InternetWeek2018
DNS Day
DNSデータ収集と可視化

2018/11/29版

株式会社インターネットイニシアティブ
其田　学

Manabu Sonoda
其田　学

所属

株式会社インターネットイニシアティブ
日本DNSオペレーターズグループ 幹事

経歴

- 某ISPでL1からL8までのフルスタックエンジニア
- 現在IIJ勤務
  - IIJの回線系フルリゾルバの設計、構築、運用
  - IIJのDNSアウトソーシング系サービスの権威DNSの設計、構築、運用
  - D.DNS.JPの構築、運用
  - コミュニティ活動、啓蒙活動（イマココ）

DNSの運用に軸足を置くネットワークエンジニアのつもりです。。

DNSメッセージベースのデータ

- 今回のフォーカスはここ
  - クエリーログ
  - DNSメッセージ

サーバーステータスベースのデータ

- 今回はフォーカス外
  - bindのstatistics-channels
  - Unboundのunbound-cotrol stats
  - など

総クエリ数

- とりあえず見て、いつもと違う形になってたら調査

RCODE別のクエリ数

- 正常に運用できているかの指標

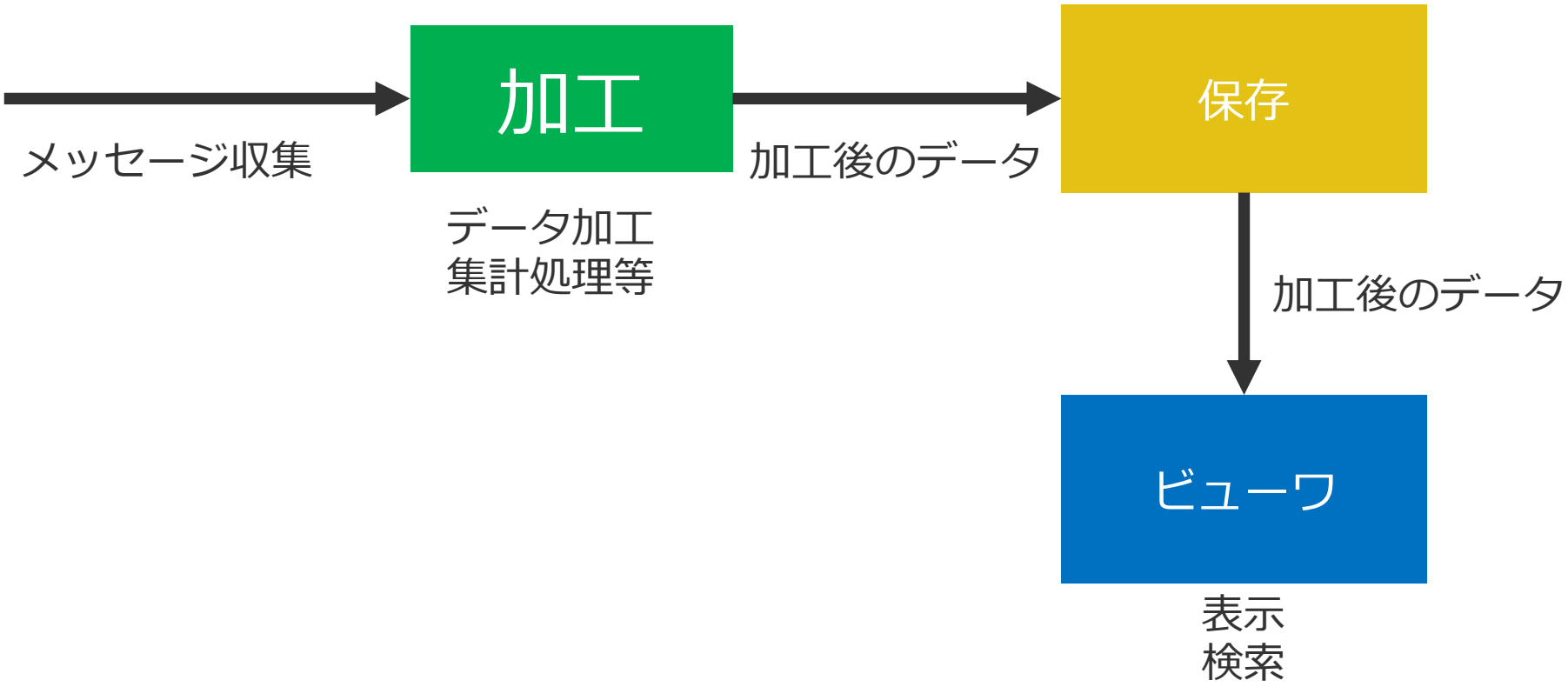
QNAME別のクエリ数

- どんな名前クエリーがどれだけあるのか

クライアントのIPアドレス別のクエリ数

- どのIPアドレスからのクエリーが多いのか見たい

基本的なDNSメッセージの収集、加工、可視化の流れ



DNSデータの収集

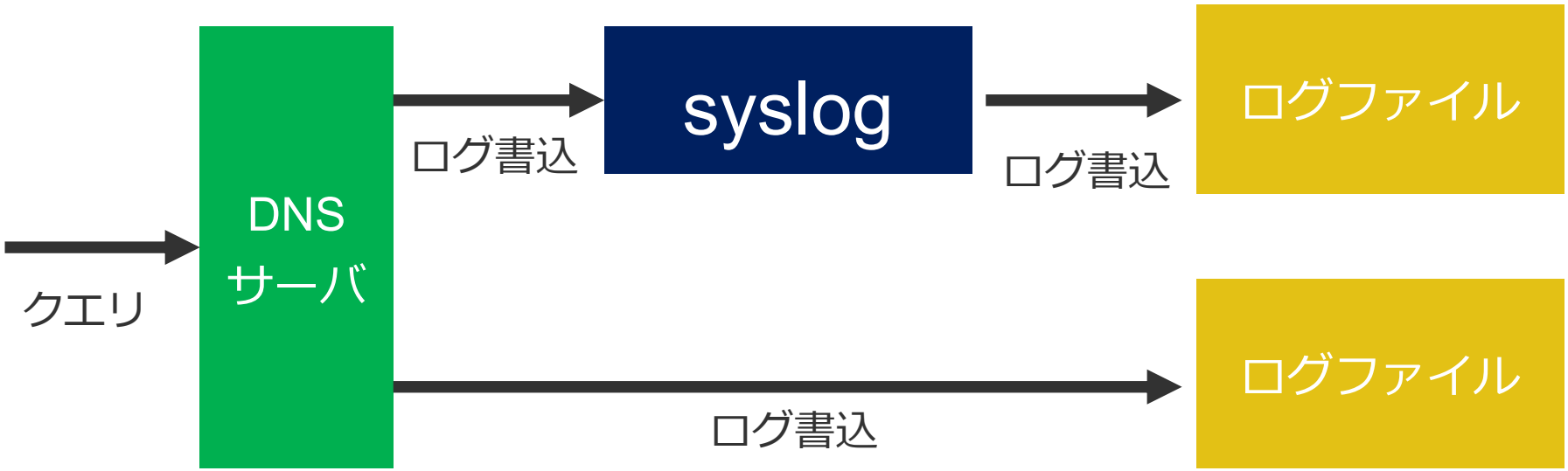
DNSメッセージベースのデータの収集方法

DNSメッセージベースのデータの収集方法

- クエリーログ
- パケットキャプチャ
- DSC
- DNSTAP

ソフトウェアに付随するクエリーログの機能

- フルリゾルバ向け



Unboundの場合

設定内容

```
server:
        log-queries: yes
```

クエリーログ出力

```
Nov 14 18:09:38 unbound-host unbound: [214143:0] info: start of service (unbound 1.8.1).
Nov 14 18:09:39 unbound-host unbound: [214143:1] info: 127.0.0.1 localhost. A IN
Nov 14 18:09:39 unbound-host unbound: [214143:6] info: 127.0.0.1 localhost. A IN
```

Unboundの場合、一般のログと混ざるため、使い物にならない

BIND9の場合

設定内容

```
logging {
  channel querylog {
    file "/var/log/named/query.log" versions 1 size 10M;
    print-time yes;
  };
 category queries {
   querylog;
 };
};
```

クエリーログ出力

```
15-Nov-2018 14:08:58.121 client ::1#53443 (www.iij.ad.jp): query: www.iij.ad.jp IN A +E (::1)
15-Nov-2018 14:09:15.150 client ::1#53446 (www.nic.ad.jp): query: www.nic.ad.jp IN A +E (::1)
```

BIND9の場合、ログ出力先を選べるため、パースはし易い

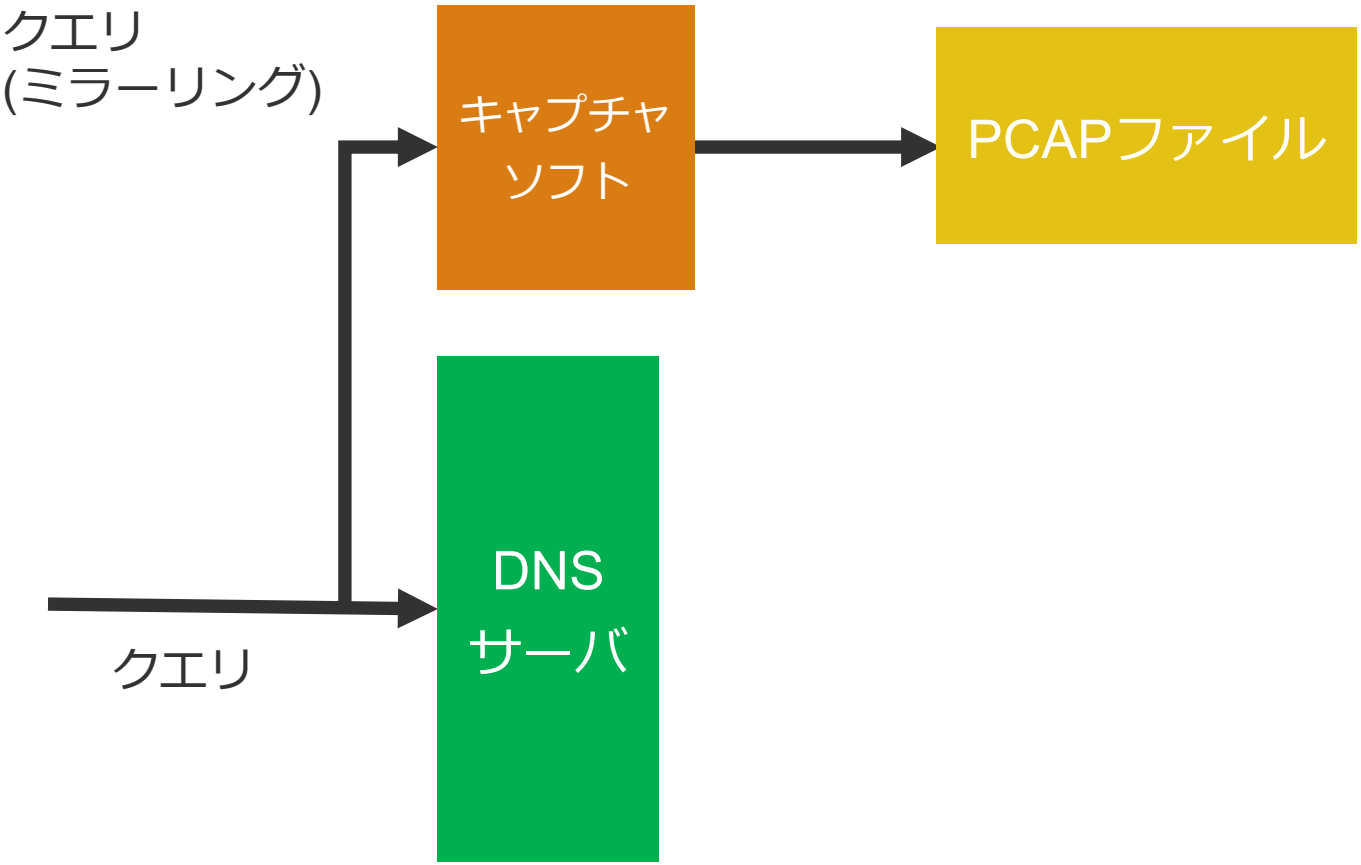
利点

- **Textベースのログが簡単に出せる**
- **Syslog連携して、ただログを貯める場合は楽**

欠点

- **リクエストのログしか出せない**
- **実装によってフォーマットが違うため、パーサーを作ってしまうと、他の実装に手を出しずらくなる。**
  - ダイバシティを考えるとパーサーの実装が倍

tcpdumpやdnscap等で、パケットキャプチャを行う方法



tcpdumpで１分毎にファイルローテートする例

```
# tcpdump -p -s 0
-w /var/tmp/dns.%Y%m%d%H%M.pcap
-G 60
-Z root
-z gzip
-i eth0
port 53
```

-p promiscuous mode OFF
-s 0 取得するパケットサイズを最大に
-w pcapファイル出力先(strftimeの書式が使える)
-G rotateする秒数
-Z ファイルオーナーを指定
-z rotate時にzipをコマンドを起動して圧縮
-i キャプチャーするインターフェース
port53　tcp/udp dst/src port 53番でフィルタリング

みんな大好きwiresharkのコマンドライン版

```
# tshark -i en0 -Y dns
# tshark -r pcap.file -Y dns
Capturing on 'Ethernet'
326   8.784449 192.168.0.56 -> 192.168.0.20 DNS 92 Standard query 0x2f81 A localhost OPT
327   8.785773 192.168.0.20 -> 192.168.0.56 DNS 138 Standard query response 0x2f81 A localhost A
127.0.0.1 NS localhost AAAA ::1 OPT
```

色々な条件でフィルタできて便利ですが、重い
※本番環境でtsharkを叩くのは絶対やめましょう絶対だぞ！

利点

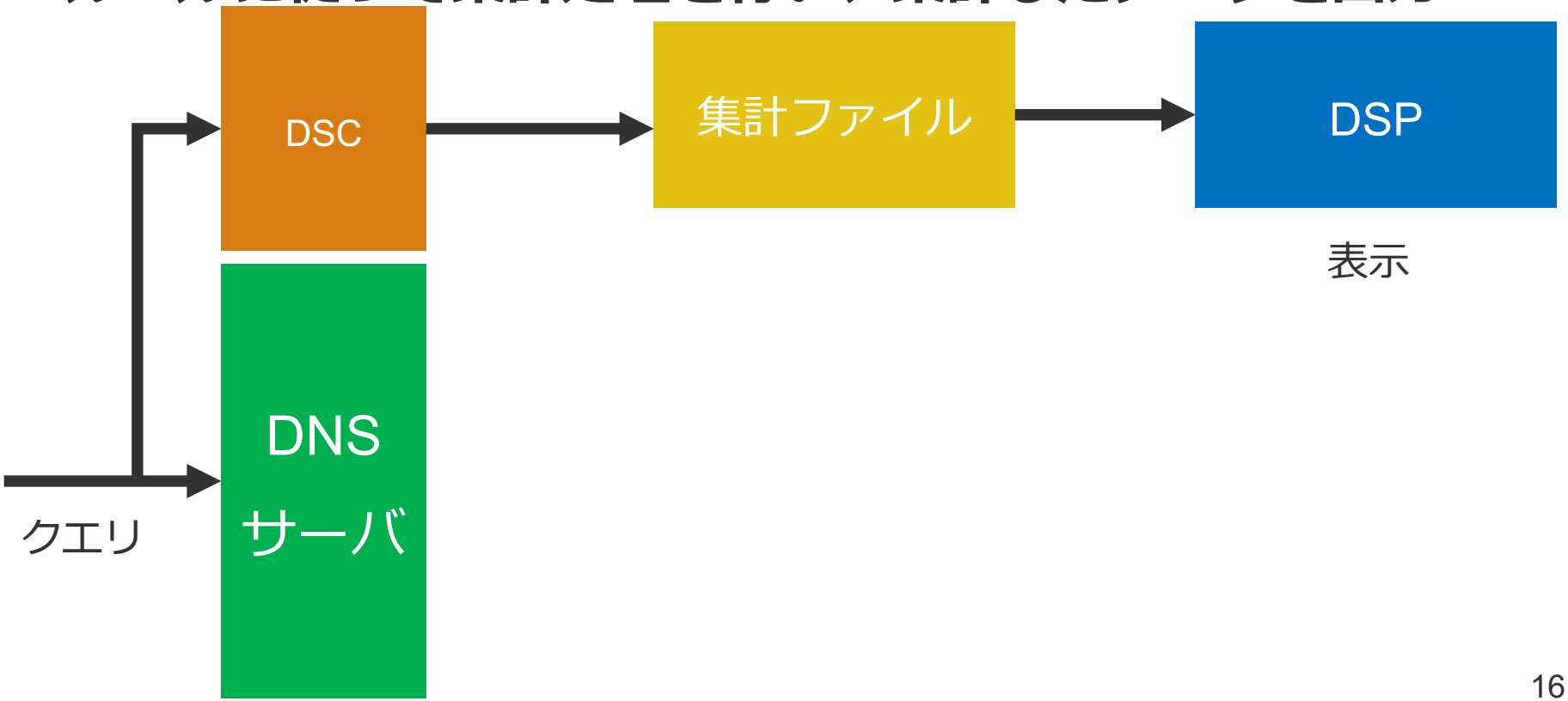
- L2SW上でポートミラーリングすることで、DNSサーバと別ホストでキャプチャができる。
- 同じホスト上でキャプチャする場合でも、DNSソフトウェアに何かするわけでは無いのでバグを踏みにくい。
- DNSソフトウェアの実装に影響されない

欠点

- PCAPファイルは、必要ないレイヤーのデータも含まれるので、容量が大きくなる傾向にある。
- DNS暗号化されたパケットは解読できないため、DNS over TLS, DNS over HTTPSには対応できない

DNS Statistics Collector
https://www.dns-oarc.net/tools/dsc

- **DNS運用で良い意味でとりあえず入れておく物**
- **パケットキャプチャで、DNSメッセージを取得し、**
- **ルールに従って集計処理を行い、集計したデータを出力**



導入

パッケージ https://dev.dns-oarc.net/packages/

基本的な設定

```
Interface eth0;
bpf_program “port 53 and not ( net 監視ホストセグメント)”;
client_v4_mask 255.255.255.0;
client_v6_mask FFFF:FFFF:FFFF:0000:0000:0000:0000:0000;
```

- Interface　キャプチャするインターフェース
- bpf_program　Bpfフィルタ、ここで監視ホストからのクエリーを除去すると、監視パケットが上位に来ることがなくなる
- client_v4_mask　リクエストIPのIPv4のマスク値
  マスクしない場合は255.255.255.255
- client_v6_mask　リクエストIPのIPv4のマスク値
  マスクしない場合は
  FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF

データセット

dataset <dataset名> <type> <label1> <label2> <FILTERS> [PARAMATER] ;

Example:

dataset rcode dns ALL:null Rcode:rcode replies-only;

- dataset名　データセットの名前
- type　データの種類、dns,ip (tcp,udpや、IPv4,IPv6等のlabel用)
- label1,2　いっぱいあるので
- filter　フィルタリングルール

Any	何もしない
queries-only	リクエストのみ
replies-only	レスポンスのみ
nxdomains-only	NXDomainのみ
などなど	

Labelはカウントする対象

例えばRcode:rcodeの場合は、rcode毎にメッセージをカウントします

Label名	役割
All:null	集計しない
Qtype:qtype	Qtype毎に集計
Rcode:rcode	Rcode毎に集計
Qname:qname	Qname毎に集計
TLD:tld	TLD毎に集計
SecondLD:second_ld	2LD毎に集計
ThirdLD:third_ld	3LD毎に集計
ClientAddr:client	クライアントIP毎に集計
ClientSubnet:client_subnet	クライアントIPをマスクしたもの毎に集計

他にもGeoDBを使って国名毎の集計や、AS番号毎の集計もできます。

datasetに関してはデフォルト設定でほぼ事足ります

特によく使うdataset

Dataset 名	label1	label2
qtype	All:null	Qtype:qtype
rcode	All:null	Rcode:rcode
client_subnet	All:null	ClientSubnet:client_subnet
client_addr_vs_rcode	ClientSubnet:client_subnet	Rcode:rcode

デフォルトではコメントアウトされている、
second_ld_vs_rcodeとthird_ld_vs_rcodeもあると役に立つので有効にした方が良いでしょう。

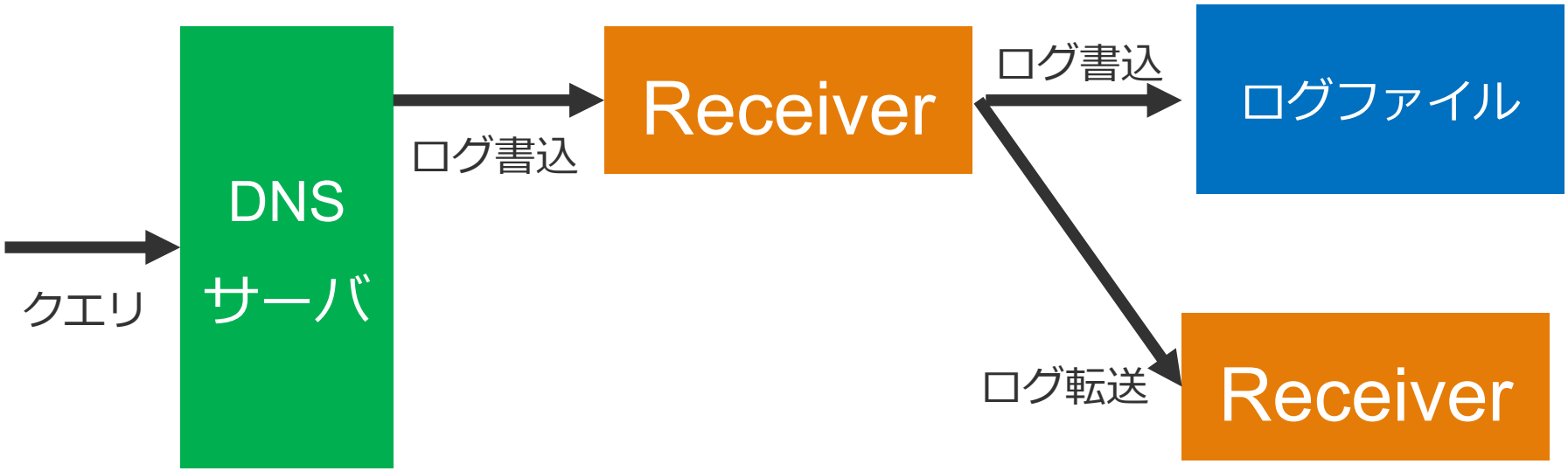
DNS Statistics Collector
パケットキャプチャで、統計情報を生成してくれるデーモン

利点
- **パケットキャプチャ方式の為、ソフトウェアに依存しない。**
- **統計化も行う**
- **DSPという標準の可視化の方法がある。**
- **統計化した後のデータを残すので、データ容量が小さい**

欠点
- **統計化されてるので、詳細なところまでは追えない**
- **プリセット以外の加工をするにはソースの改造が必要**
  - 4LDとかできません
- **パケットキャプチャ方式ではDNS暗号化には対応できない。**
  - DNSTAP版も作成中のようです。

DNSTAP (http://dnstap.info)

ソフトウェアのクエリーログ機能が標準化されたもの
Protocol Buffer ベースで構造化されたクエリデータが手に入る
DNSサーバがReceiverがListenしているUnix socketに
Frame Streams Protocolで書込む
Receiverは自由に実装でき、受け取ったクエリデータを
ファイルに落としたり、別ホストに飛ばしたりできる。



DNSTAPで収集可能な項目（抜粋）

カラム名	内容
message.type	このDNSメッセージが何のメッセージが分かる CQ:クライアントクエリ(再起検索要求) CR:クライアントレスポンス(再起検索要求の応答) RQ:リゾルバクエリー（非再起検索要求） RR:リゾルバレスポンス（非再起検索要求の応答）
query_time, response_time	Time
socket_family	INET or INET6
socket_protocol	TCP or UDP
query_address	IPv4 or IPv6 address
query_message	DNSメッセージのバイナリ

このメッセージが何のメッセージか、分かるのはDNSTAPの大きな特徴

- スキーマ https://github.com/dnstap/dnstap.pb/blob/master/dnstap.proto
- 実際の例 https://gist.github.com/edmonds/5772879

Unboundは--enable-dnstap付きでコンパイルが必要です

設定方法

dnstapセクションを設定し、reloadします

```
dnstap:
        dnstap-enable: yes
        dnstap-socket-path: "/var/run/unbound/dnstap.sock"
        dnstap-send-identity: yes
        dnstap-send-version: yes
        dnstap-log-resolver-query-messages: yes
        dnstap-log-resolver-response-messages: yes
        dnstap-log-client-query-messages: yes
        dnstap-log-client-response-messages: yes
        dnstap-log-forwarder-query-messages: yes
        dnstap-log-forwarder-query-messages: yes
```

クエリーログ出力

Localのunboundにwww.nic.ad.jpのAを問い合わせた時の挙動

```
# sudo -u unbound dnstap -u /var/run/unbound/dnstap.sock
 ; dnstap.sockで待受開始、unboundのユーザで起動してsockを作る。
dnstap: opened input socket /var/run/unbound/dnstap.sock
 ; unboundが接続
dnstap.FrameStreamSockInput: accepted a socket connection
 ; クライアントがwww.nic.ad.jpのIN Aを問い合わせ
CQ 127.0.0.1 UDP 42b "www.nic.ad.jp." IN A
 ; リゾルバが203.119.1.1(a.dns.jp)にwww.nic.ad.jpのIN Aを問い合わせ
RQ 203.119.1.1 UDP 42b "www.nic.ad.jp." IN A
 ; 203.119.1.1(a.dns.jp)からレスポンス
RR 203.119.1.1 UDP 412b "www.nic.ad.jp." IN A
 ; リゾルバが203.119.1.1(ns5.nic.ad.jp.)にwww.nic.ad.jpのIN Aを問い合わせ
RQ 202.12.30.225 UDP 42b "www.nic.ad.jp." IN A
 ; 203.119.1.1(ns5.nic.ad.jp.) からレスポンス
RR 202.12.30.225 UDP 1964b "www.nic.ad.jp." IN A
 ;クライアントへレスポンス
CR 127.0.0.1 UDP 58b "www.nic.ad.jp." IN A
```

クエリーログ出力

-y付きで、yaml形式で表示させることもできます。


```
---
type: MESSAGE
message:
  type: CLIENT_QUERY
  query_time: !!timestamp 2018-11-14 08:58:03.962419
  socket_family: INET
  socket_protocol: TCP
  query_address: 127.0.0.1
  query_port: 33806
  query_message: |
    ;; opcode: QUERY, status: NOERROR, id: 62610
    ;; flags: rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

    ;; QUESTION SECTION:

    ;; ADDITIONAL SECTION:

    ;; OPT PSEUDOSECTION:
    ; EDNS: version 0; flags: ; udp: 4096
    ; PADDING:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000: ; udp: 4096
```

対応済み

- Unbound
- Bind9
- Knot
- Knot Resolver
- CoreDNS
- Dnsdist

対応中

- NSD
- PowerDNS

- **go-dnstap**
  - 公式のツール
  - file,tcp,unix socketから読み込み
  - Text形式、YAML、fstrm形式でのファイル書込に対応
- **fstrm**
  - https://github.com/farsightsec/fstrm
  - fstrmのC言語実装
  - tcp,unix socketから読み込み
  - fstrm形式でのファイル書込に対応

Protocol-bufferベースなので、色々な言語でライブラリは多いが、パケットの中身を見る程度で、DSCのような汎用的に使えるようなツールがないのが現状

利点

- **標準化されているので、実装が対応していれば、パーサーを使いまわせる。**
- **そのパケットがstub resolverからのクエリーなのか、resolverから出たクエリーなのかなどが判別可能**
  - Typeフィールド
- **PCAPに比べるとデータ容量が小さい**
  - ちょっとだけ

欠点

- **ソフトウェアが対応していなければ使えない**
  - NSDとか
- **ソフトウェアに手を入れる実装なので、バグを踏む確率は増える。**

- **クエリーログ**
  - DNSサーソフトウェア側の対応が必要
- **パケットキャプチャ**
  - ポートミラーリングと組み合わせると、DNSサーバに影響なくデータが取れる
  - TLS系には対応できない
- **DNSTAP**
  - 構造化されたDNSメッセージが取れる
  - DNSサーソフトウェア側の対応が必要
  - TLSも対応
  - フルクエリ取るのを今からやるならこれ
- **DSC**
  - 加工、集計を行う
  - DSPとの組み合わせで可視化もサポート
  - TLS系には対応できない
  - 迷ったらとりあえずこれを入れておけ

DNSデータの加工、保存

加工すべきもの１「QNAME」

そのままだと有意なデータにならないことがある
例えばランダムサブドメイン攻撃を捉えることは難しい
2LDや3LDあたりで区切った名前も統計対象とすると良い

```
1.www.example.jp 1 access
2.www.example.jp 1 access
（略）
100.www.example.jp 1 access
```

区切らないとと100個の名前が1access
3LDで区切るとwww.example.jpが100アクセス

DSCはもちろん対応

加工すべきもの２「クライアントIP」

IPv4は/24でマスク、IPv6は/48程度でマスクするのが良い

特にIPv6広大な空間がある為、マスクしないとDBが悲鳴をあげます。

DSCはもちろん対応

- **一番簡単な方法はDSCを使うこと**
  - 一定間隔毎の集計したXMLまたはJSONデータに加工

- **各種クエリーログ、PCAPファイル、DNSTAPのデータ**
  - 加工、集計をとる汎用的な物は今のところない
  - 独自にプログラムを書く必要がある

- **特定の日時の見たい場合だけ**
  - ファイルで保存

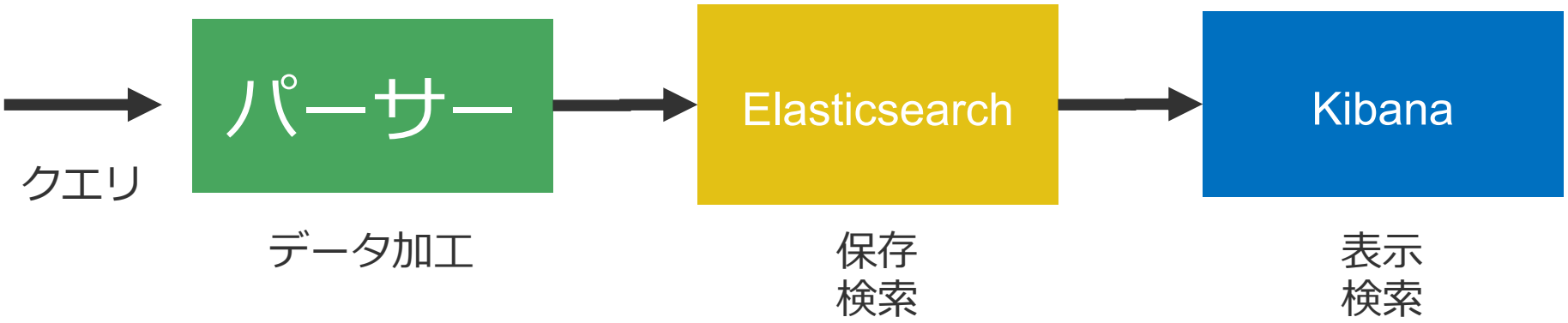
- **検索、分析、機械学習、集計、可視化などを行いたい**
  - Elasticsearchなどの全文検索エンジンに保存
  - 必要な資源（ディスク、CPU、帯域）はファイルとは桁違い
    - DNSサーバよりESの方がたくさん必要になったり。。。
  - ログの保存期間は資源次第ですが、

- **Elasticsearchの場合**
  - Kibanaで可視化
- **DSCのファイル**
  - DSPで可視化
  - DSCファイルをパースし、Elasticsearchに入れてkibanaで可視化
- **それ以外**
  - 独自で頑張る

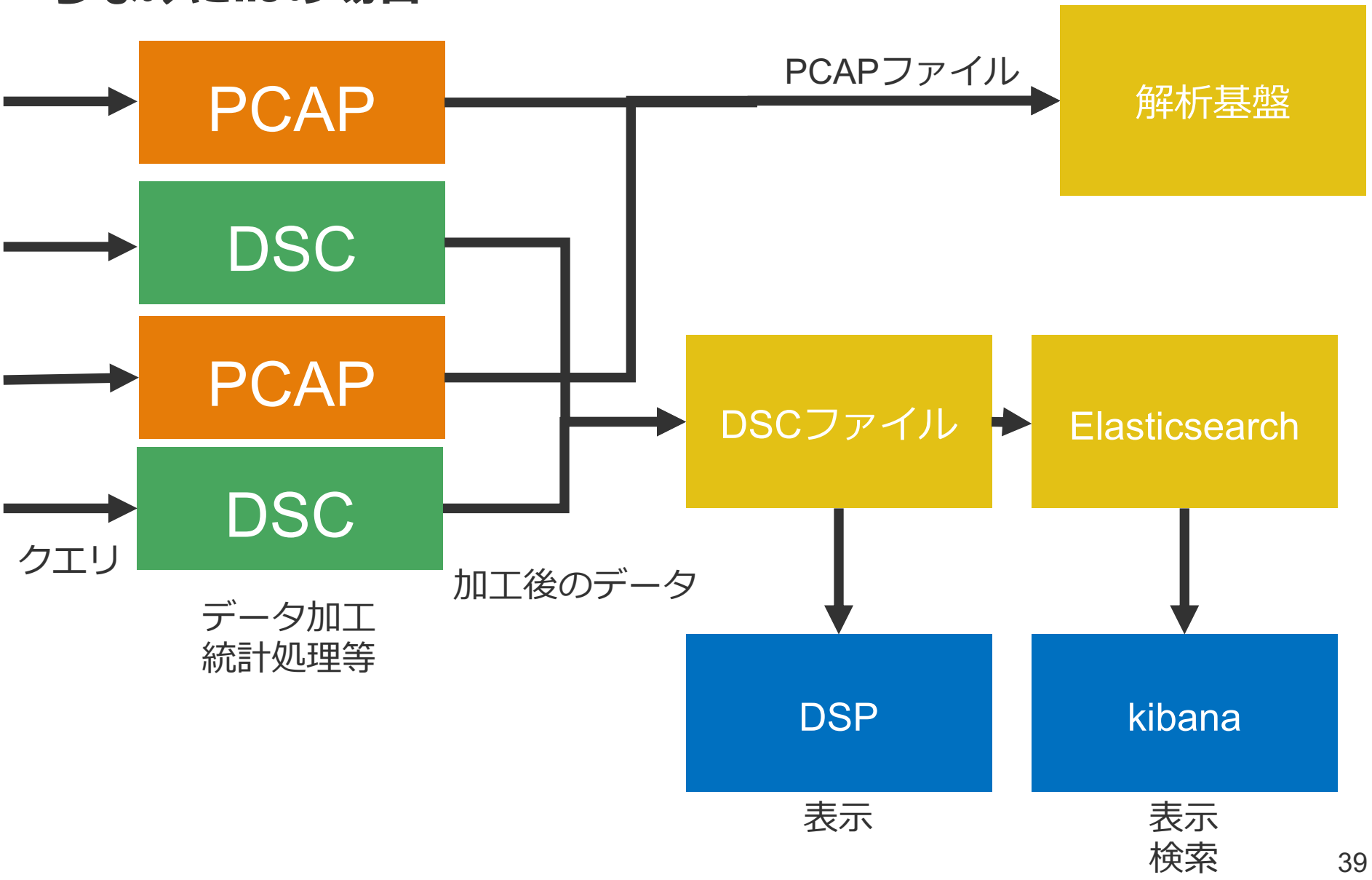
結局何を使えばいいのか

- **集計する場合**

- **集計しない場合**

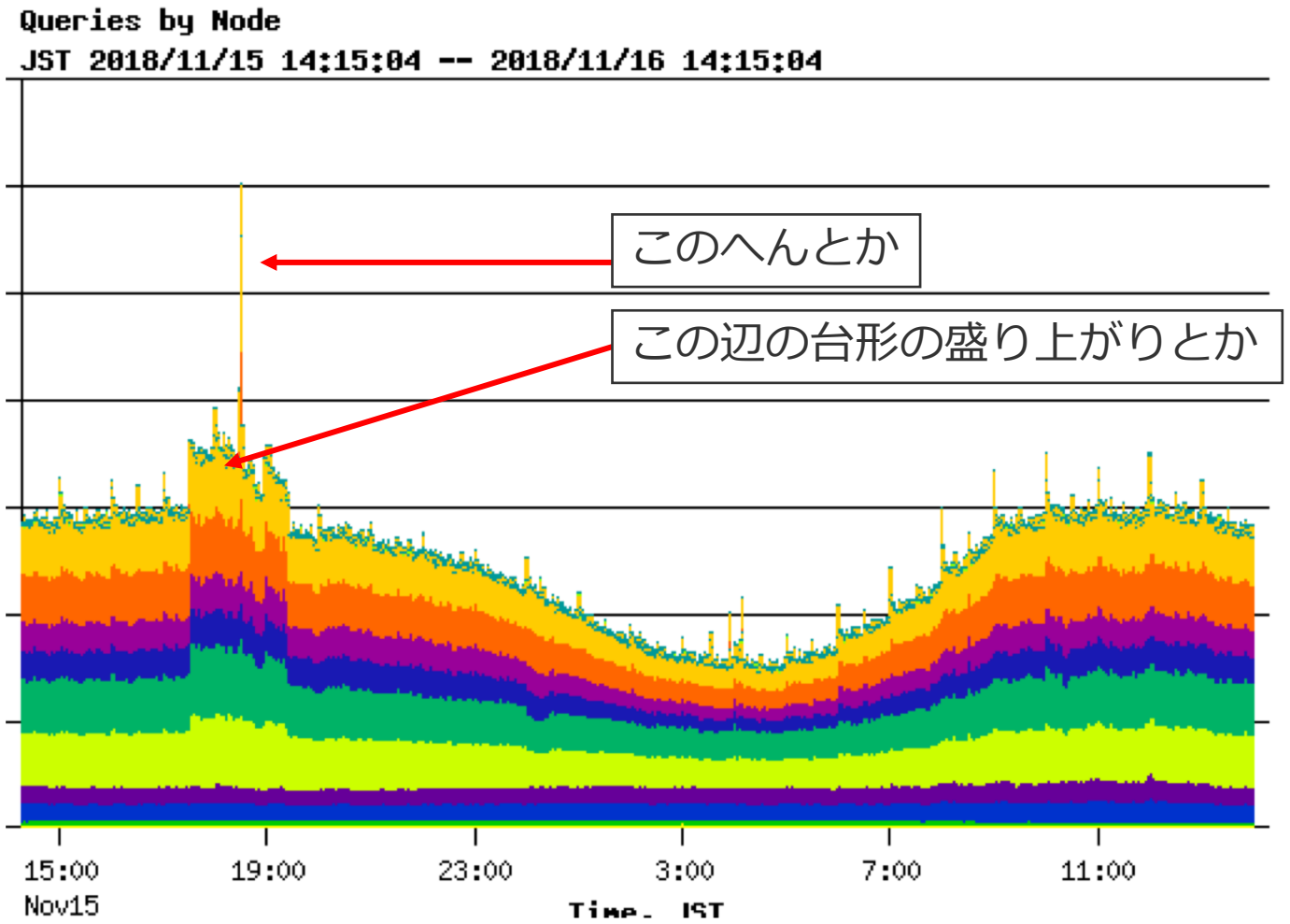


ちなみにIIJの場合


PCAP
DSC
PCAP
DSC
クエリ
データ加工
統計処理等
PCAPファイル
解析基盤
加工後のデータ
DSCファイル
Elasticsearch
DSP
表示
kibana
表示
検索

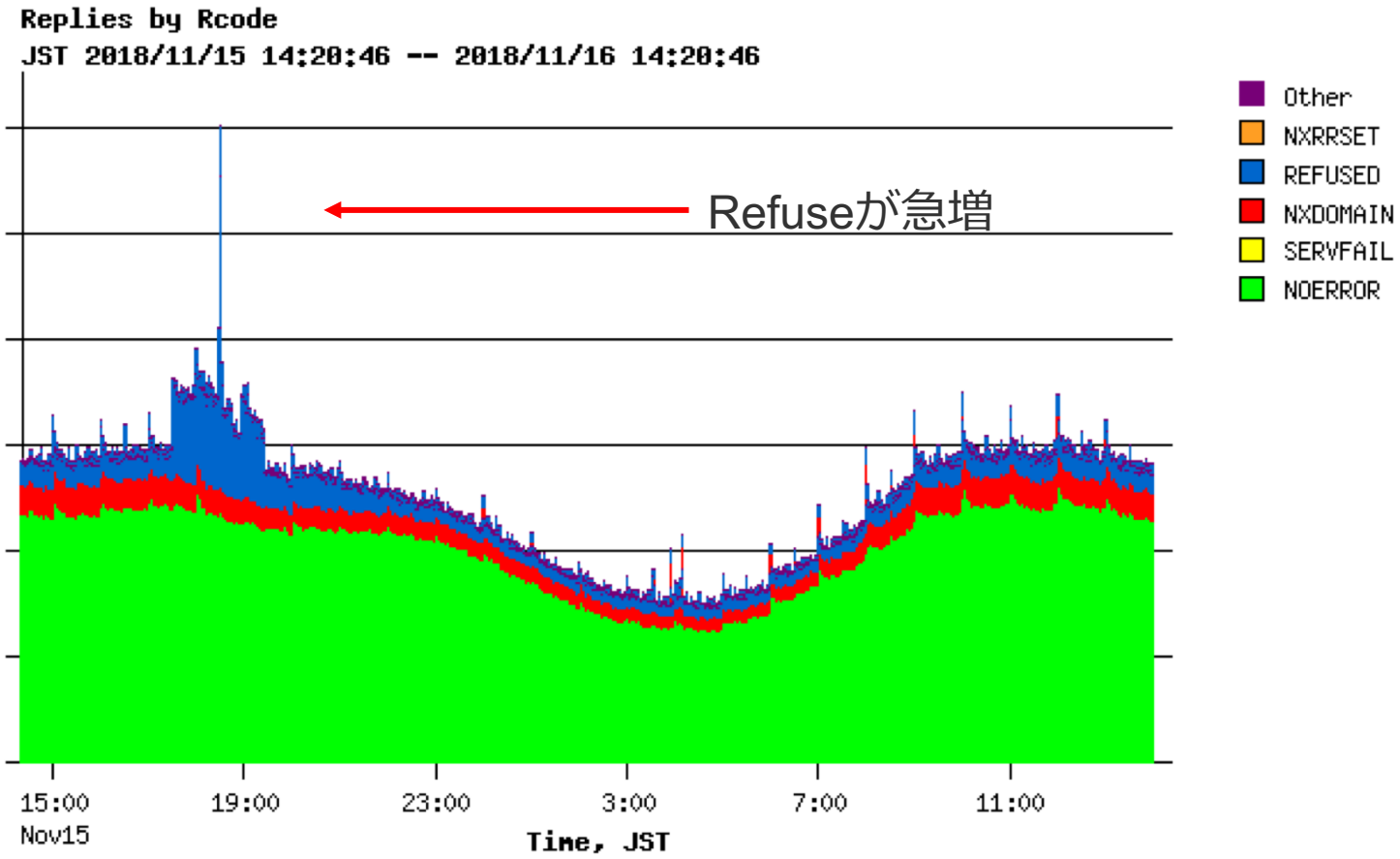
最後に
DNSデータの可視化の例

DSCを使ったとある権威DNSサーバ群のホスト別のQPS画面



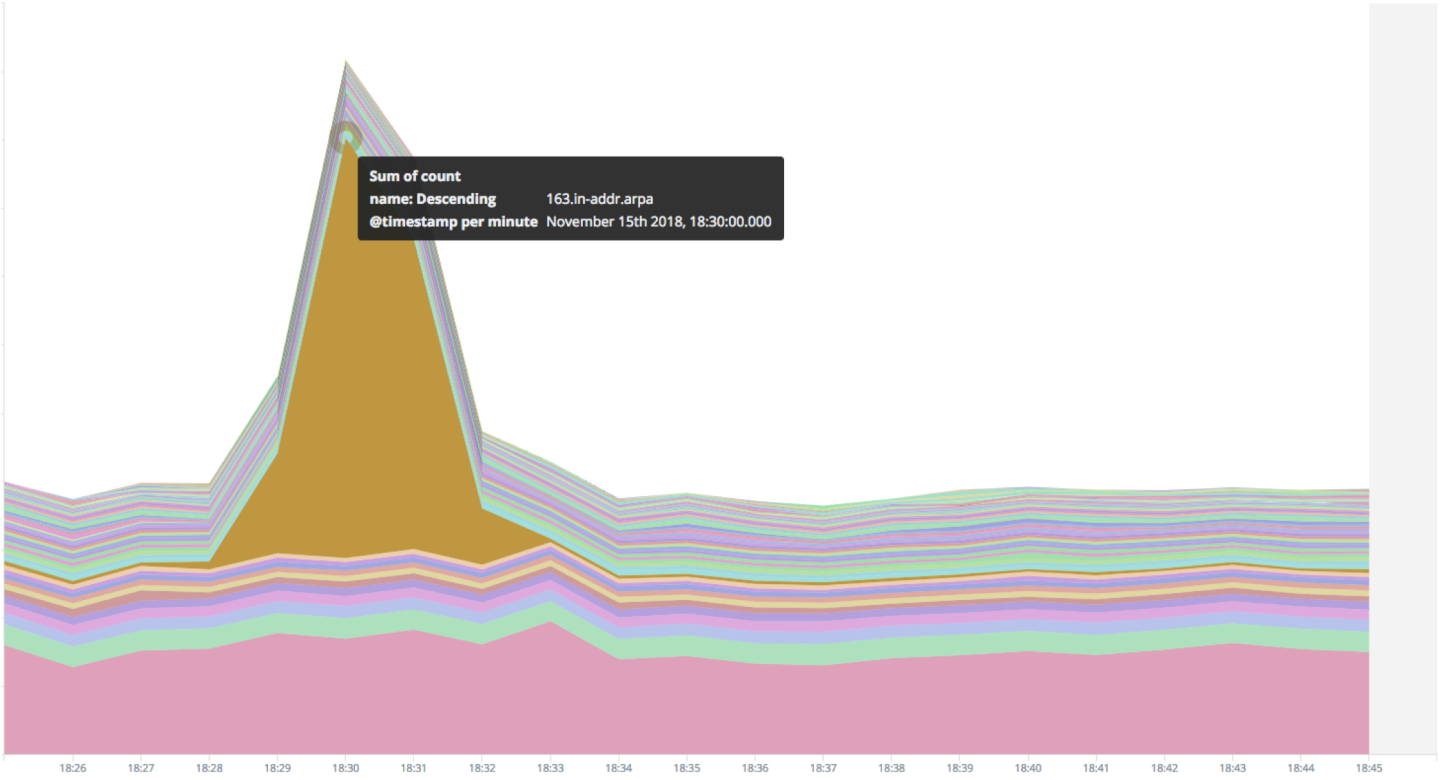
同じ時刻のRCODE別のクエリー数

NOERRORには変化ないので、正常な問い合わせは問題無いことがわかる



Rcodeの集計グラフはデータの可視化の中でも一番役に立つもので、
例えば、ソフトウェアアップデート後にグラフの傾向が変わっていないことで
正常性を判断することがでます。

DSCで集計したデータをElasticsearchに投入し、Kibanaで表示した例



163.in-addr.arpa以下のゾーンに対する大量のアクセスと判明。
ISPあるあるです。

- **集計する場合は、DSC+DSPが鉄板**
  - DSC+Elasticsearch+kibanaという手も
- **集計しない場合は、Elasticsearch+kibana**

日本のインターネットは1992年、IIJとともにはじまりました。以来、IIJグループはネットワーク社会の基盤をつくり、技術力でその発展を支えてきました。インターネットの未来を想い、新たなイノベーションに挑戦し続けていく。それは、つねに先駆者としてインターネットの可能性を切り拓いてきたIIJの、これからも変わることのない姿勢です。IIJの真ん中のIはイニシアティブ――IIJはいつもはじまりであり、未来です。

Appendix

発表後アップデート予定