DataCenter とソフトウェア開発ワークショップ

SF-TAP : Scalable and Flexible Traffic Analysis Platform

L7レベルトラフィック解析プラットフォームの開発

情報通信研究機構
サイバー攻撃対策総合研究センター
サイバー攻撃検証研究室
高野 祐輝

共同研究者：
三浦，安田，明石，井上

https://github.com/SF-TAP/

おしながき

- 背景と目的，本研究の位置づけ
- SF-TAPの設計
- SF-TAPの実装
- パフォーマンス計測
- デモ

背景と目的

- もっとプログラマブルなL7解析器がほしい
  - Python，Ruby，Cで解析ロジックを書きたい
  - DSL覚えたくない
  - Cで文字列処理したくない
    - 自前でTCPストリームの再構成処理とか書いてらんない
- もっとスケーラビリティの高い解析器がほしい
  - 高bps，高pps
  - 水平スケール，コアスケール
- コモディティベースで実現したい
  - 高価で取り回しの難しいアプライアンスを使いたくない
  - ベンダーロックインからの開放
- NFVのVNFとしてサービスチェインを行い，ネットワークトラフィック解析をソフトウェアで自由に行いたい

関連研究と本研究の位置づけ

フローレベル解析技術	L7プロトコル判別
GASPP[USENIX ATC 2014]	nDPI　　l7-filter
SCAP[IMC 2012] libnids	libprotoident

本研究：SF-TAP
＋モジュラリティ＆スケーラビリティ

トラフィックキャプチャ技術

BPF
pcap

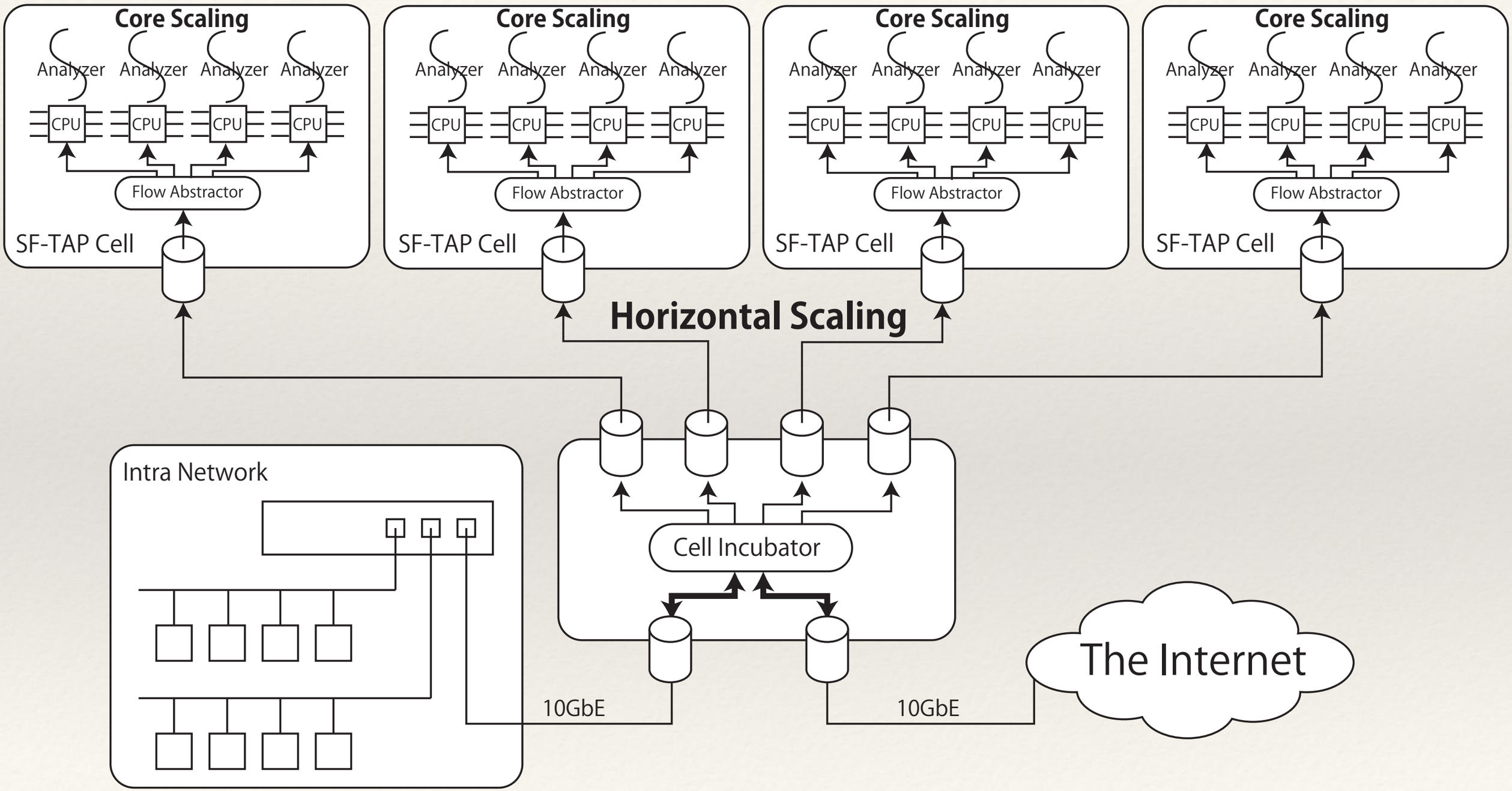
netmap
DPDK

https://github.com/SF-TAP/

SF-TAPの提案

- 柔軟で，スケーラビリティのあるL7レベルトラフィック解析基盤を提案

SF-TAPの動作概念図



SF-TAPの設計原理（1）

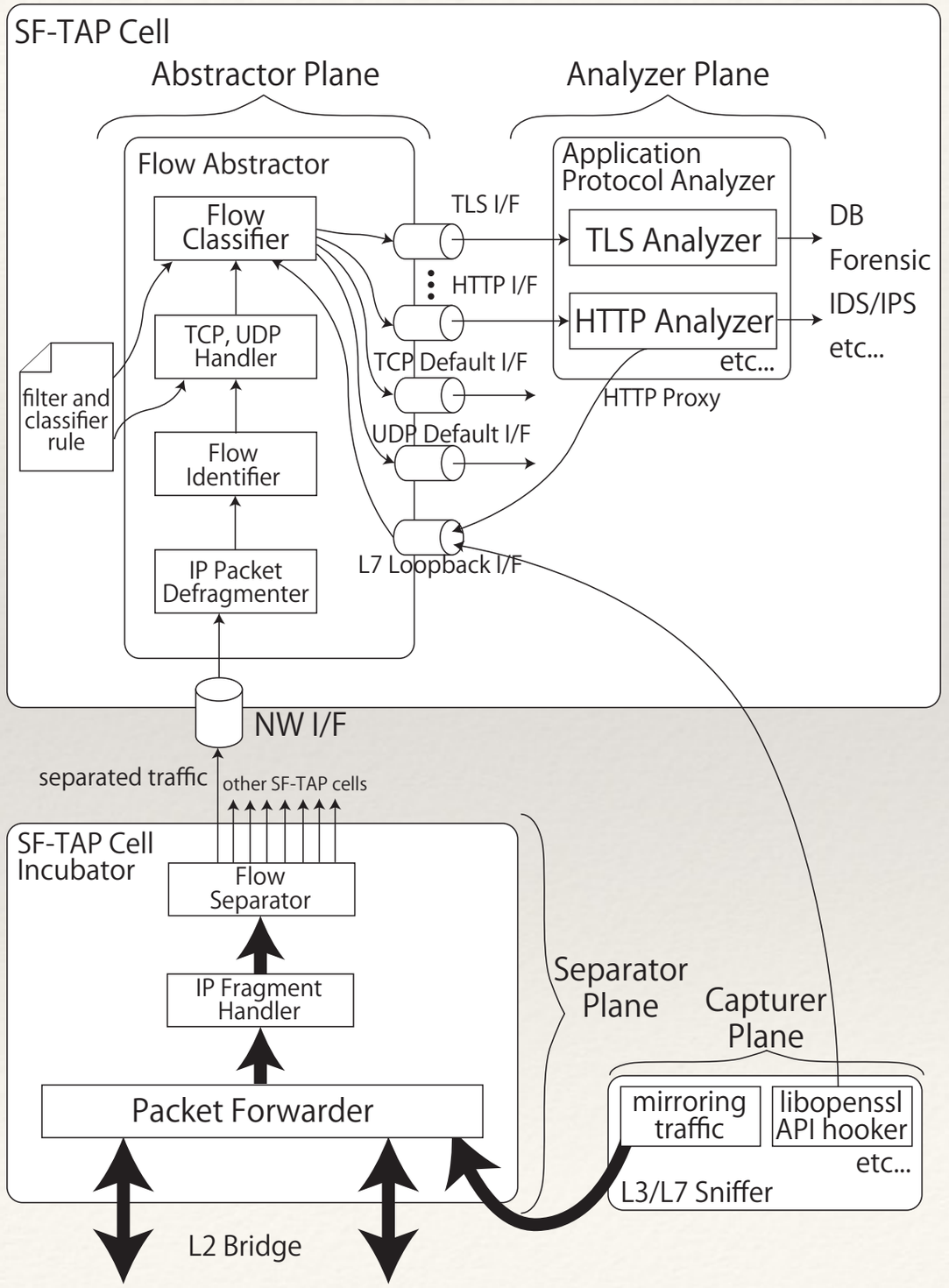
- フロー抽象化
  - トラフィックをL7プロトコルで抽象化
  - Unixの/devやBPFのような抽象化IF
    - 開発者の得意な言語で解析ロジックの記述が可能
    - ネットワークフォレンジック，IDS，IPS，機械学習など，用途に応じた言語の選択が可能
- モジュラアーキテクチャ
  - 解析ロジックとキャプチャ部分の分離
  - 解析ロジックの容易な付替えが可能に

SF-TAPの設計原理（２）

- 水平スケール
  - 解析ロジック（機械学習など）の処理には，多量の計算リソースが必要
    - 台数効果で計算リソース不足を解消
- コアスケール
  - 計算リソースを有効に活用
  - キャプチャ部分及び，解析ロジック部分のコアスケールを可能に

SF-TAPの設計

- SF-TAP Cell
  - Flow Abstractor
    - IPパケット再構成
    - フロー識別
    - TCPストリーム再構成
    - L7プロトコル判別
  - Application Protocol Analyzer
    - ユーザが記述
    - HTTPパーサなど
- SF-TAP Cell Incubator
  - フロー単位のトラフィック分割
  - TAP＆インラインモード
  - IPフラグメント対応



https://github.com/SF-TAP/

SF-TAPの実装

- Flow Abstractor
  - C++で実装
  - L7 IF部分はUnix Domain Socketを利用
  - pcapを用いてトラフィックキャプチャ
- SF-TAP Cell Incubator
  - C++で実装
  - netmapを利用
- HTTP Parser
  - Pythonで実装
  - 解析ロジックの一例

Flow Abstractorの設定ファイル

```
1   http:
2       up       = ^[-a-zA-Z]+ .+ HTTP/1\.(0\r?\n|1\r?\n([-a-
    zA-Z]+: .+\r?\n)+)
3       down     = ^HTTP/1\.[01] [1-9][0-9]{2} .+\r?\n
4       proto    = TCP  # TCP or UDP
5       if       = http # path to UNIX domain socket
6       nice     = 100  # priority
7       balance = 4     # balaced by 4 IFs
8
9   torrent_tracker: # BitTorrent Tracker
10      up       = ^GET .*(announce|scrape).*\?.*info_hash
    =.+&.+ HTTP/1\.(0\r?\n|1\r?\n([-a-zA-Z]+: .+\r?\n)+)
11      down     = ^HTTP/1\.[01] [1-9][0-9]{2} .+\r?\n
12      proto    = TCP
13      if       = torrent_tracker
14      nice     = 90  # priority
15
16  dns_udp:
17      proto    = UDP
18      if       = dns
19      port     = 53
20      nice     = 200
```

Flow Abstractorの設定ファイル

この正規表現にマッチしたトラフィックが出力される

出力先IF

L4プロトコル選択

パターンの優先順位

ロードバランス用

ポート番号指定

```
1   http:
2       up       = ^[-a-zA-Z]+ .+ HTTP/1\.(0\r?\n|1\r?\n([-a-
    zA-Z]+: .+\r?\n)+)
3       down     = ^HTTP/1\.[01] [1-9][0-9]{2} .+\r?\n
4       proto    = TCP  # TCP or UDP
5       if       = http # path to UNIX domain socket
6       nice     = 100  # priority
7       balance = 4     # balaced by 4 IFs
8
9   torrent_tracker: # BitTorrent Tracker
10      up       = ^GET .*(announce|scrape).*\?.*info_hash
    =.+&.+ HTTP/1\.(0\r?\n|1\r?\n([-a-zA-Z]+: .+\r?\n)+)
11      down     = ^HTTP/1\.[01] [1-9][0-9]{2} .+\r?\n
12      proto    = TCP
13      if       = torrent_tracker
14      nice     = 90  # priority
15
16  dns_udp:
17      proto    = UDP
18      if       = dns
19      port     = 53
20      nice     = 200
```

Flow Abstractorの
抽象L7 IFディレクトリ構造

```
1   $ ls -R /tmp/sf-tap
2   loopback7=          tcp/                    udp/
3
4   /tmp/sf-tap/tcp:
5   default=            http2=                  ssh=
6   dns=                http3=                  ssl=
7   ftp=                http_proxy=             torrent_tracker=
8   http0=              irc=                    websocket=
9   http1=              smtp=
10
11  /tmp/sf-tap/udp:
12  default=            dns=                    torrent_dht=
```

Flow Abstractorの出力例

```
1  ip1=192.168.0.1,ip2=192.168.0.2,port1=62918,port2=80,hop=0,l3=ipv4,l4=tcp,event=CREATED
2  ip1=192.168.0.1,ip2=192.168.0.2,port1=62918,port2=80,hop=0,l3=ipv4,l4=tcp,event=DATA,from=2,match=down,len=1398
3
4  1398[bytes] Binary Data
5
6  ip1=192.168.0.1,ip2=192.168.0.2,port1=62918,port2=80,hop=0,l3=ipv4,l4=tcp,event=DESTROYED
```

- IPアドレス，ポート番号，L3/L4プロトコル，hopでフロー判別
  - hopはFlow Abstractorへ，トラフィックを再注入した場合に利用される
- TCPの複雑なイベントを，CREATED，DATA，DESTROYEDに抽象化

HTTP Parser

- 解析ロジックの一例
- Pythonで実装（469行）
- Flow Abstractorの出力をストリーム処理でパースし，JSON形式で出力

```
 {
  "client": {
    "port": "61906",
    "ip":"192.168.11.12",
    "header": {
      "host": "www.nsa.gov",
      "user-agent":"Mozilla\/5.0 (Macintosh; Intel Mac OS X 10.9; rv:31.0) Gecko\/20100101 Firefox\/31.0",
      "connection": "keep-alive",
      "pragma": "no-cache",
      "accept": "text\/html,application\/xhtml+xml,application\/xml;q=0.9,*\/*;q=0.8",
      "accept-language": "ja,en-us;q=0.7,en;q=0.3", 11 "accept-encoding": "gzip, deflate",
      "cache-control": "no-cache"
    },
    "method": {
      "method": "GET",
      "uri": "\/",
      "ver": "HTTP\/1.1"
    },
    "trailer": {}
  },
  "server": {
    "port": "80",
    "ip": "23.6.116.226",
      "header": {
      "connection": "keep-alive",
      "content-length":"6268",
      "date": "Sat, 16 Aug 2014 11:38:25 GMT",
      "content-encoding": "gzip",
      "vary": "Accept-Encoding",
      "x-powered-by": "ASP.NET",
      "server": "Microsoft-IIS\/7.5",
      "content-type": "text\/html"
    },
    "response": {
      "ver": "HTTP\/1.1",
      "code": "200",
      "msg": "OK"
    },
    "trailer": {}
  }
}
```

デモ