# dSharK: A General, Easy to Program and Scalable Framework for Analyzing In-network Packet Traces

Da Yu[†] , Yibo Zhu[§] , Behnaz Arzani[§] , Rodrigo Fonseca[†] ,
Tianrong Zhang[§] , Karl Deng[§] , Lihua Yuan[§]

[†]Brown University [§]Microsoft

# Network reliability is critical

Cloudflare: A bad config (router rule) caused all of their edge routers to crash, taking down all of Cloudflare.

LILY HAY NEWMAN SECURITY 11.06.17 05:00 PM

HOW A TINY ERROR SHUT OFF THE INTERNET FOR PARTS OF THE US

Steam Outage: How to Monitor Da
Conn

Posted by N

Stack Overflow: A bad firewall config blocked stackexchange/stackoverflow.

Etsy: Sending multicast traffic without properly configuring switches caused an Etsy global outage.

How a typo took do
internet

*Hello, operator*

By Casey Newton | @CaseyNewton | Mar 2, 2017, 1:24pm EST

SHARE

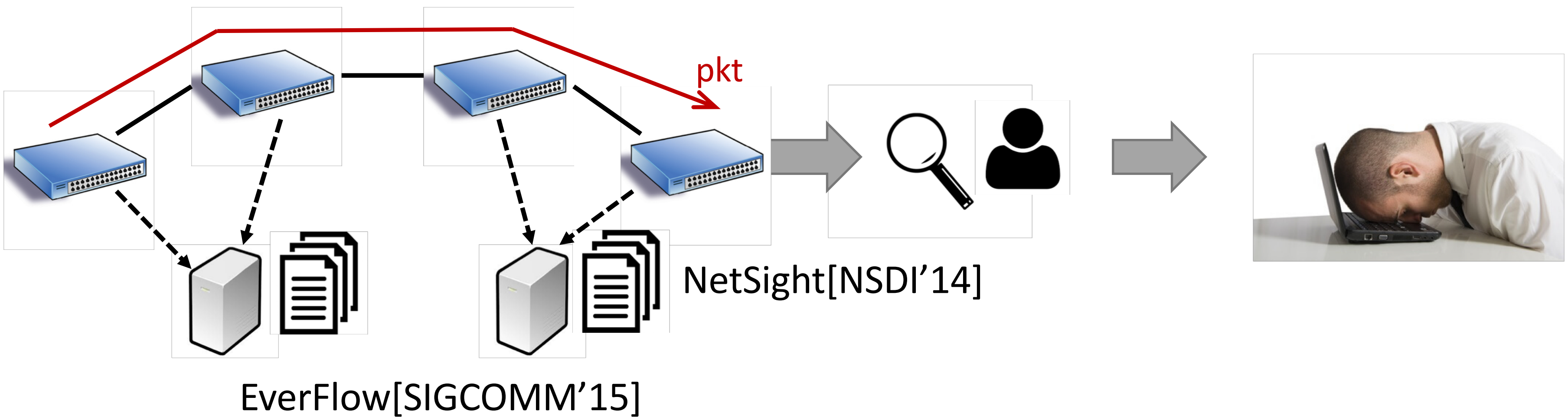
suffers service outage due to odd traffic routing

Looks like someone pressed the wrong button on the routing machine

# Existing tools are the first attempts

| | End-host based | Topology or hardware specific | Target on specific problems |
|---|---|---|---|
| Trumpet[SIGCOMM'16] | ✗ | | ✗ |
| Sonata[SIGCOMM'18] | | ✗ | |
| PathDump[OSDI'16] | ✗ | ✗ | |
| 007[NSDI'18] | ✗ | ✗ | ✗ |
| SwitchPointer[NSDI'18] | ✗ | ✗ | |
| Pingmesh[SIGCOMM'15] | ✗ | ✗ | ✗ |
| INT | | ✗ | |

# In-network packet capture is the last resort




Analyzing the in-network packet traces is challenging!

# In-network analysis: challenges

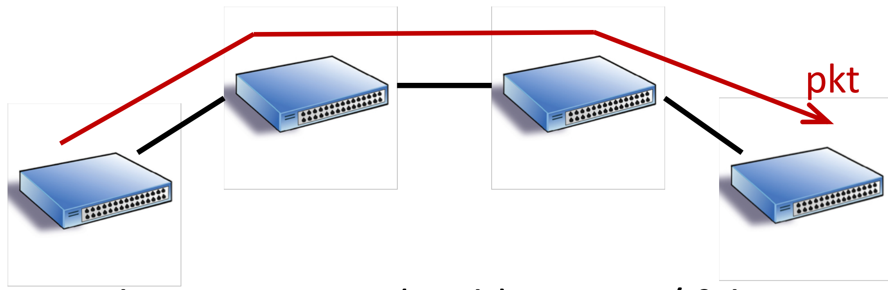
## Volume

- 3.33 Mpps line-speed (10 Gbps, 1500 Bytes)

## Analysis logic varies

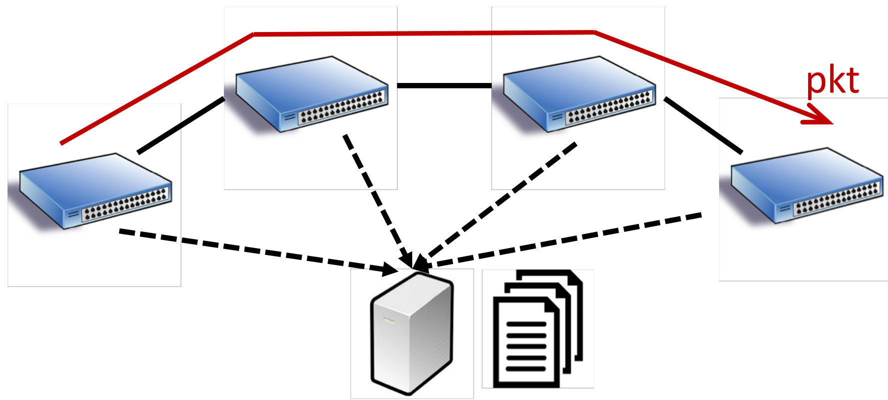
- Logic is different case by case

# Example: Route error checker



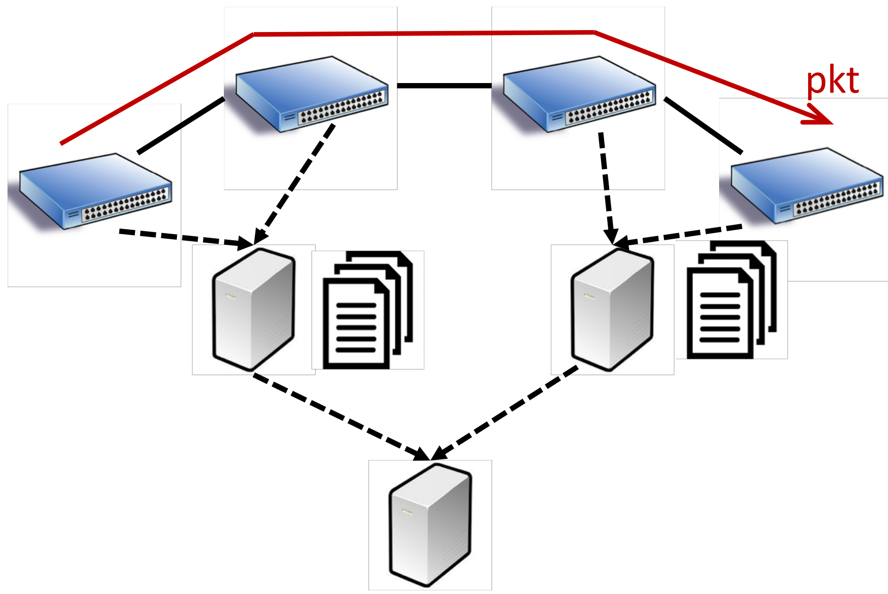

hasRouteError(path) -> true / false

# Example: Route error checker



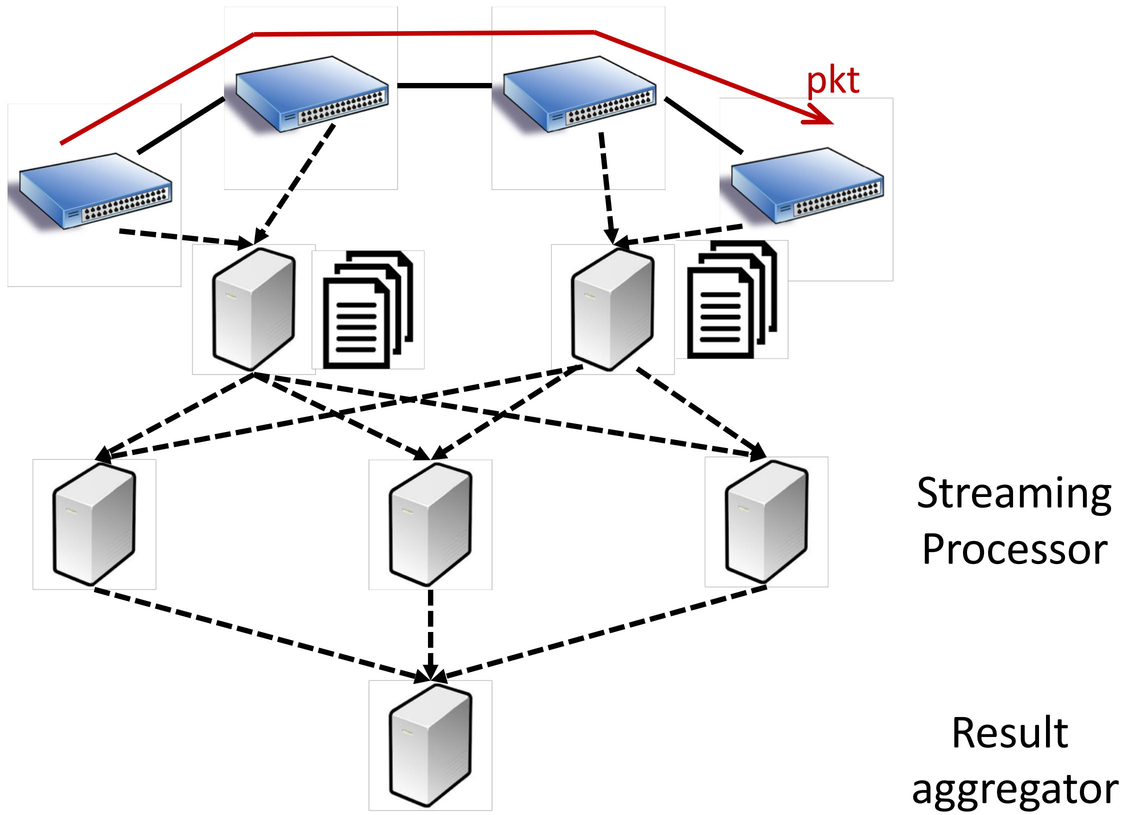

hasRouteError(path) -> true / false

# Example: Route error checker




hasRouteError(path) -> true / false

# Example: Route error checker




hasRouteError(path) -> true / false

# In-network analysis: challenges

## Volume

- 3.33 Mpps line-speed (10 Gbps, 1500 Bytes)

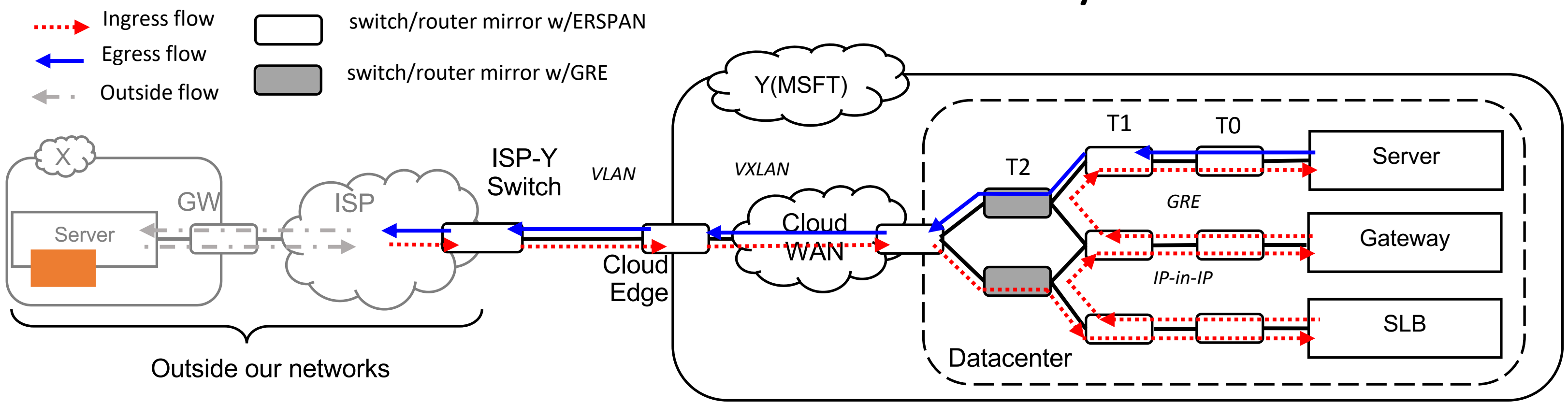
## Analysis logic varies

- Logic is different case by case

## Difficult to get robust analysis

- Header transformation
  - Headers are modified by the middleboxes

# Packet headers are modified by middleboxes




| Header format | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Headers added after mirroring | | | Mirrored headers | | | | | | | |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | | | | | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | | | | 802.1Q | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | GRE | | | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | IPV4 | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | GRE | | IPV4 | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | IPV4 | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | GRE | | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | GRE | | | IPV4 | GRE | | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | GRE | | ETHERNET | IPV4 | TCP |

# Same protocol headers bring ambiguity

| Header format | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Headers added after mirroring | | | Mirrored headers | | | | | | | |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | | | | | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | | | | 802.1Q | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | GRE | | | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | IPV4 | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | GRE | | IPV4 | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | IPV4 | IPV4 | UDP | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | GRE | | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | GRE | | | IPV4 | GRE | | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | GRE | | ETHERNET | IPV4 | TCP |

# dShark: three goals

Scalable

Broadly applicable

Robust in the wild

- Header transformation

# dShark: three goals

## Scalable

- Components work independently and in parallel.

## Broadly applicable

## Robust in the wild

- Header transformation

# dShark: three goals

## Scalable

- Components work independently and in parallel.

## Broadly applicable

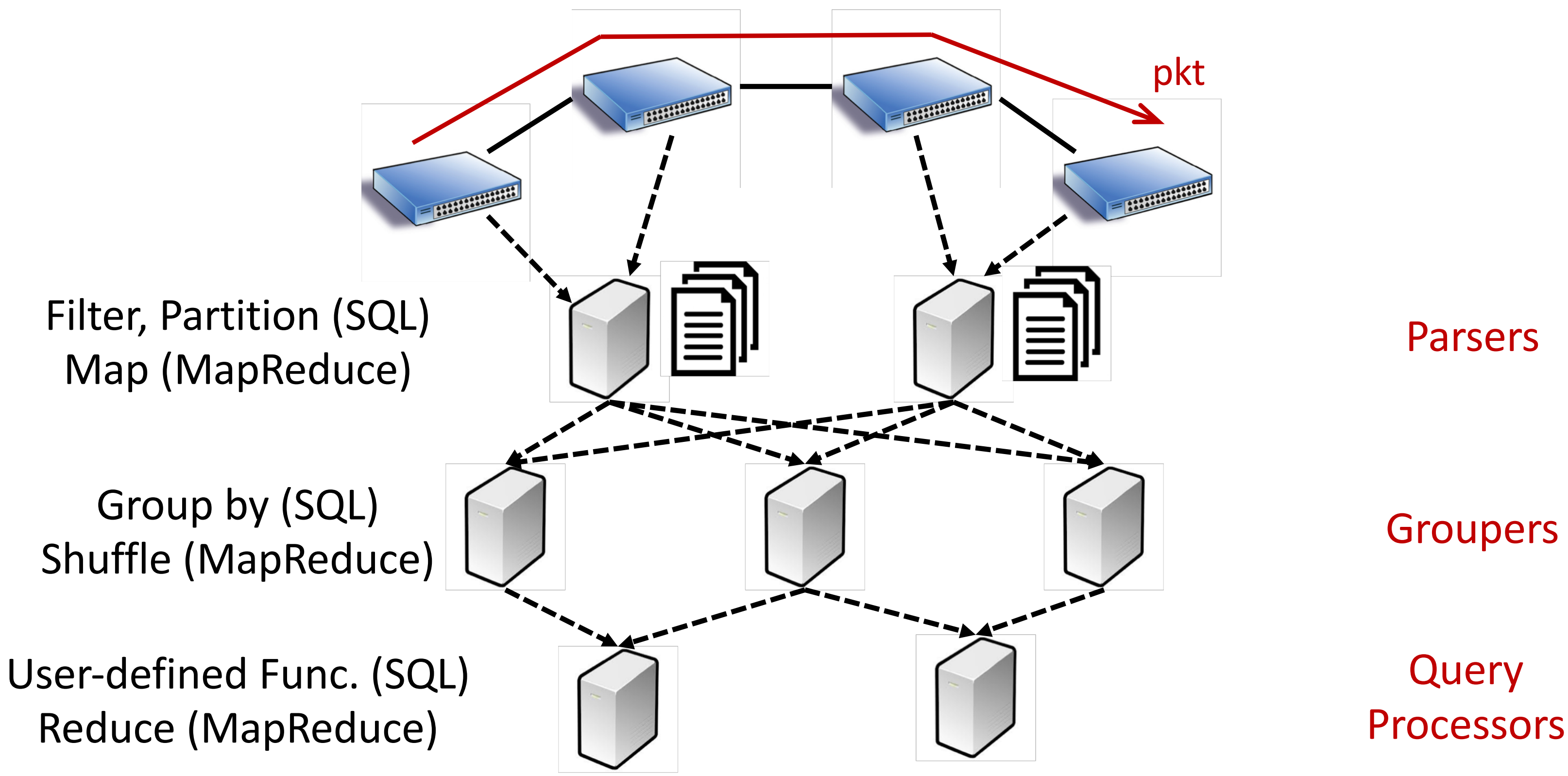
## Robust in the wild

- Header transformation

# How operators manually process traces

Observation #1:

- Four diagnosis steps: parse, filter, aggregate and analyze




Need to tightly integrated with the collecting infrastructure!

# How operators manually process traces

Observation #2:

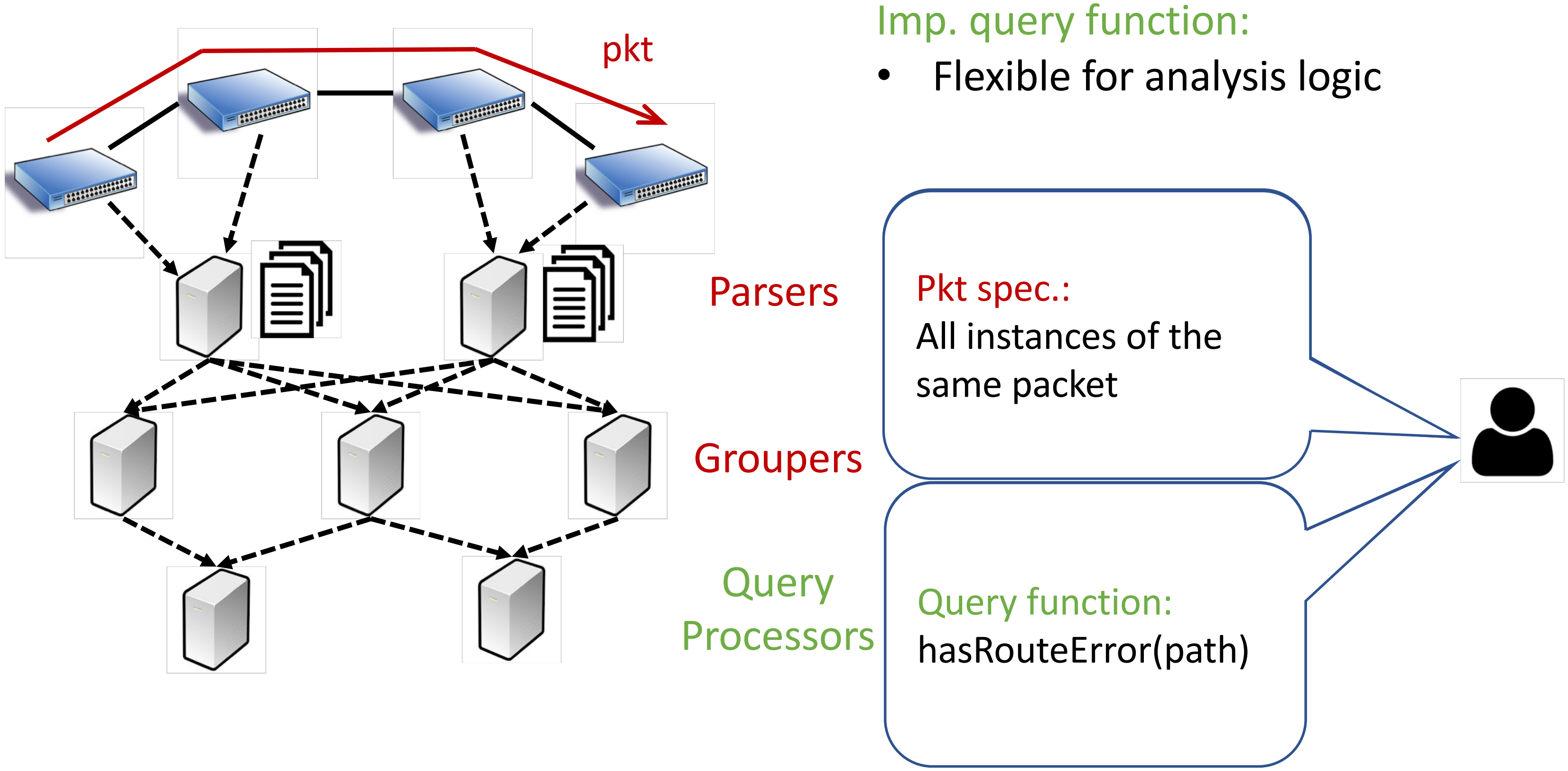
- Diagnosis logic always run on top of 4 aggregation types

| | One-Hop | Multi-Hop |
|---|---|---|
| One-Packet | check appearance of a packet | show full path of each packet in the network |
| Multi-Packet | diagnose middlebox behaviors | complicated cases that requires end information |

# dShark's programming model

Decl. packet spec.:
- Familiar to the operators

Imp. query function:
- Flexible for analysis logic



# Declarative spec. in parsers and groupers

A packet summary is a byte array that only contains fields that the operators are interested in.

```
{
  Summary: {
    Key: [ipId, seqNum],
    Additional: [ttl]
  },

  Name: {
    ipId: ipv4.id,
    seqNum: tcp.seq,
    ttl: ipv4.ttl
  }
}
```

Definition of a packet summary

How to extract the values in the header

# Same protocol headers bring ambiguity

| Header format | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Headers added after mirroring | | | Mirrored headers | | | | | | | | |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | | | | | | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | | | | | 802.1Q | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | UDP | | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | GRE | | | IPV4 | UDP | | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | UDP | | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | IPV4 | IPV4 | UDP | | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | GRE | | IPV4 | IPV4 | UDP | | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | IPV4 | IPV4 | UDP | | VXLAN | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | | GRE | | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | GRE | | | IPV4 | | GRE | | ETHERNET | IPV4 | TCP |
| ETHERNET | IPV4 | ERSPAN | ETHERNET | | IPV4 | | GRE | | ETHERNET | IPV4 | TCP |

ipv4[0]

ipv4[3]
ipv4[-1]

# Declarative spec. in parsers and groupers

```
{
  Summary: {
    Key: [ipId, seqNum],
    Additional: [ttl]
  },

  Name: {
    ipId: ipv4[-1].id,
    seqNum: tcp[-1].seq,
    ttl: ipv4[:].ttl
  }
}
```

Definition of a packet summary

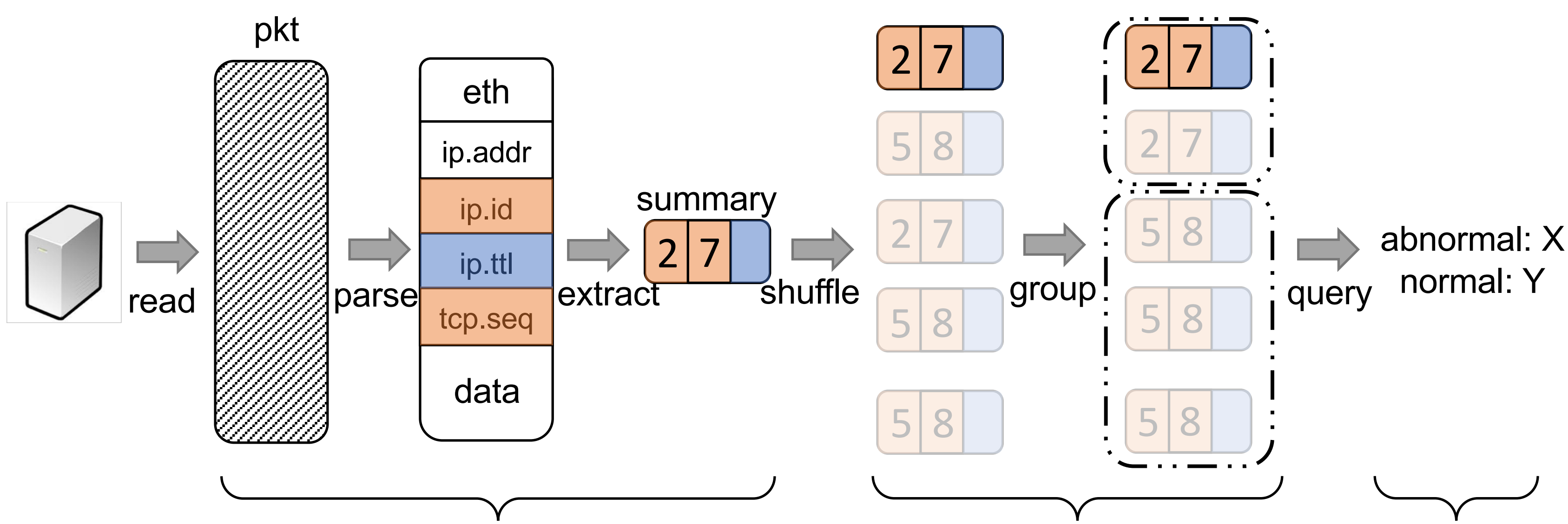
How to extract the values in the header

# Imperative diagnosis logic in the query processors

```
Pair<Object, Object> query (const vector<Summary>& group) {

  // ReconstructS the path based on TTL
  constructPath(group);

  // Checks path
  bool result = hasRouteError(group);

  return make_pair(result, 1);
}
```

- In practice, this is implemented in 49 lines of code for the query function!

# dShark overview




**Parser**

- Reads packets
- Parses, extracts fields for summaries
- Sends summaries based on the keys

**Grouper**

- Receives summaries
- Aggregates summaries based on the keys

**Query Processor**

- Analyze summary groups
- Returns the output

# Another example: load balancer profiler

Spec:

```
{
  Summary: {
    Key: [ipId, seqNum],
    Additional: [vip, pip]
  },

  Name: {
    ipId: ipv4[-1].id,
    seqNum: tcp[-1].seq,
    vip: ipv4[-1].dst,
    pip: ipv4[0].dst
  }
}
```

- Innermost ipId and tcp seq # to identify a packet
- Virtual IP and the physical IP

Func.:

```
Pair<Pair<IP, IP>, int> query(group) {
  // Validate data
  ...

  return Pair((Pair(vip, pip), 1);
}
```

- Returns the map

- In practice, this is implemented in 18 lines of code for the query function!

| Group pattern | Application | Analysis logic | In-nw ck. only | Header transf. | Query LOC |
|---|---|---|---|---|---|
| One packet on one hop | Loop-free detection [21]<br>*Detect forwarding loop* | *Group:* same packet(ipv4[0].ipid, tcp[0].seq) on one hop<br>*Query:* does the same packet appear multiple times on the same hop | No | No | 8 |
| | Overloop-free detection [69]<br>*Detect forwarding loop involving tunnels* | *Group:* same packet(ipv4[0].ipid, tcp[0].seq) on tunnel endpoints<br>*Query:* does the same packet appear multiple times on the same endpoint | Yes | Yes | 8 |
| One packet on multiple hops | Route detour checker<br>*Check packet's route in failure case* | *Group:* same packet(ipv4[-1].ipid, tcp[-1].seq) on all switches<br>*Query:* is valid detour in the recovered path(ipv4[:].ttl) | No | Yes* | 49 |
| | Route error<br>*Detect wrong packet forwarding* | *Group:* same packet(ipv4[-1].ipid, tcp[-1].seq) on all switches<br>*Query:* get last correct hop in the recovered path(ipv4[:].ttl) | No* | Yes* | 49 |
| | Netsight [21]<br>*Log packet's in-network lifecycle* | *Group:* same packet(ipv4[-1].ipid, tcp[-1].seq) on all switches<br>*Query:* recover path(ipv4[:].ttl) | No* | Yes* | 47 |
| | Hop counter [21]<br>*Count packet's hop* | *Group:* same packet(ipv4[-1].ipid, tcp[-1].seq) on all switches<br>*Query:* count record | No* | Yes* | 6 |
| Multiple packets on one hop | Traffic isolation checker [21]<br>*Check whether hosts are allowed to talk* | *Group:* all packets at dst ToR(SWITCH=dst_ToR)<br>*Query:* have prohibited host(ipv4[0].src) | No | No | 11 |
| | Middlebox(SLB, GW, etc) profiler<br>*Check correctness/performance of middleboxes* | *Group:* same packet(ipv4[-1].ipid, tcp[-1].seq) pre/post middlebox<br>*Query:* is middlebox correct(related fields) | Yes | Yes | 18† |
| | Packet drops on middleboxes<br>*Check packet drops in middleboxes* | *Group:* same packet(ipv4[-1].ipid, tcp[-1].seq) pre/post middlebox<br>*Query:* exist ingress and egress trace | Yes | Yes | 8 |
| | Protocol bugs checker(BGP, RDMA, etc) [69]<br>*Identify wrong implementation of protocols* | *Group:* all BGP packets at target switch(SWITCH=tar_SW)<br>*Query:* correctness(related fields) of BGP(FLTR: tcp[-1].src\|dst=179) | Yes | Yes* | 23‡ |
| | Incorrect packet modification [21]<br>*Check packets' header modification* | *Group:* same packet(ipv4[-1].ipid, tcp[-1].seq) pre/post the modifier<br>*Query:* is modification correct (related fields) | Yes | Yes* | 4◇ |
| | Waypoint routing checker [21,43]<br>*Make sure packets (not) pass a waypoint* | *Group:* all packets at waypoint switch(SWITCH=waypoint)<br>*Query:* contain flow(ipv4[-1].src+dst, tcp[-1].src+dst) should (not) pass | Yes | No | 11 |
| | DDoS diagnosis [43]<br>*Localize DDoS attack based on statistics* | *Group:* all packets at victim's ToR(SWITCH=vic_ToR)<br>*Query:* statistics of flow(ipv4[-1].src+dst, tcp[-1].src+dst) | No | Yes* | 18 |
| Multiple packets on multiple hops | Congested link diagestion [43]<br>*Find flows using congested links* | *Group:* all packets(ipv4[-1].ipid, tcp[-1].seq) pass congested link<br>*Query:* list of flows(ipv4[-1].src+dst, tcp[-1].src+dst) | No* | Yes* | 14 |
| | Silent black hole localizer [43,69]<br>*Localize switches that drop all packets* | *Group:* packets with duplicate TCP(ipv4[-1].ipid, tcp[-1].seq)<br>*Query:* get dropped hop in the recovered path(ipv4[:].ttl) | No* | Yes* | 52 |
| | Silent packet drop localizer [69]<br>*Localize random packet drops* | *Group:* packets with duplicate TCP(ipv4[-1].ipid, tcp[-1].seq)<br>*Query:* get dropped hop in the recovered path(ipv4[:].ttl) | No* | Yes* | 52 |
| | ECMP profiler [69]<br>*Profile flow distribution on ECMP paths* | *Group:* all packets at ECMP ingress switches(SWITCH in ECMP)<br>*Query:* statistics of flow(ipv4[-1].src+dst, tcp[-1].src+dst) | No* | No | 18 |
| | Traffic matrix [43]<br>*Traffic volume between given switch pairs* | *Group:* all packets at given two switches(SWITCH in tar_SW)<br>*Query:* total volume of overlapped flow(ipv4[-1].src+dst, tcp[-1].src+dst) | No* | Yes* | 21 |

Please check the paper for details!

# dShark: three goals

## Scalable

- Components work independently and in parallel.

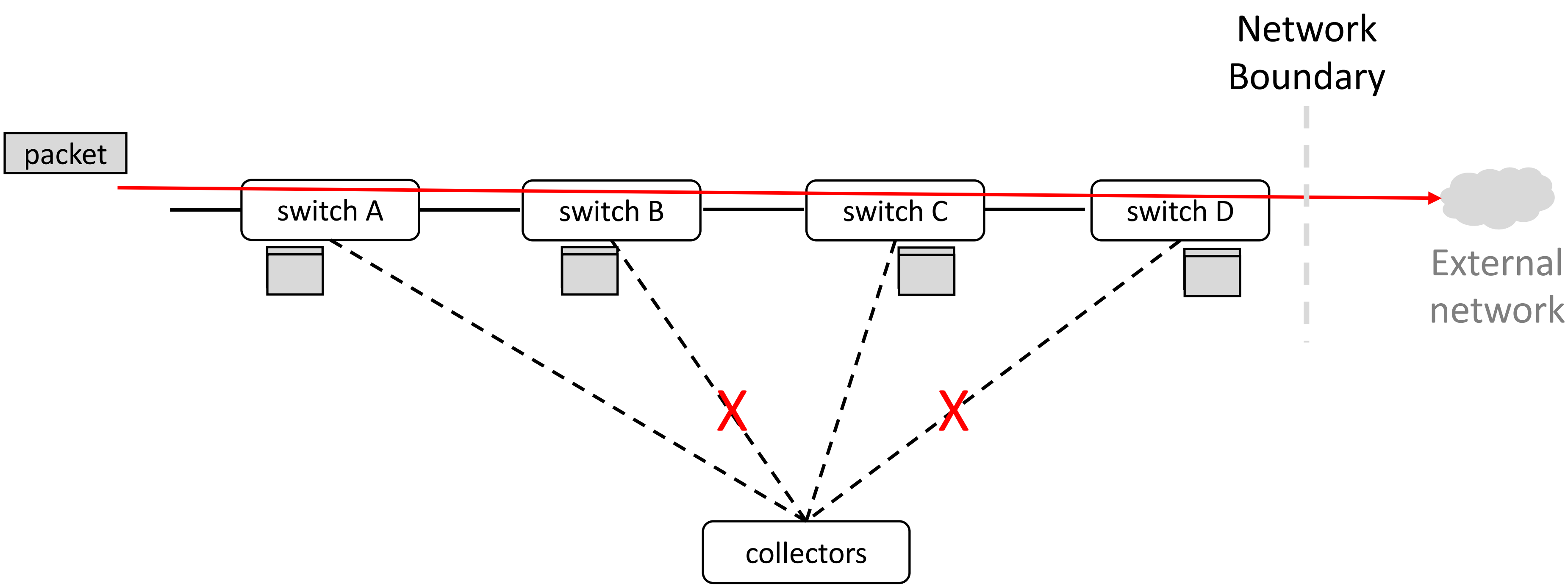
## Broadly applicable

- dShark's programming model is general.
- It supports 4 types of aggregation that covers 18 typical analysis apps.

## Robust in the wild

- Header transformation
  - Define packet signature in the summary.
  - Leverage the index of protocol.
- Capture noise

# Tolerate capture noise



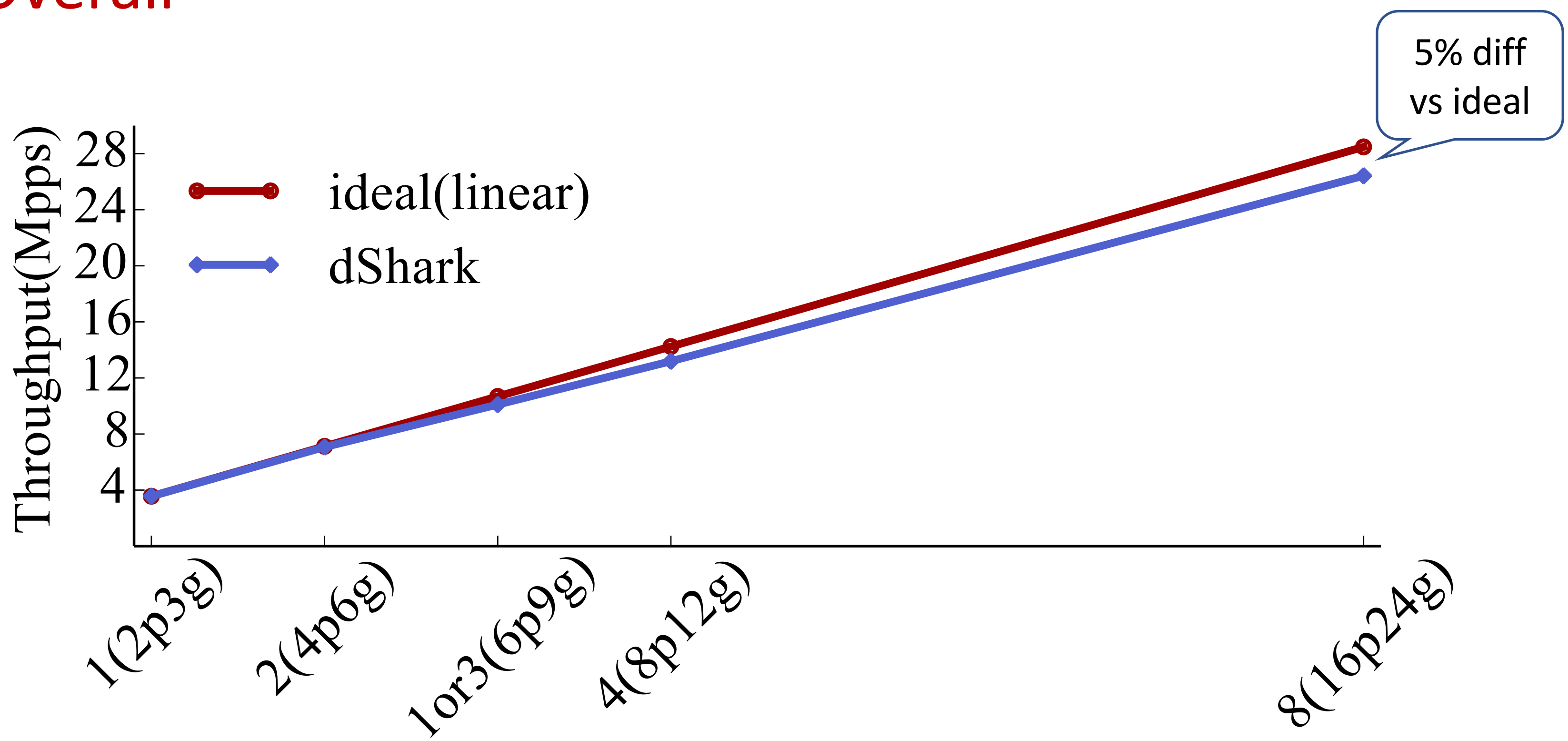

1. Recover by the next hop(s)
2. Leverage end-to-end information

# Performance of dShark

- 8 VMs from a public cloud
- Each has:
  - 16-core 2.4GHz vCPU
  - 56GB memory
  - 10Gbps virtual network
- Feed with real traces from production

# dShark scales nearly linearly

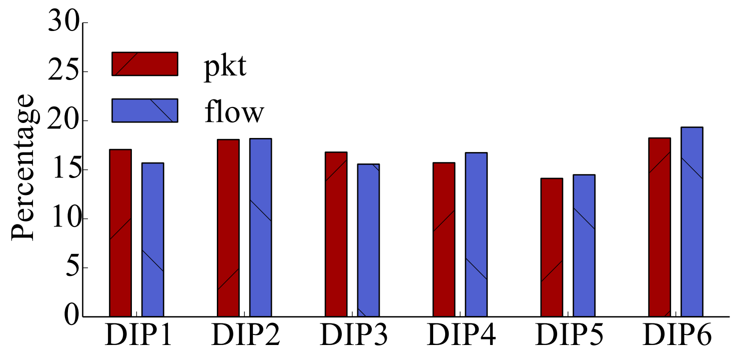
## Overall



# Some findings

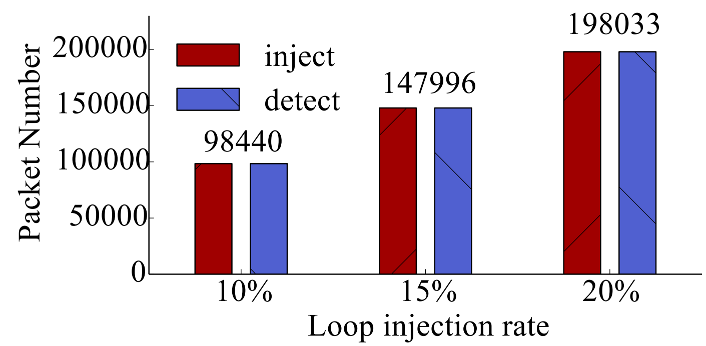
## Case 1:

Profile an SLB




## Case 2:

Detect loops




## Case 3:

Detect packet drops on T1

Retran. as evidence of actual drops

| | |
|---|---|
| Naïve | 5,588 |
| dShark | 7 |

Diff. caused by noise

Please check the paper for details!!

# Conclusion

- dShark is a general, easy-to-program, scalable and high-performance in-network packet trace analyzer.

- Takeaways:
  - dShark's programming model is broadly applicable
    - We use this model to implement 18 different typical diagnosis apps
  - Operators focus on the logic without worrying about:
    - Header transformation, capture noise, scalability
  - dShark is fast and can scale linearly