Traffic Exceptions

Mat Wood

Network Automation Engineer @ Facebook

Hackathon Agenda

Schedule

9:30 am (now)	Introduction & Theme Topic Group Assembly Tutorial & Demos
10:30 am	Break into groups
12:30 pm	Lunch
1:30 pm	Resume Groups
3:00 pm	Break/Refreshments
6:00 pm	Hack Deadline Prototype Demos Voting Raffle

Group Assembly

- Pitch ideas to recruit for your group
- Create a group Slack channel
  - https://nanoghackathons.slack.com/
  - Reach out if you don’t have an account
- Groups wanting to start working early can break off
  - Please use Slack channel for comms during the Hack Tutorial

Hack Time

- Reach out for help with:
  - Code & configs
  - Tesuto Lab Resources (Including custom labs)
  - In Slack Channel
- Work on your idea until 6pm
- Make sure to save time for your presentation!

Prototype Demos

- 5-10 minute presentation
  - What does your Hack do?
  - How did you do it?
- Make sure to take screenshots along the way
  - Live demos are unpredictable
  - Labs will stop being available

Voting

- Crowd vote for favorite Hack
- Winning team(s) to give Prototype Presentation
  - Wednesday, 3pm
  - Lone Star Salon D-H, Level 3

Raffle

- Prizes will be raffled after Prototype Demos
- Tickets you received at registration
- Must be present to win

Handling
Traffic Exceptions

Traditional Routing

- Routing is prescriptive of pre-defined desired topology
  - Protocols and costs define desired traffic flow
  - BGP Policy expresses business logic as reachability
  - TE adds constraints to path selection

- Reactive scenarios focus around link failure
  - Solving: How to retain connectivity & capacity
  - IGP reconvergence of CSPF
  - LSP signaled over available capacity
  - Try to get back to desired topology

What if we could react to individual traffic flows?

Handling Traffic Exceptions

- Traffic Triggering
  - Monitor traffic flows and flag based on desired characteristics
- Network Config
  - Supports the desired outcome of triggered flows
  - E.g. Redirect traffic to desired network segments
- Traffic Influence
  - Mechanism to connect the triggering to the network data plane

Wait, this looks familiar…

DDoS Mitigation

- Traffic Triggering
  - Detect attacks from rules/machine learning
  - Customer phone call
- Network Config
  - BGP with pre-defined policy & communities to drop traffic
- Traffic Influence
  - Remotely-Triggered Black Hole (RTBH)
  - BGP FlowSpec
    - Remote programming of Drop/Rate-limit for flows

We can do so much more!

Group Assembly

Group Pitch

- Your Name
- Your Project Idea
- What you would like help with
- Slack group channel name

Demo:
Malicious Domains

`bit.ly/nanog77-demo-dns`

Disclaimer

No parts of this demo are representative of Facebook’s network

Demo

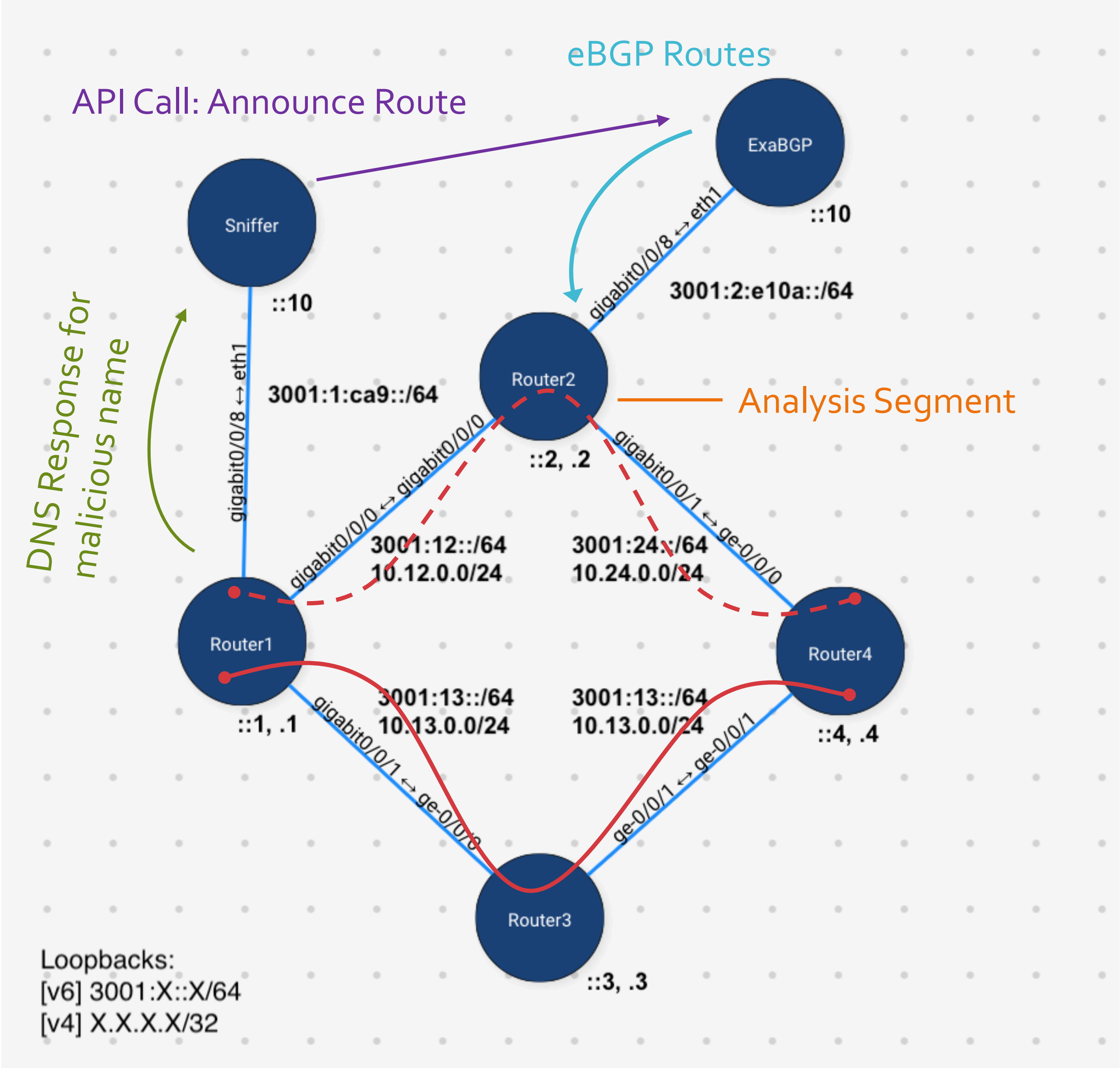
Goals

- Inspect DNS responses for malicious domain name requests (blacklisted)
- Subsequent flows to the resolved host should be monitored
- Redirect traffic **destined towards the host** to the monitoring network segment

Demo

Technologies

- The API we already know and love:
  - BGP
- ExaBGP as a route injector
  - Add HTTP endpoint for remote commands
- Python + Scapy for sniffing and flagging interesting traffic



API Call: Announce Route

eBGP Routes

DNS Response for malicious name

Analysis Segment

Loopbacks:
[v6] 3001:X::X/64
[v4] X.X.X.X/32

Traffic Triggering

Traffic Triggering

Goals

- Detect Interesting Traffic
  - DNS Responses for blacklisted domains
- Python + Scapy script is the start of traffic influence pipeline
  - Use existing libraries like Scapy
  - Focus on the business logic

Traffic Triggering

DNS Response Analysis

```
BAD_QUERIES = set([
    "badhacks.com.",
    "malicious-mail-order.net.",
])


def analyze(packet: Packet) -> Optional[str]:
    """ Check for malicious DNS query/response.
        If this is a DNS response for blacklisted
        domain, return resolved IP address
    """
    if packet.haslayer(DNS):
        if not packet[DNS].qr or not packet[DNS].qd:
          return  # Nothing we’re interested in

        if packet[DNS].qd.qname.decode() in BAD_QUERIES:
            return packet[DNS].an.rdata
```

Traffic Triggering

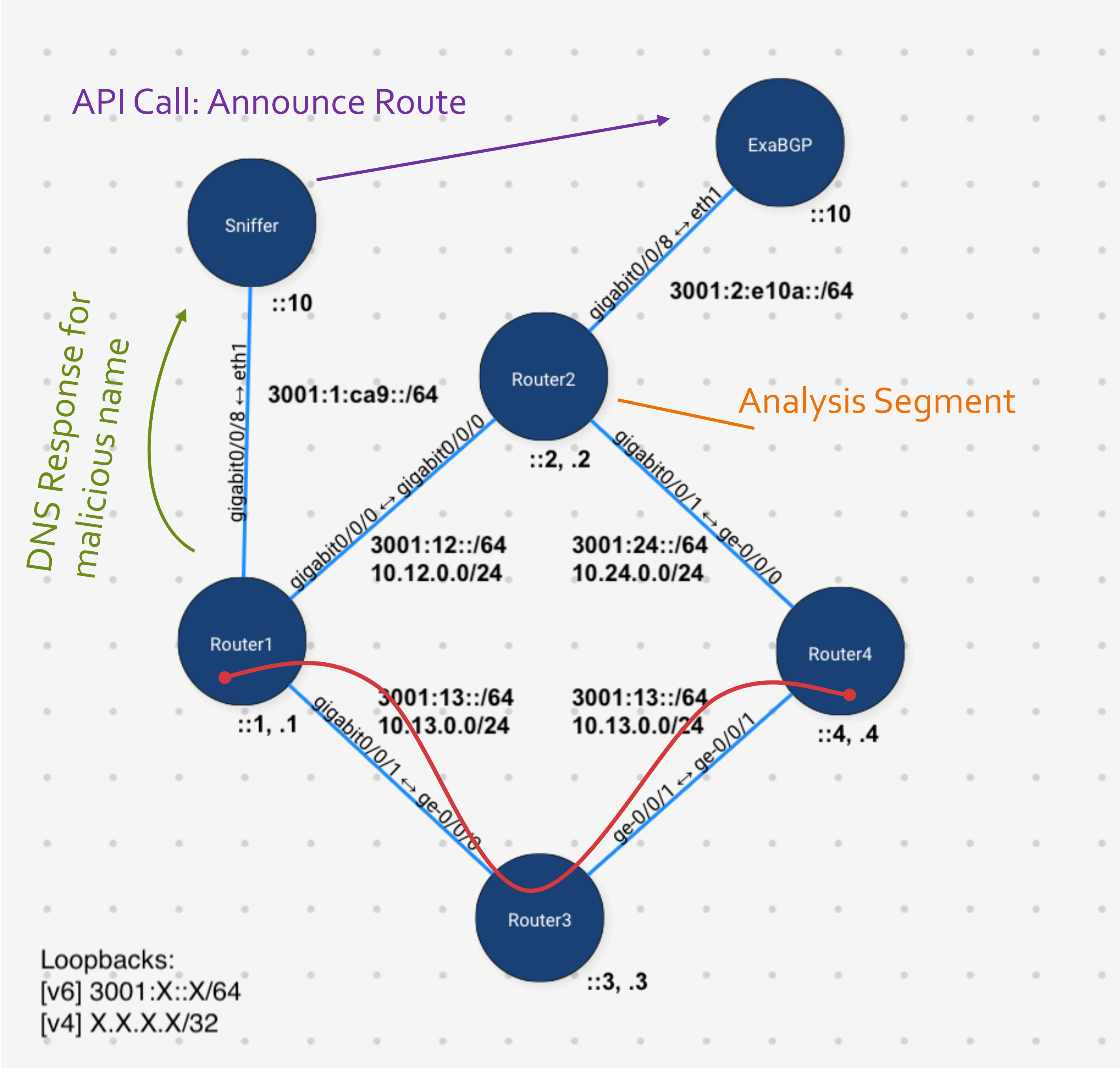
Scapy Per-packet Processing

```
def process_packet(packet: Packet) -> Optional[str]:
    """ Process the incoming packet """
    dest_ip = analyze(packet)
    if dest_ip:
        trigger_exabgp(dest_ip)
```

```
def trigger_exabgp(dst_ip: str):
    """ Send announcement to ExaBGP """
    command = f"announce route {dst_ip}/128 next-hop self"

    params = urlencode({"command": command})
    client = HTTPConnection(EXABGP_HOST)
    client.request("POST", "/command", params)
```

```
scapy.sniff(filter="udp src port 53", prn=process_packet)
```

Traffic Triggering



Traffic Influence

Traffic Influence

Goals

- Receive detected routes and inject into BGP
- Redirect traffic destined for the malicous host
- Use ExaBGP to inject traffic redirects

Traffic Influence

ExaBGP HTTP API

```python
# HTTP API for ExaBGP

from flask import Flask, request
from sys import stdout

app = Flask(__name__)

# Setup a ‘command’ route for prefix advertisements
@app.route("/command", methods=["POST"])
def command():
    command = request.form["command"]
    # Write command to stdout for ExaBGP
    stdout.write(f"{command}\n")
    stdout.flush()
    return f"{command}\n"

if __name__ == "__main__":
    app.run(host="3001:2:e10a::10", port=5000)
```

Traffic Influence

exabgp-conf.ini

```
process http-api {
    run /usr/bin/python3 $HOME/http_api.py;
    encoder json;
}

# Router2
neighbor 3001:2:e10a::2 {
    router-id 10.10.10.10;
    local-address 3001:2:e10a::10;
    local-as 65010;
    peer-as 65000;

    family {
        ipv4 unicast;
        ipv6 unicast;
    }

    announce {
        ipv6 { # Test routes
            unicast 3001:99:a::/64 next-hop self;
            unicast 3001:99:b::/64 next-hop self;
        }
    }
}
```

Network Config

Network Config

Router2 config

```
route-policy exabgp
  if source in (3001:2:e10a::10) then
    set local-preference 4294967295
  endif
  pass
end-policy
!
router bgp 65000
 neighbor 3001:2:e10a::10
  description ExaBGP Peering
  remote-as 65010
  !
  address-family ipv4 unicast
   route-policy exabgp in
   route-policy exabgp out
   next-hop-self
  !
  address-family ipv6 unicast
   route-policy exabgp in
   route-policy exabgp out
   next-hop-self
  !
 !
!
```

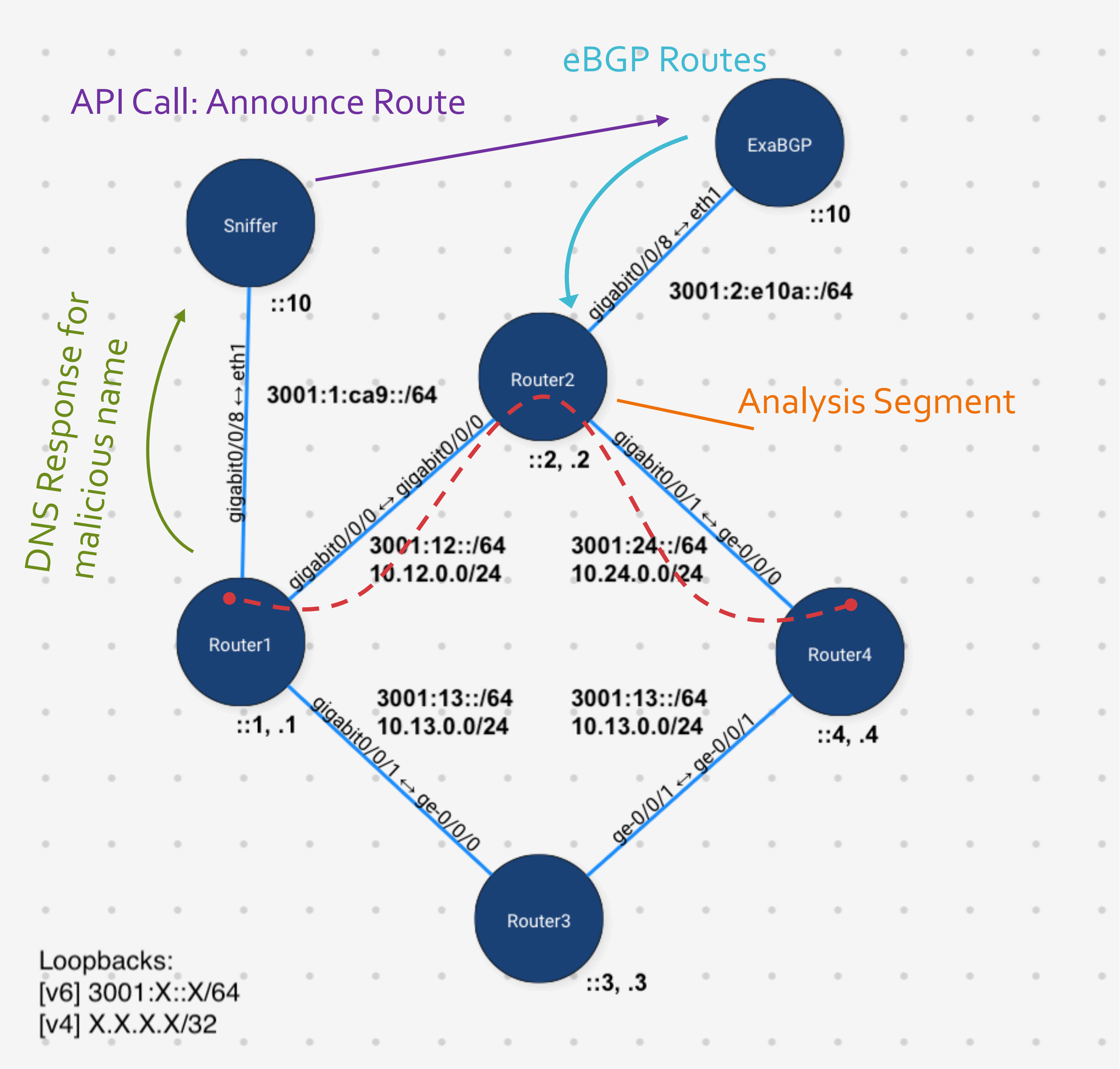
ExaBGP Peer Verification

Network Config

```
router2#show bgp ipv6 unicast summary | b Neighbor
Neighbor        Spk    AS Up/Down  St/PfxRcd
3001:1::1         0 65000 00:56:50         1
3001:2:e10a::10   0 65010 00:00:45         2
3001:3::3         0 65000 00:56:45         0
3001:4::4         0 65000 00:56:34         0
```

```
router2#show bgp ipv6 uni | b Network
Network               Next Hop          LocPrf Weight Path
*>i3001:1:ca9::/64    3001:1::1         0      100      0    i
*> 3001:2:e10a::/64   ::                0      32768         i
*> 3001:99:a::/64     3001:2:e10a::10          0       65010 i
*> 3001:99:b::/64     3001:2:e10a::10          0       65010 i
```

Traffic Triggering



See it in Action

Automatic Triggering

See it in Action

```
sniffer$ ./detect_dns.py traffic.pcap
INFO:root:Detecting DNS queries from traffic.pcap...
WARNING:root:Request for badhacks.com.: 3001:10:66::5
DNS Response with 3001:10:66::5 is a malicious query
```

```
router2#show bgp ipv6 uni 3001:10:66::5/128
BGP routing table entry for 3001:10:66::5/128
Versions:
  Process           bRIB/RIB  SendTblVer
  Speaker                 51          51
Paths: (1 available, best #1)
  65010
    3001:2:e10a::10 from 3001:2:e10a::10 (10.10.10.10)
    Origin IGP, localpref 4294967295, valid, external, best
    Received Path ID 0, Local Path ID 1, version 51
    Origin-AS validity: (disabled)
```

See it in Action

BGP Advertisement Verification

```
router4> show route protocol bgp 3001:10:66::5

inet6.0: 24 destinations, 24 routes (24 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

3001:10:66::5/128  *[BGP/170] 00:03:40, localpref 4294967295, from 3001:2::2
                    AS path: 65010 I, validation-state: unverified
                    >  to fe80::9099:ff:fe07:1 via ge-0/0/0.0
```

```
router1#show bgp ipv6 uni 3001:10:66::5/128
BGP routing table entry for 3001:10:66::5/128
Versions:
  Process           bRIB/RIB  SendTblVer
  Speaker                 51          51
Paths: (1 available, best #1)
  65010
      3001:2:e10a::10 (metric 1) from 3001:2::2 (2.2.2.2)
      Origin IGP, localpref 4294967295, valid, internal, bestst
      Received Path ID 0, Local Path ID 1, version 51
```

Shortcomings

- BGP Unicast Routes aren’t granular enough
    - /32 and /128 could affect more traffic than we want
- We can’t influence based on source prefix
    - Especially not individual traffic flows

Reactive Network

- Traffic Triggering
  - Malicious L7 API requests
  - TCP Retransmits, further analysis
  - TTL as source-defined priority
    - Higher TTL implies “scenic route” 😂
  - TCP options encoding of a BGP Community?
    - Intent Based Networking™
- Network Config
  - Network segment(s) attract traffic via BGP FlowSpec
- Traffic Influence
  - ExaBGP provides an API to advertise FlowSpec rules

Flowspec

- RFC 5575
- Flow Specification Rules via BGP
    - AFIs: IPv4 & IPv6
    - SAFI: 133, 134
- NLRI contains list of Match criteria
- Extended Communities specify the action
- Installed on client/edge devices

Flowspec Matches

- Dest and/or Source prefix
- IP Protocol
- Dest and/or Source port
- ICMP type/code
- TCP Flags
- Packet Length, DSCP, Fragments

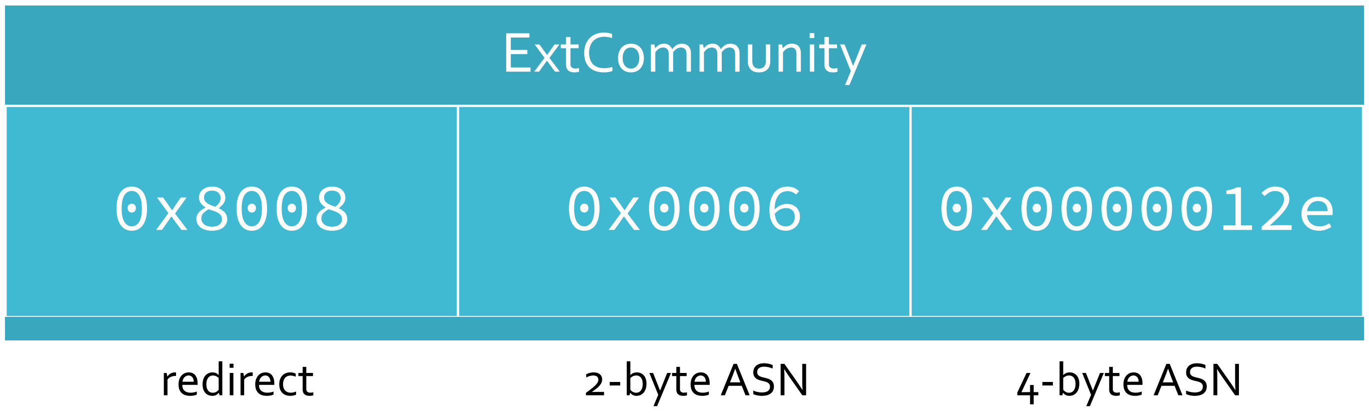
- Operators
  - Numeric: `==`, `>`, `<`
  - Boolean: AND, NOT

Flowspec Actions

- Traffic Rate
  - Bytes-per-second (zero == discard)
- Action
  - Terminal action (don’t process more rules)
  - Sample (for logging purposes)
- Redirect
  - Allows for redirection into a VRF
- Traffic Marking
  - Apply DSCP value to matching packets

Flowspec Actions

- Redirect Community
  - **`redirect:6:302`**



ExtCommunity		
0x8008	0x0006	0x0000012e
redirect	2-byte ASN	4-byte ASN

Demo:
TCP Retransmits

bit.ly/nanog77-demo-flowspec

Disclaimer

No parts of this demo are representative of Facebook’s network

Demo

Goals

- Inspect flows for high TCP Retransmits
- Traffic Analysis segment with additional monitoring/troubleshooting tools
- Redirect **interesting flows** to the monitoring network segment

Demo

Technologies

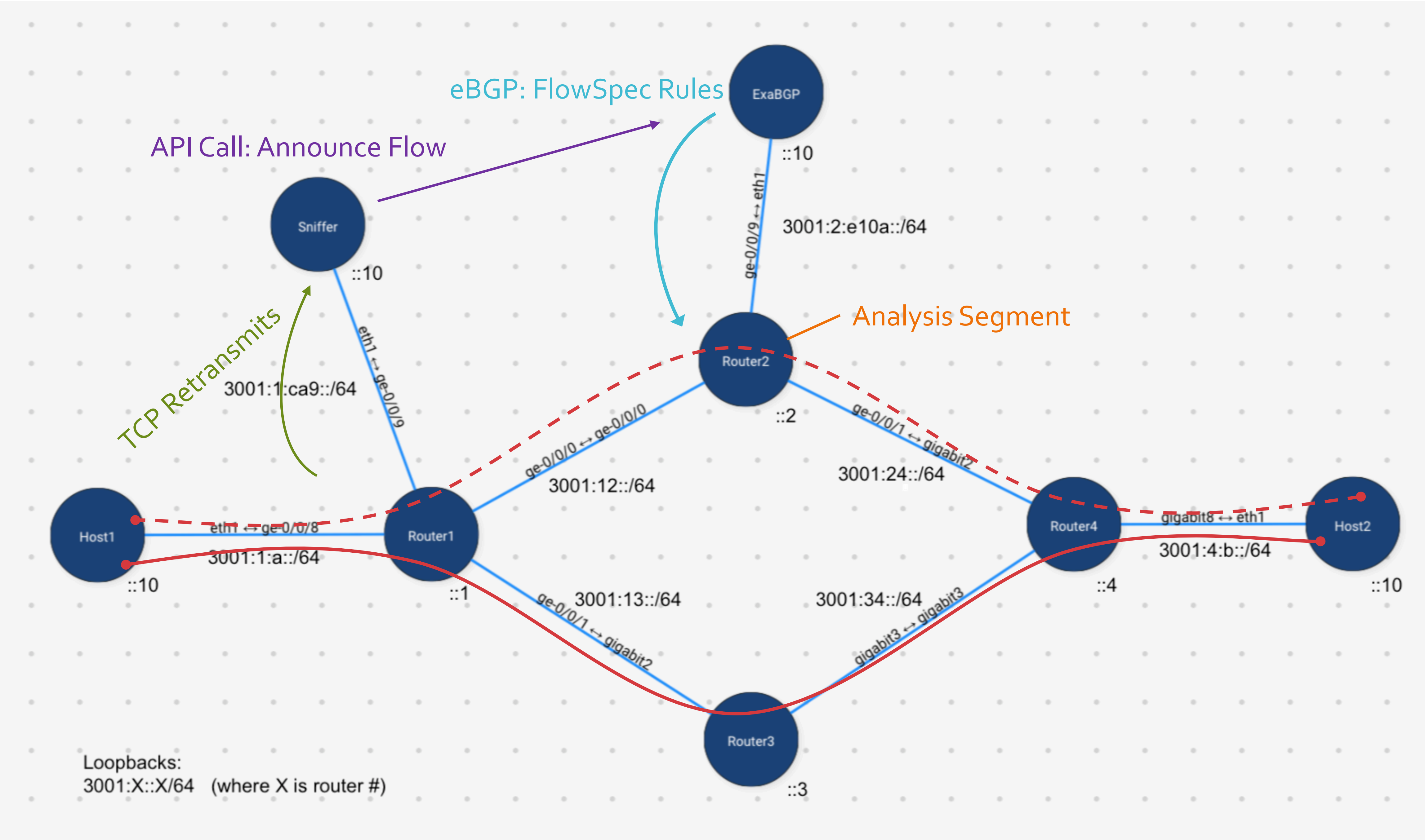
- The API we already know and love:
  - BGP (now with FlowSpec)
- ExaBGP as a route injector
  - Add HTTP endpoint for remote commands
- Python + Scapy for sniffing and flagging interesting traffic

Demo

FlowSpec

- N-Tuple Matching
  - Src/dst IP, Src/dst port, DSCP, etc
- Match Actions
  - Drop/Rate-limit (DDoS mitigation)
  - Traffic Marking
  - Redirect (next-hop)
- Propagated via BGP
  - Communities express the desired action

Great NANOG Talk: DDoS Mitigation using BGP FlowSpec



API Call: Announce Flow

eBGP: FlowSpec Rules

TCP Retransmits

Analysis Segment

Loopbacks:
3001:X::X/64 (where X is router #)

Traffic Triggering

Traffic Triggering

Goals

- Detect Interesting Traffic
  - TCP flows with high # of Retransmits
- Python script for easy business logic as start of traffic influence pipeline
  - Use existing libraries like Scapy

Traffic Triggering

TCP Flow Records

```python
class FlowKey(NamedTuple):
    """ Flow Signature """
    src_ip: str
    src_port: int
    dest_ip: str
    dest_port: int
```

```python
class FlowStatus(object):
    """ Flow Object to keep track of retransmits """

    def __init__(self) -> None:
        self.retransmits = 0
        # Sequence is Tuple[seq, ack]
        self.last_sequence: Tuple(int, int) = (0, 0)

        # Has been sent to ExaBGP?
        self.has_been_triggered = False
```

Traffic Triggering

TCP Retransmit Detection

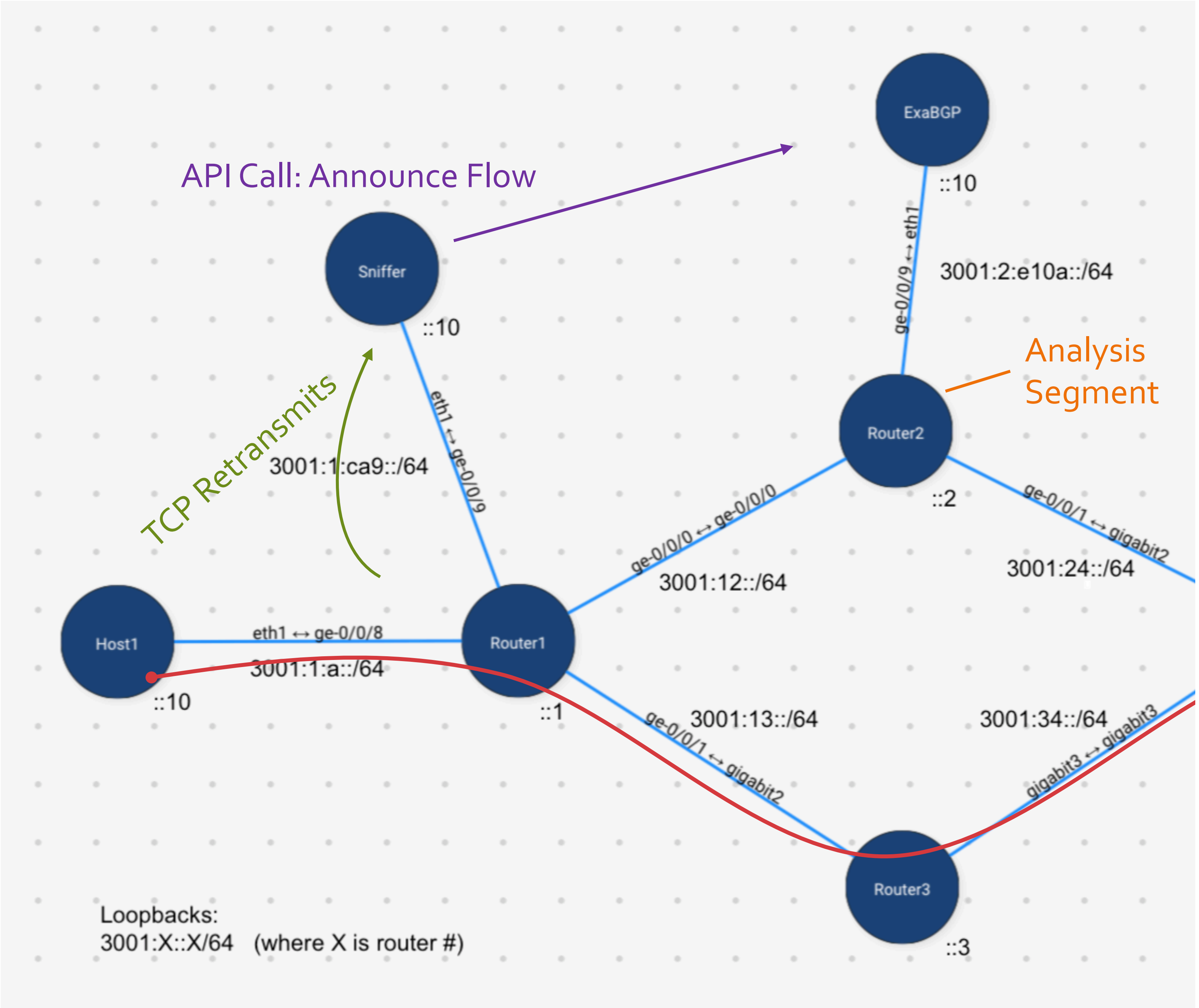
```
class FlowStatus(object):
    # ...
    def analyze(self, packet: Packet) -> int:
        """ Detect retransmits

            Returns current TCP retransmit count
        """
        sequence = (packet[TCP].seq, packet[TCP].ack)
        if sequence > self.last_sequence:
            self.last_sequence = sequence
        else:
            self.retransmits += 1

        if packet[TCP].flags.F or packet[TCP].flags.R:
            raise SessionTerminated()

        return self.retransmits
```

Traffic Triggering

Scapy Per-packet Processing

```
flows: Dict[FlowKey, FlowStatus] = {}

def process_packet(packet: Packet) -> Optional[str]:
    key = FlowKey.from_packet(packet)

    if key not in flows:
        flows[key] = FlowStatus() # Init a new flow

    try:
        flow_retransmits = flows[key].analyze(packet)
        if flow_retransmits >= RETRANSMIT_THRESHOLD:
            if not flows[key].has_been_triggered:
                trigger_exabgp(key)
            flows[key].has_been_triggered = True
            return f"Flow {key!r} has retransmits!"
    except SessionTerminated:
        del flows[key]


scapy.sniff(filter="tcp", prn=process_packet)
```

Traffic Triggering



Traffic Influence

Traffic Influence

Goals

- Receive detected FlowSpec flows
  - Allow for redirect ingTCP flows with high retransmits
- Use ExaBGP to inject FlowSpec flows for traffic redirection

Traffic Influence

ExaBGP HTTP API

```
# HTTP API for ExaBGP

from flask import Flask, request
from sys import stdout

app = Flask(__name__)

# Setup a ‘command’ route for prefix advertisements
@app.route("/command", methods=["POST"])
def command():
    command = request.form["command"]
    # Write command to stdout for ExaBGP
    stdout.write(f"{command}\n")
    stdout.flush()
    return f"{command}\n"

if __name__ == "__main__":
    app.run(host="3001:2:e10a::10", port=5000)
```

Traffic Influence

exabgp-conf.ini (part1)

```
process http-api {
    run /usr/bin/python3 $USER/http_api.py;
    encoder json;
}

# Router2
neighbor 3001:2:e10a::2 {
    router-id 10.10.10.10;
    local-address 3001:2:e10a::10;
    local-as 65010;
    peer-as 65000;

    family {
        ipv4 unicast;
        ipv4 flow;
        ipv6 unicast;
        ipv6 flow;
    }



...
```

Traffic Influence

exabgp-conf.ini (part2)

```
...
    announce {
        ipv6 {
            # Test routes
            unicast 3001:99:a::/64 next-hop self;
            unicast 3001:99:b::/64 next-hop self;
        }
    }

    # Test Flows
    flow {
        route TEST {
            match {
                source 3001:99:a::10/128;
                destination 3001:99:b::10/128;
            }
            then {
                redirect 6:302;
            }
        }
    }
}
```

Network Config

Network Config

Router2 config

```
protocols {
    bgp {
        group exabgp {
            type external;
            import [ FLOWSPEC EXABGP ];
            family inet6 {
                unicast;
                flow {
                    no-validate FLOWSPEC;
                }
            }
            peer-as 65010;
            neighbor 3001:2:e10a::10 {
                local-address 3001:2:e10a::2;
            }
        }
        group internal-peers {
            family inet6 {
                unicast;
                flow;
            }
            ...
        }
    }
}
```

Network Config

Router2 config

```
policy-options {
    policy-statement EXABGP {
        term 1 {
            from neighbor 3001:2:e10a::10;
            then { local-preference 4294967295; }
        }
        term 2 { then accept; }
    }
    policy-statement FLOWSPEC {
        term 1 {
            from community TCP-REDIRECT;
            then {
                next-hop self;
            }
        }
        term 2 {
            from community TCP-REDIRECT;
            then {
                local-preference 4294967295;
                accept;
            }
        }
        term 3 { then accept; }
    }
    community TCP-REDIRECT members redirect:6:302;
}
```

Network Config

ExaBGP Peer Verification

```
router2> show bgp summary
...
Peer                  AS     InPkt   OutPkt Last Up/Dwn State...
3001:2:e10a::10    65010 38      36             16:27 Establ
    inet6.0: 2/2/2/0
    inet6flow.0: 0/0/0/0
```

```
router2> show route protocol bgp all table inet6

inet6.0: 25 destinations, 25 routes (25 active, 0 holddown, 0
hidden)
+ = Active Route, - = Last Active, * = Both

3001:99:a::/64     *[BGP/170] 00:18:00, localpref 100
                    AS path: 65010 I, validation-state: unverified
                    >  to 3001:2:e10a::10 via ge-0/0/9.0
3001:99:b::/64     *[BGP/170] 00:17:59, localpref 100
                    AS path: 65010 I, validation-state: unverified
                    >  to 3001:2:e10a::10 via ge-0/0/9.0
```

Network Config

Router1 config

```
protocols {
    bgp {
        group internal-peers {
            type internal;
            local-address 3001:1::1;
            family inet6 {
                unicast;
                flow {
                    no-validate FLOWSPEC;
                }
            }
            export CONNECTED;
            neighbor 3001:2::2;
            neighbor 3001:3::3;
            neighbor 3001:4::4;
        }
    }
}
policy-options {
    policy-statement FLOWSPEC {
        term 1 {
            from community TCP-REDIRECT;
            then { next-hop peer-address; }
        }
        term 2 { then accept; }
    }
    community TCP-REDIRECT members redirect:6:302;
}
```

Network Config

Router1 config

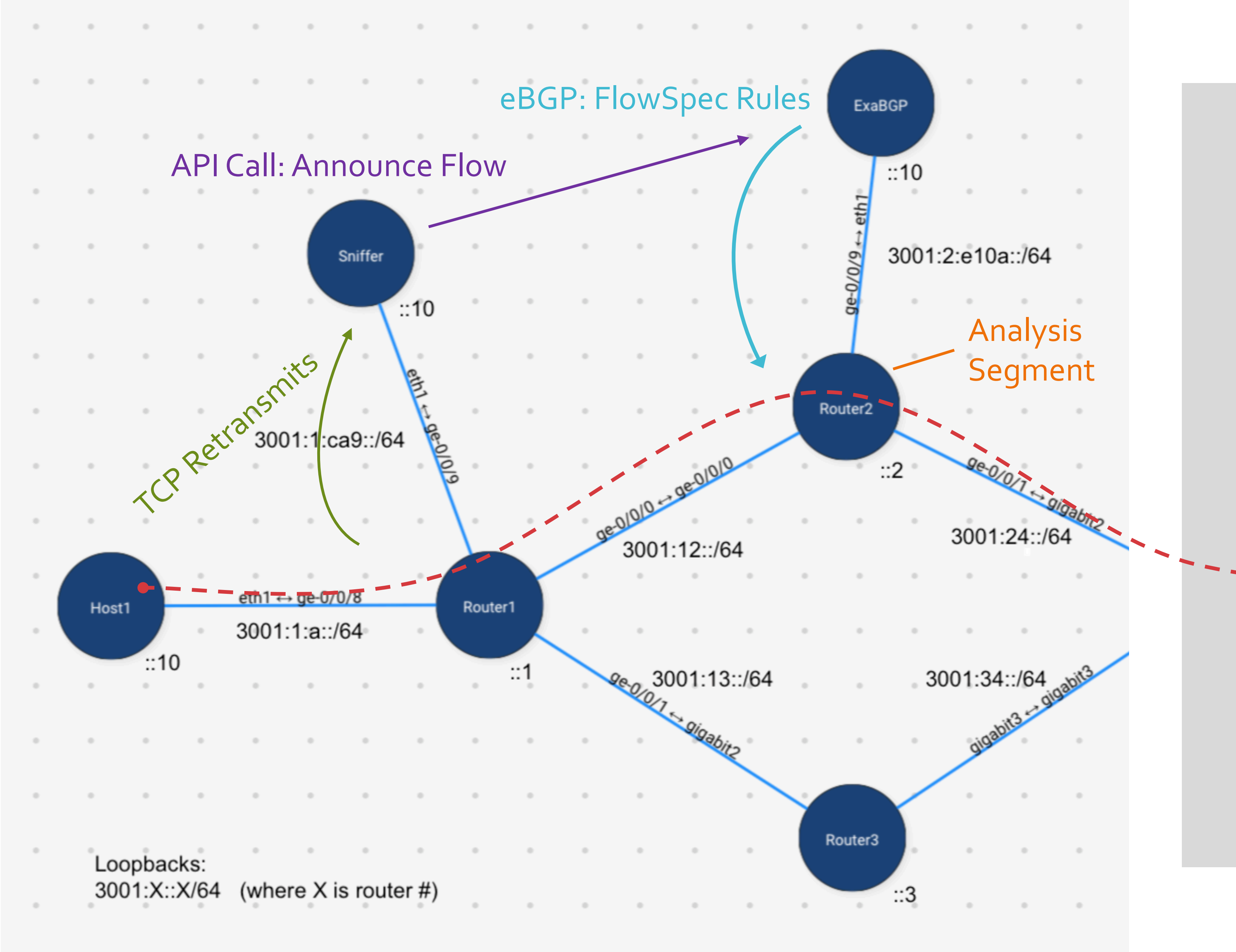
```
routing-instances {
    flowspec-redirect {
        instance-type vrf;
        interface lo0.302;
        route-distinguisher 6:302;
        vrf-target target:6:302;
        routing-options {
            rib flowspec-redirect.inet.0;
            rib flowspec-redirect.inet6.0 {
                static {
                    defaults {
                        resolve;
                    }
                    route ::/0 {
                        next-hop 3001:2::2;
                        resolve;
                    }
                }
            }
            resolution {
                rib flowspec-redirect.inet6.0 {
                    resolution-ribs inet6.0;
                }
            }
        }
    }
}
```

Network Config

Router4 config

```
flowspec
 local-install interface-all
!
router bgp 65000
 address-family ipv4 unicast
  network 4.4.4.4/32
 !
 address-family ipv6 unicast
 !
 address-family ipv6 flowspec
 !
 session-group internal-peers
  remote-as 65000
  update-source Loopback0
 !
 neighbor-group internal-peers
  use session-group internal-peers
  address-family ipv6 unicast
  !
  address-family ipv6 flowspec
  !
 !
 neighbor ...
  use neighbor-group internal-peers
 !
!
```

Traffic Triggering



See it in Action

See it in Action

Steady State

```
router1> show route table inet6flow.0
router1>
```

```
router4#show bgp ipv6 flow
router4#
```

```
host1$ traceroute -s 3001:1:a::10 3001:4:b::10
traceroute to 3001:4:b::10 (3001:4:b::10), 30 hops max, 80 byte
packets
 1  3001:1:a::1 (3001:1:a::1)  6.959 ms  6.915 ms  6.888 ms
 2  3001:13::3 (3001:13::3)  14.177 ms  14.120 ms  14.123 ms
 3  3001:34::4 (3001:34::4)  14.091 ms  14.062 ms  14.044 ms
 4  3001:4:b::10 (3001:4:b::10)  22.202 ms  22.186 ms  22.169 ms

$ traceroute -s 3001:1:a::20 3001:4:b::10
traceroute to 3001:4:b::10 (3001:4:b::10), 30 hops max, 80 byte
packets
 1  3001:1:a::1 (3001:1:a::1)  7.885 ms  7.730 ms  7.756 ms
 2  3001:13::3 (3001:13::3)  23.147 ms  23.121 ms  23.099 ms
 3  3001:34::4 (3001:34::4)  23.053 ms  22.991 ms  23.010 ms
 4  3001:4:b::10 (3001:4:b::10)  22.994 ms  22.963 ms  22.946 ms
```

See it in Action

Traffic Redirection

```
sniffer$ curl --form \
  "command=announce flow route source 3001:1:a::10/128 \
   destination 3001:4:b::10/128 redirect 6:302" \
   http://[3001:2:e10a::10]:5000/command
```

```
router1> show route table inet6flow.0

inet6flow.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

3001:1:a::10/128,3001:4:b::10/128/term:1
                *[BGP/170] 00:38:34, localpref 65000
                    AS path: 65010 I, validation-state: unverified
                    >  to 3001:2::2
```

```
router4#show bgp ipv6 flow | b Network
     Network          Next Hop            Metric LocPrf Weight Path
 * i  Dest:3001:4:B::10/0-128,Source:3001:1:A::10/0-128
                      3001:2::2                    65000      0 65010 i
```

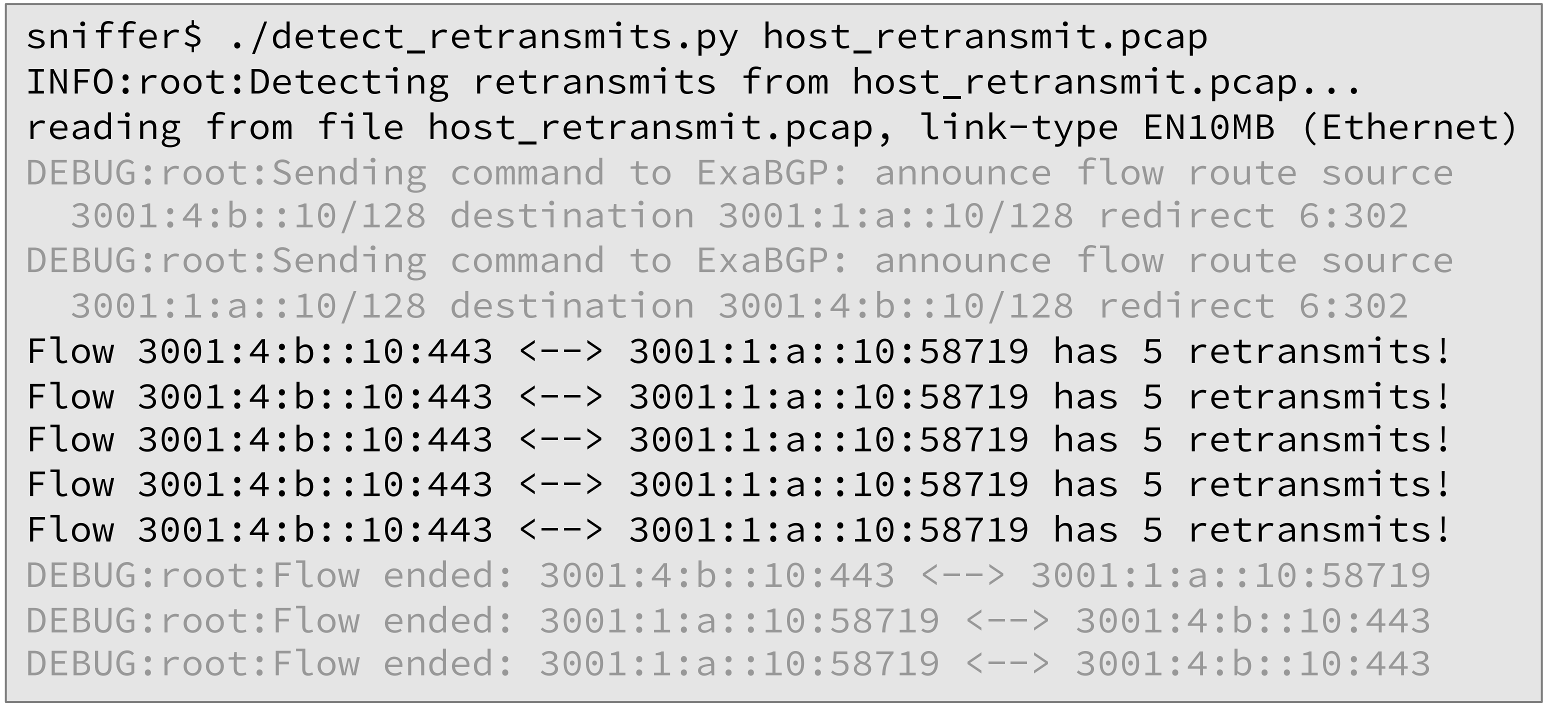
See it in Action

Traffic Redirection

```
host1$ traceroute -s 3001:1:a::10 3001:4:b::10
traceroute to 3001:4:b::10 (3001:4:b::10), 30 hops max, 80 byte
packets
 1  3001:1:a::1 (3001:1:a::1)  2.321 ms  2.241 ms  2.208 ms
 2  3001:12::2 (3001:12::2)  9.576 ms  9.544 ms  9.499 ms
 3  3001:24::4 (3001:24::4)  21.666 ms  21.637 ms  21.618 ms
 4  * 3001:4:b::10 (3001:4:b::10)  21.559 ms  21.502 ms

host1$ traceroute -s 3001:1:a::20 3001:4:b::10
traceroute to 3001:4:b::10 (3001:4:b::10), 30 hops max, 80 byte
packets
 1  3001:1:a::1 (3001:1:a::1)  7.527 ms  7.399 ms  7.399 ms
 2  3001:13::3 (3001:13::3)  14.992 ms  14.953 ms  14.955 ms
 3  3001:34::4 (3001:34::4)  30.839 ms  30.804 ms  30.805 ms
 4  3001:4:b::10 (3001:4:b::10)  22.710 ms  22.618 ms  22.583 ms
```

See it in Action

Automatic Triggering



```
sniffer$ ./detect_retransmits.py host_retransmit.pcap
INFO:root:Detecting retransmits from host_retransmit.pcap...
reading from file host_retransmit.pcap, link-type EN10MB (Ethernet)
DEBUG:root:Sending command to ExaBGP: announce flow route source
  3001:4:b::10/128 destination 3001:1:a::10/128 redirect 6:302
DEBUG:root:Sending command to ExaBGP: announce flow route source
  3001:1:a::10/128 destination 3001:4:b::10/128 redirect 6:302
Flow 3001:4:b::10:443 <--> 3001:1:a::10:58719 has 5 retransmits!
Flow 3001:4:b::10:443 <--> 3001:1:a::10:58719 has 5 retransmits!
Flow 3001:4:b::10:443 <--> 3001:1:a::10:58719 has 5 retransmits!
Flow 3001:4:b::10:443 <--> 3001:1:a::10:58719 has 5 retransmits!
Flow 3001:4:b::10:443 <--> 3001:1:a::10:58719 has 5 retransmits!
DEBUG:root:Flow ended: 3001:4:b::10:443 <--> 3001:1:a::10:58719
DEBUG:root:Flow ended: 3001:1:a::10:58719 <--> 3001:4:b::10:443
DEBUG:root:Flow ended: 3001:1:a::10:58719 <--> 3001:4:b::10:443
```

See it in Action

Automatic Triggering

```
router2> show route protocol bgp table inet6flow.0

inet6flow.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

3001:1:a::10/128,3001:4:b::10/128/term:1
                *[BGP/170] 00:06:12, localpref 65000, from 3001:2:e10a::10
                    AS path: 65010 I, validation-state: unverified
                    Receive
3001:4:b::10/128,3001:1:a::10/128/term:2
                *[BGP/170] 00:06:12, localpref 65000, from 3001:2:e10a::10
                    AS path: 65010 I, validation-state: unverified
                    Receive
```

```
router1> show route table inet6flow.0

inet6flow.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

3001:1:a::10/128,3001:4:b::10/128/term:1
                *[BGP/170] 00:07:28, localpref 65000
                    AS path: 65010 I, validation-state: unverified
                    >  to 3001:2::2
3001:4:b::10/128,3001:1:a::10/128/term:2
                *[BGP/170] 00:07:28, localpref 65000
                    AS path: 65010 I, validation-state: unverified
                    >  to 3001:2::2
```

See it in Action

FlowSpec Rule Verification

```
router1> show firewall filter __flowspec_default_inet6__

Filter: __flowspec_default_inet6__
Counters:
Name                                              Bytes                Packets
3001:1:a::10/128,3001:4:b::10/128                 15872                    124
3001:4:b::10/128,3001:1:a::10/128                  2000                     25
```

Demos are Hard

During the creation of this demo I found the following issues:

- Juniper vMX:
  - Flowspec exclude interface support (link) not supported
  - Even with MPLS label!
  - This should work in hardware (not in the lab demo)
- Cisco IOS-XR:
  - Flowspec redirect also not supported in ASR-9000v

Now it’s your turn

Hackathon

Feel free to:

- Hack on your own ideas
- Expand on the demos
  - Files available at: **`bit.ly/nanog77-demo`**

- Hackathon helpers are available for help with:
  - Coding, configs, & lab resources
- Reminder to work on your Demo Presentation
  - Take screenshots along the way!