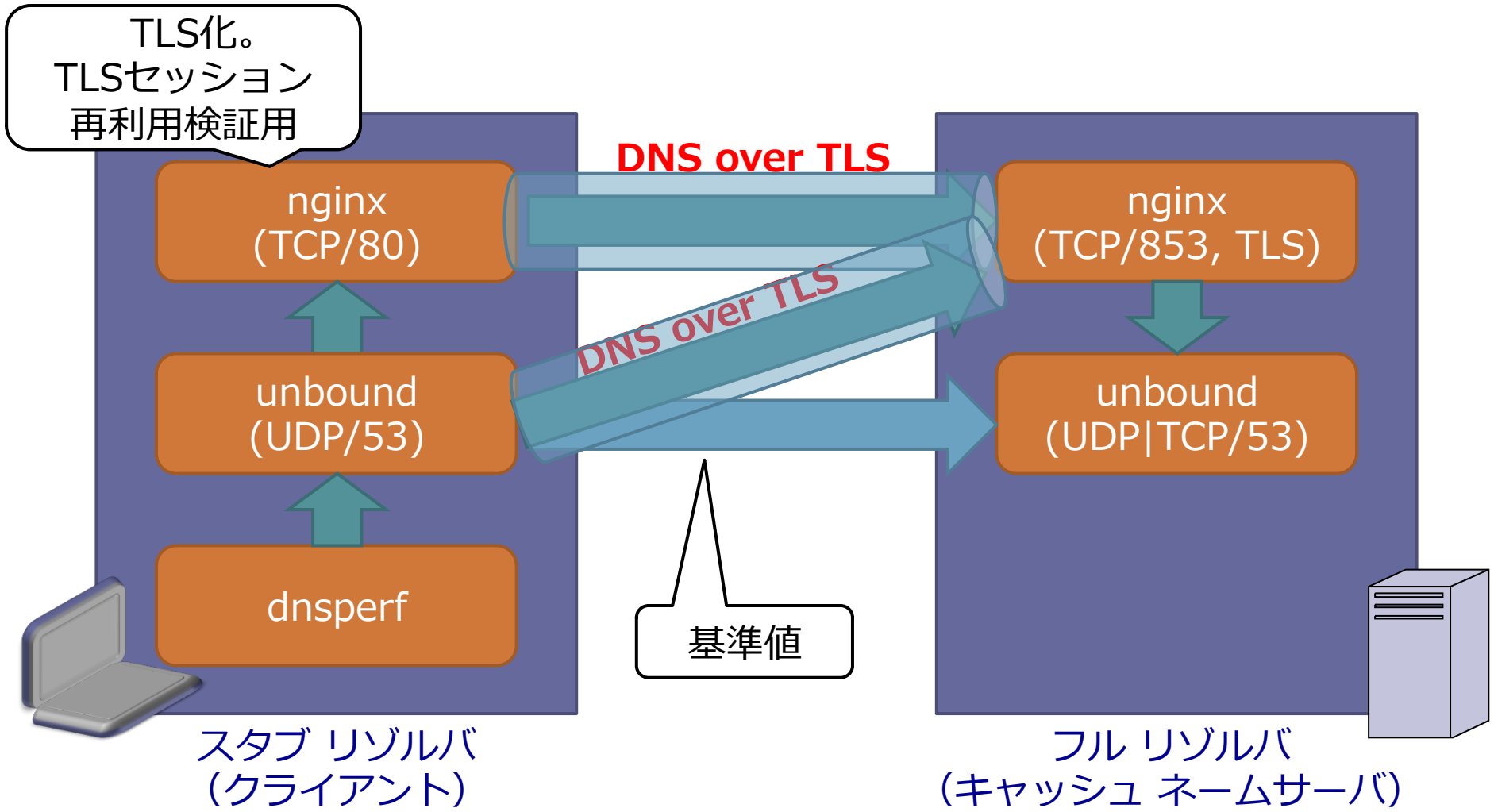
- ベンチマークツール
  - dnsperf 2.0.0.0
    - -c 10 -n 200000
    - "localhost A"を問い合わせ。
- サーバ
  - AWS EC2 c4.large
  - クライアントとサーバは、同一リージョン同一アベイラビリティゾーンであるため、ネットワークレイテンシは小さい。
- 結果について
  - 3回計測し、その平均値を有効数字2桁で四捨五入。
  - 今日、思いつきでベンチマークをやったので精度はよくない。
    - さらに、クラウド上の仮想マシンですし。
  - クライアント側がボトルネックになっていて負荷があまりかけられていない。
    - でも、CPU負荷は数%。CPU Usageはstealが数%。コンテキストスイッチが6桁台。クラウド環境な時点で負け。
  - そのため、相対的な大雑把な比較。

ベンチマーク環境

- クライアント側のunbound経由で対象サーバにクエリーを行う。
  - キャッシュ無しのフォワーダーとして動作
    - cache-max-ttl: 0
  - localhostに対する回答を返せるので無効化
    - local-zone: "localhost" transparent
- クライアント側のnginx経由でTLS化
  - TLSセッション再利用の有無の違いも計測
  - proxy_ssl on;
  - proxy_ssl_session_reuse on;

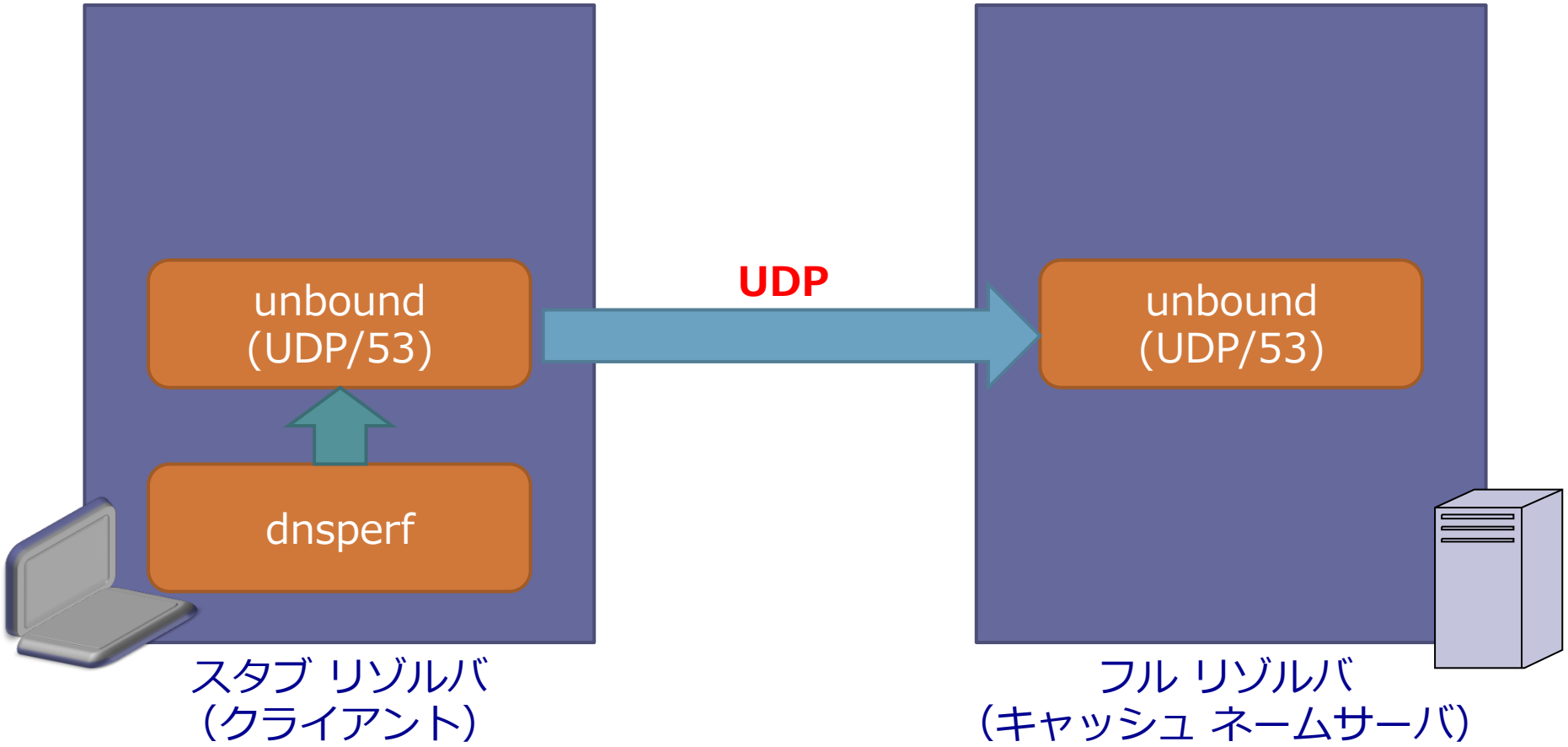
```
stream {
    upstream backend {
        server 10.0.0.38:853;
    }

    server {
        listen *:80;

        proxy_ssl on;
        proxy_ssl_session_reuse on;
        proxy_ssl_protocols TLSv1.2;
        proxy_pass backend;
    }
}
```

ベンチマーク環境


TLS化。
TLSセッション
再利用検証用
DNS over TLS
nginx
(TCP/80)
nginx
(TCP/853, TLS)
DNS over TLS
unbound
(UDP/53)
unbound
(UDP|TCP/53)
dnsperf
基準値
スタブ リゾルバ
（クライアント）
フル リゾルバ
（キャッシュ ネームサーバ）

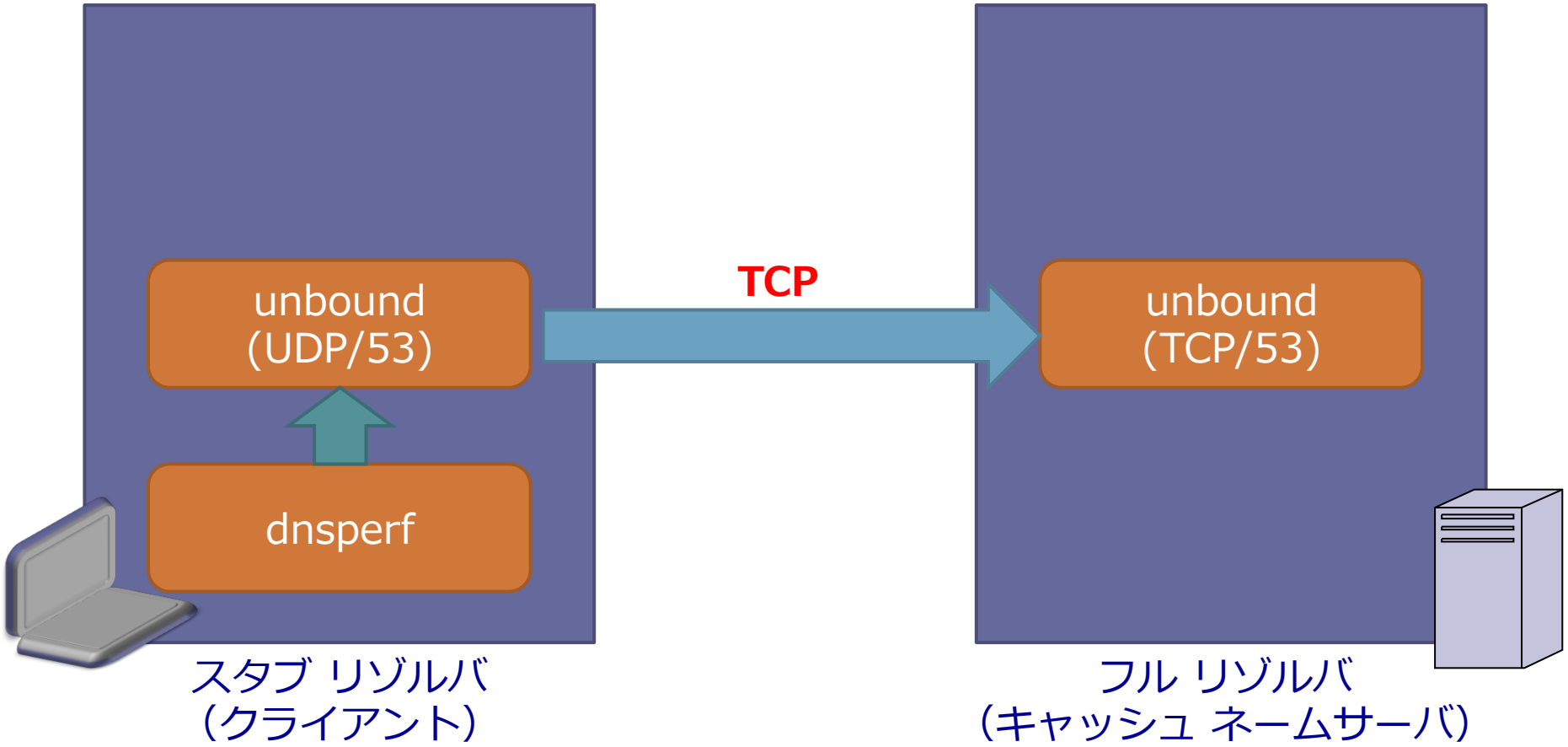
項番	qps
1	82000

1. 基準値（UDP）



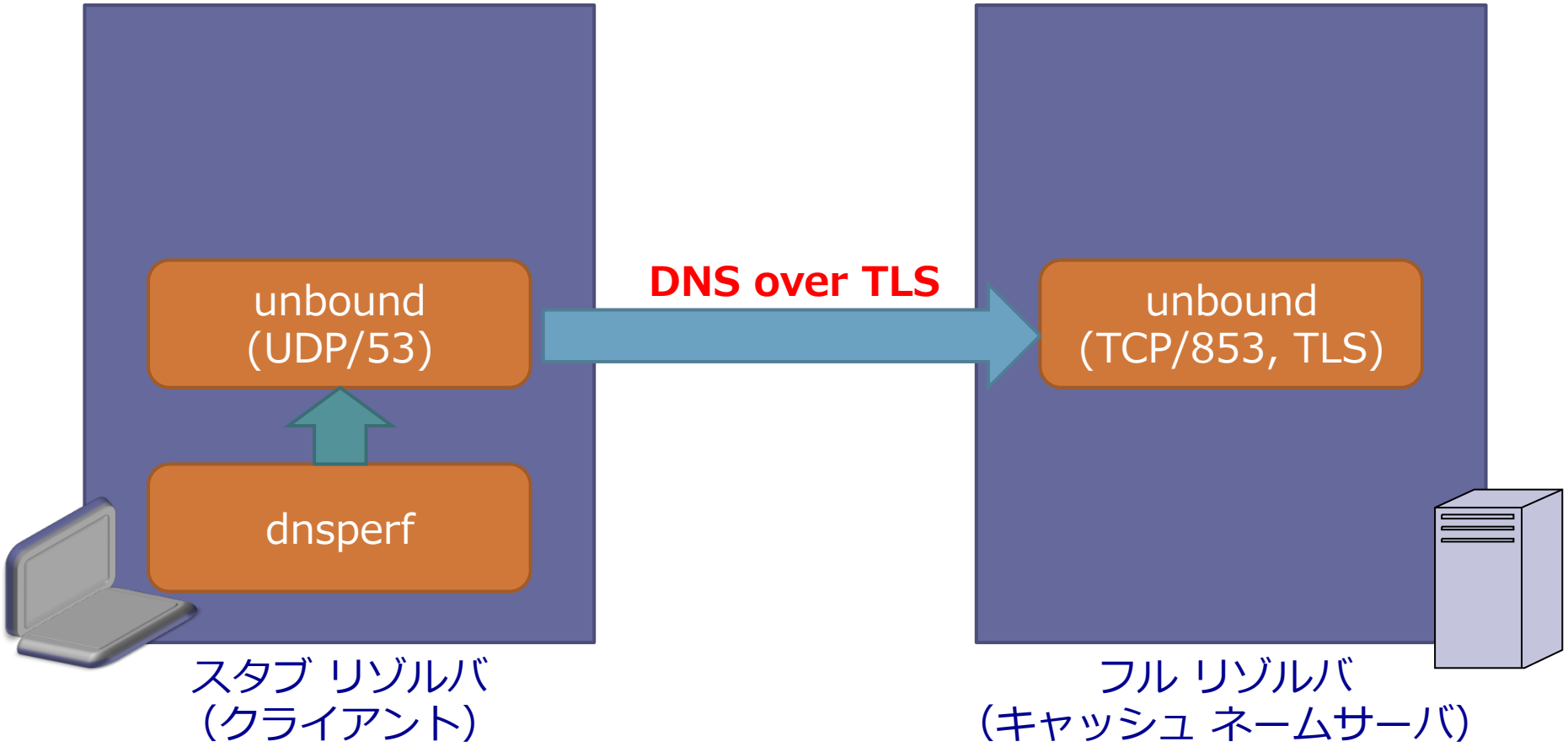
2. 基準値（TCP）

項番	qps
1	82000
2	71000



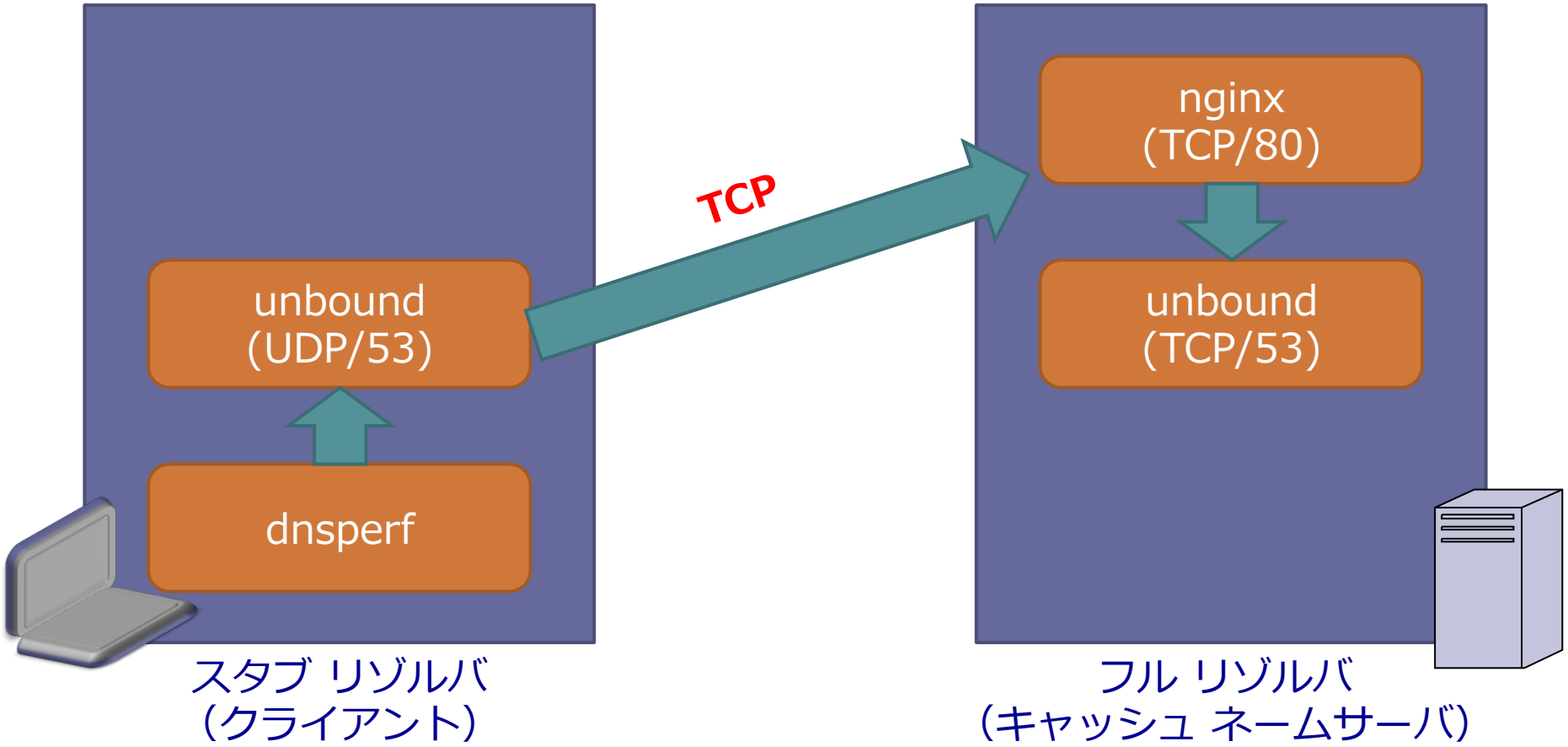
3. 基準値 (DNS over TLS)

項番	qps
1	82000
2	71000
3	1000



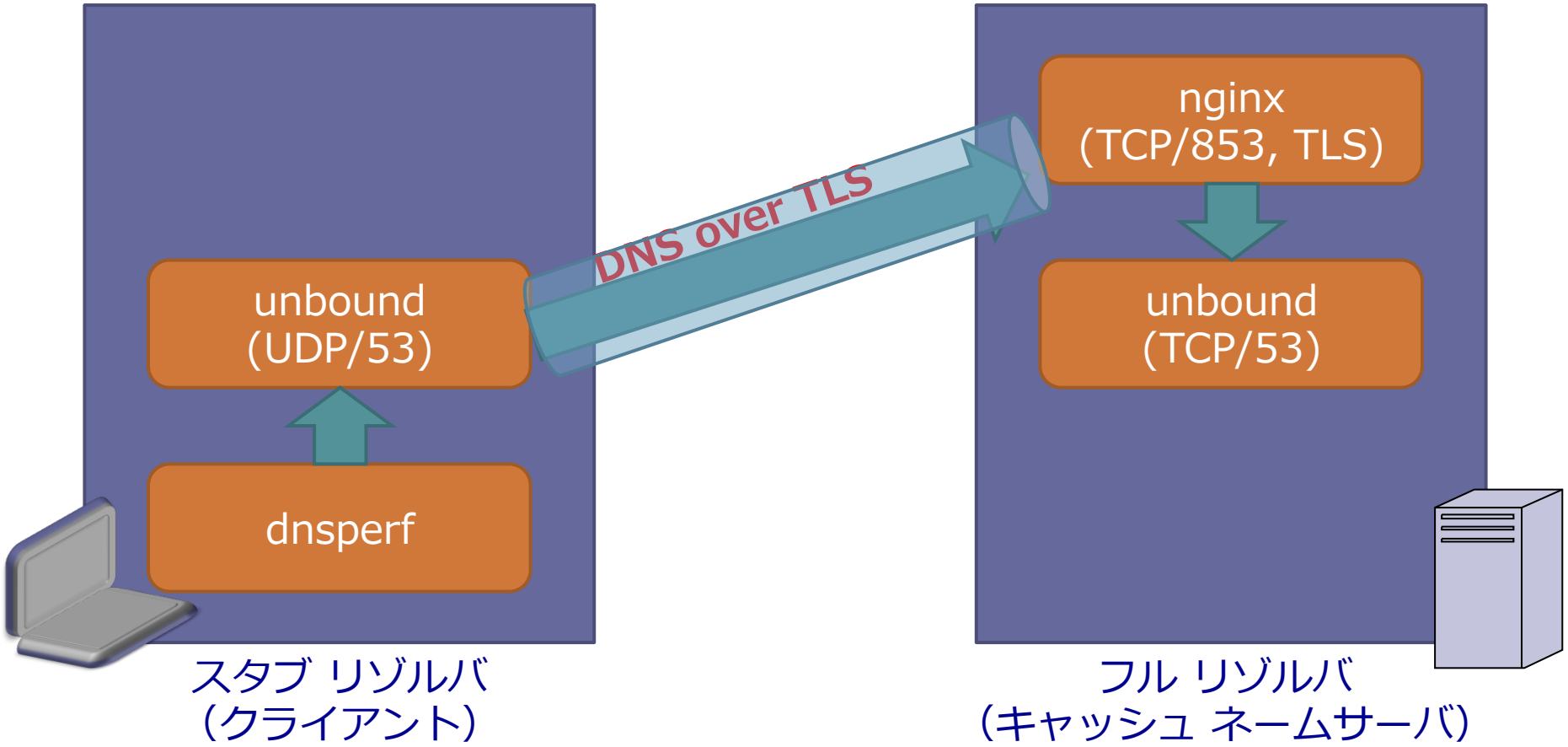
4. nginx(TLSなし)

項番	qps
1	82000
2	71000
4	69000



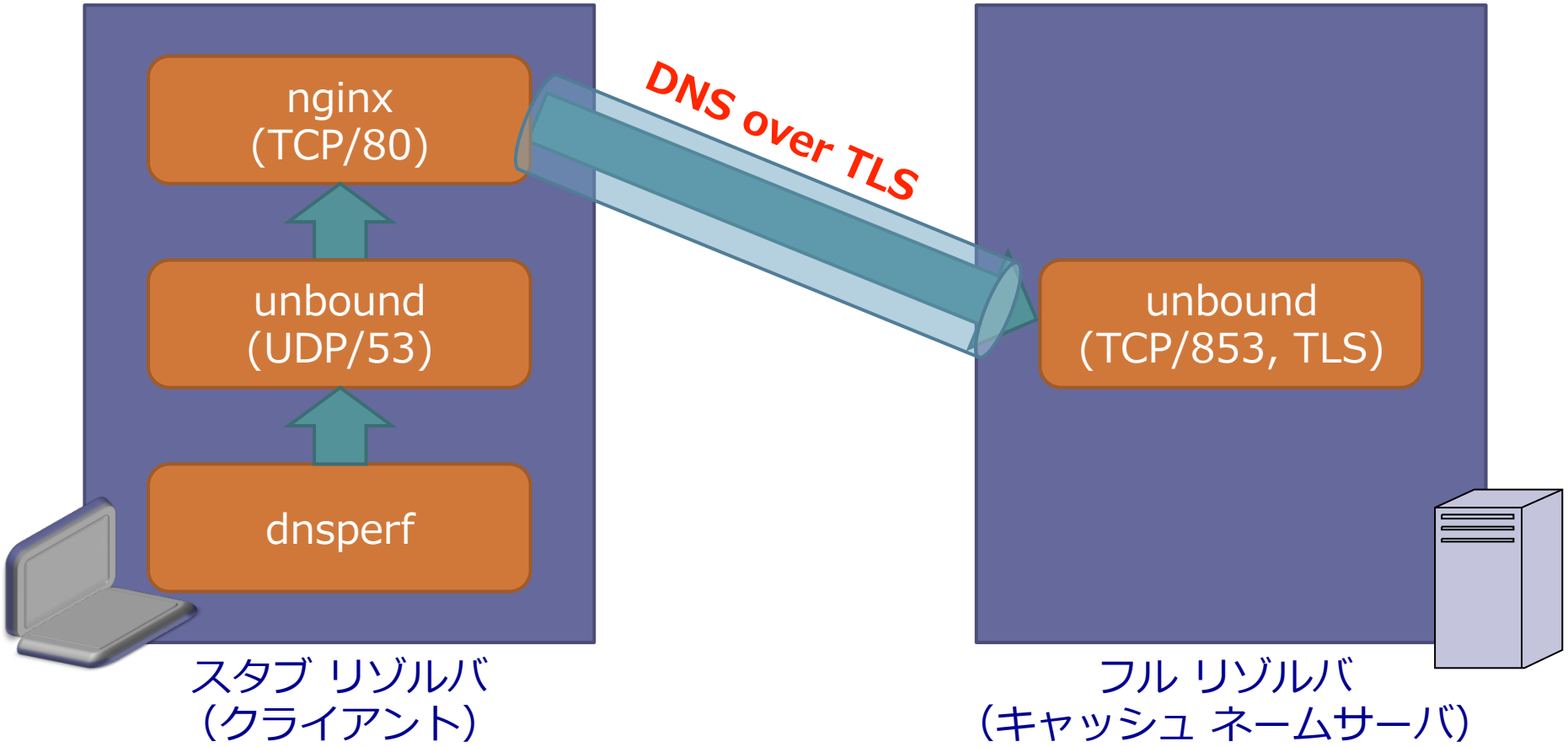
5. nginx(TLS)

項番	qps
1	82000
3	1000
5	2100



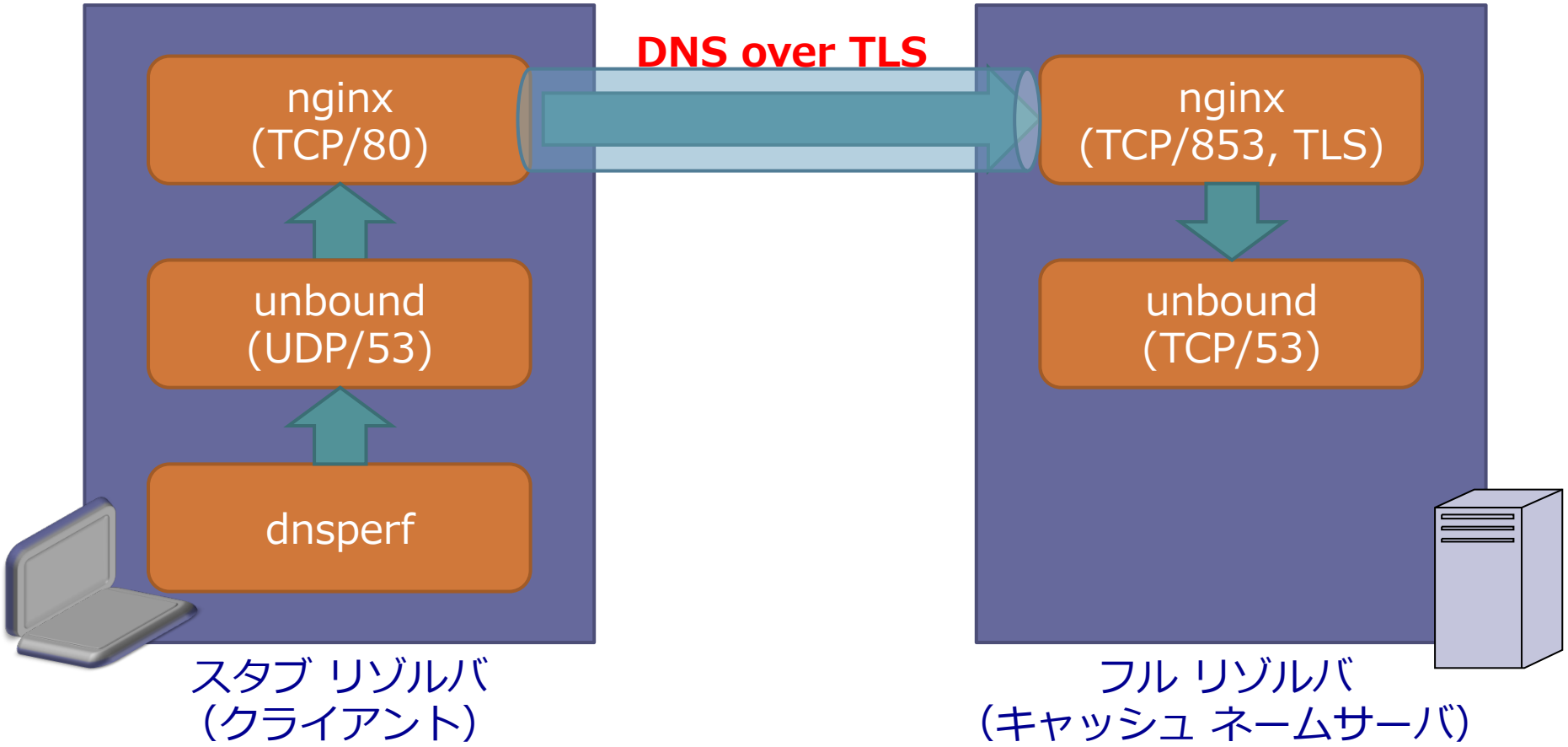
6. クライアント側nginx(TLS)、サーバ側unbound(TLS)

項番	qps
3	1000
5	2100
6	2400



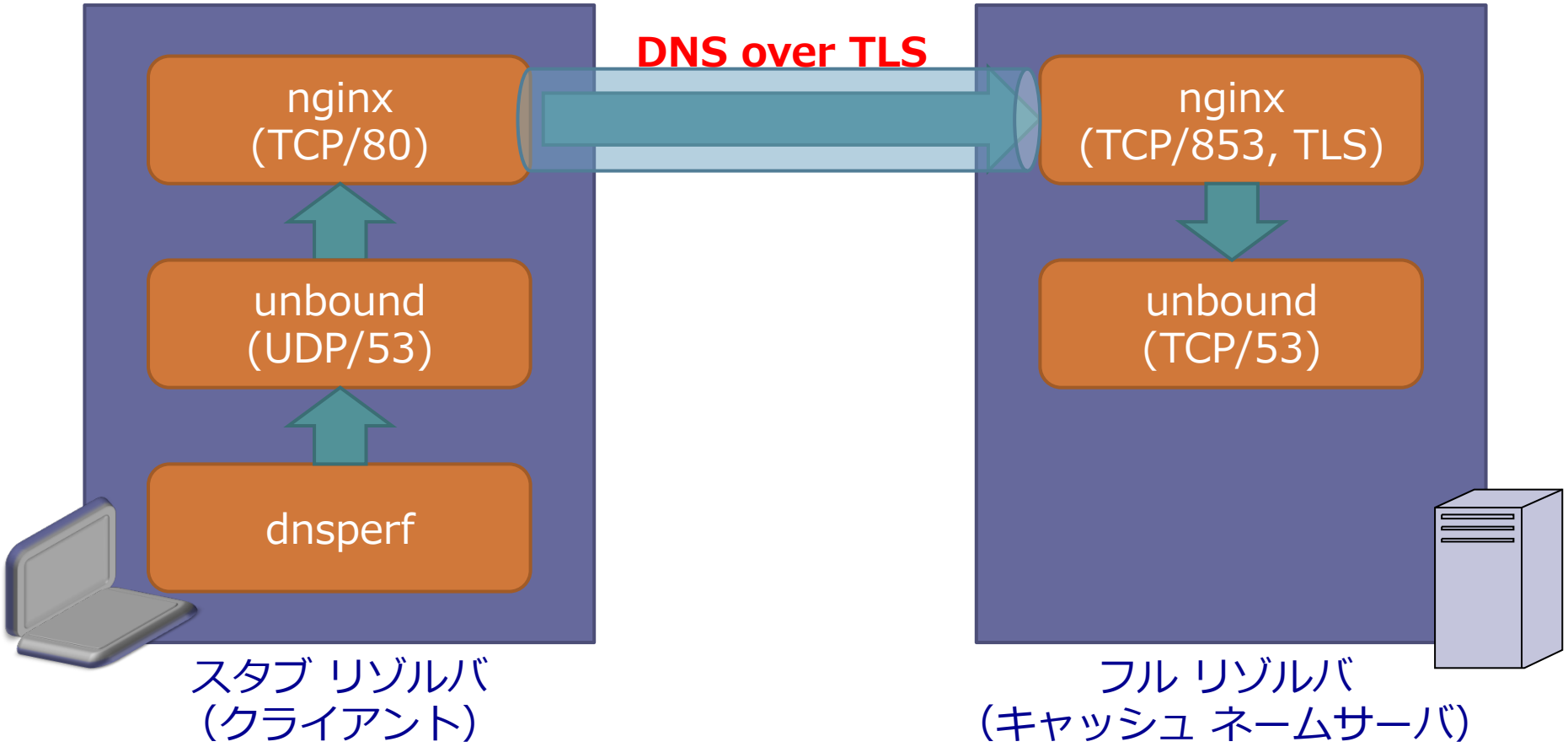
7. クライアント側nginxでTLS化

項番	qps
1	82000
3	1000
7	15000



8. クライアント側nginxでTLS化（TLSセッション再利用あり）

項番	qps
1	82000
7	15000
8	44000



結果

有効数字2桁
として四捨五入

項番	説明	プロトコル	qps
1	基準値([unbound] – [unbound])	UDP	82000
2	基準値([unbound] – [unbound])	TCP	71000
3	基準値([unbound] – [unbound])	TLS	1000
4	nginxでTCP転送 ([unbound] – [nginx – unbound])	TCP	69000
5	nginxでTLS termination ([unbound] – [nginx – unbound])	TLS	2100
6	クライアント側をnginxでTLS化 ([unbound – nginx] – [unbound])	TLS	2400
7	クライアント側をnginxでTLS化 ([unbound – nginx] – [nginx – unbound])	TLS	15000
8	クライアント側をnginxでTLS化(セッション再利用) ([unbound – nginx] – [nginx – unbound])	TLS	44000

ベンチマークのまとめ

- 負荷が十分にかけられていない状況では、UDPとTCPによる性能の違いは大きくはない。
  - ベンチマーク環境のネットワークレイテンシーが小さいのでより差が出にくい？
- unboundのTLS周り性能はよくない。
- nginxのTLS周りの性能はよい。
  - TLS termination
  - TLS proxy

ベンチマークのまとめ

- TLSのハンドシェイクのオーバーヘッドは大きい。
  - TLSセッション再利用を行わないと性能がかなり落ちる。
- TLS周りでは次の機能のどちらかを利用しないと性能的には辛い。
  - TLS session cache
  - TLS Session Resumption / TLS Session Tickets
    - RFC 5077 Transport Layer Security (TLS) Session Resumption without Server-Side State
- draft-ietf-dnsop-5966bis (DNS Transport over TCP - Implementation Requirements)で性能は改善するはず？
  - Connection Re-use
  - Query Pipelining
  - TCP Fast Open

おわり