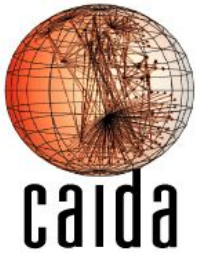
Auto-configuring BGP monitoring and hijack detection tools in real time

Vasileios Kotronis

Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science

GRNOG 9, Athens, Greece, 6 December, 2019



ARTEMIS and its Configuration File

ARTEMIS overview

RIPE NCC
RIPE NETWORK COORDINATION CENTRE
BGPSTREAM

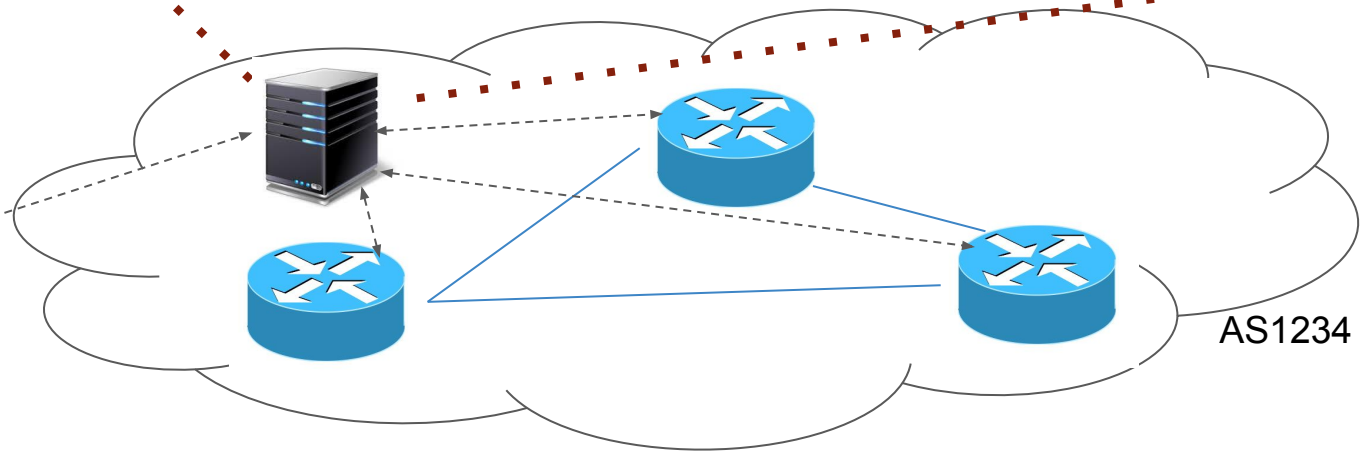
BGP Monitors:
- RIPE RIS
- RouteViews
- BMP
- Local (exaBGP)

Runs as a multi-container app in the NOC

ARTEMIS

MONITORING → DETECTION → MITIGATION

Operator Configuration File



AS1234

RIPE NCC
RIPE NETWORK COORDINATION CENTRE
BGP STREAM

BGP Monitors:
- RIPE RIS
- RouteViews
- BMP
- Local (exaBGP)

"Monitor X saw a BGP update for 10.0.0.0/23 originated by AS4."

"Origin sub-prefix HIJACK by AS4 vs. 10.0.0.0/23."

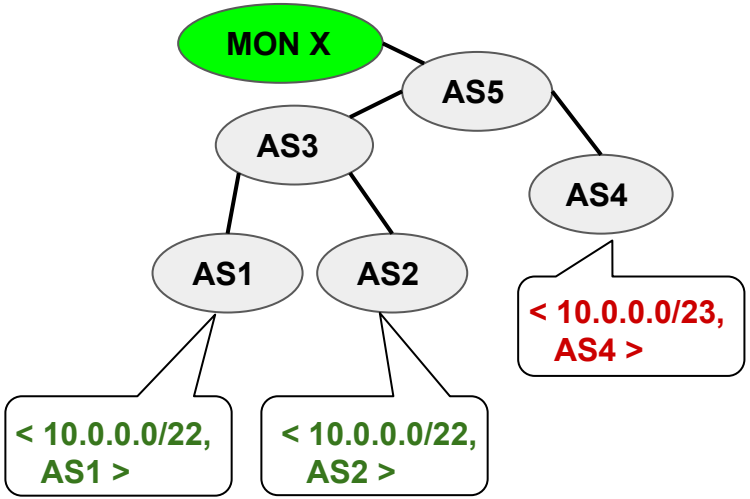
ARTEMIS

MONITORING → DETECTION → MITIGATION

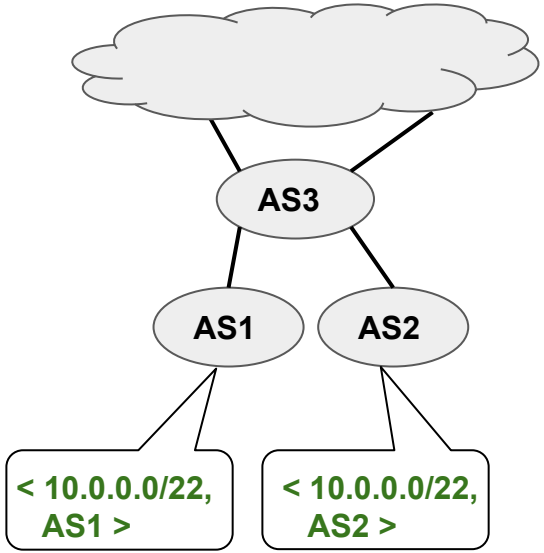
React to hijack!

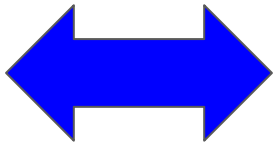
Operator Configuration File

"I own 10.0.0.0/22 and announce it from AS1 and AS2; both have AS3 as upstream."



The configuration file encodes routing policies + enables detection!





```
prefixes:
  my_prefix: &my_prefix
  - 10.0.0.0/22
  ...
monitors:
  riperis: ['']
  bgpstreamlive:
  - routeviews
  - ris
  - caida
  ...
asns:
  my_moas_asns: &my_moas_asns
  - 1
  - 2
  my_upstream_asn: &my_upstream_asn
  - 3
  ...
rules:
  ...
- prefixes:
  - *my_prefix
  origin_asns:
  - *my_moas_asns
  neighbors:
  - *my_upstream_asn
  mitigation: manual
```

Keeping the configuration up-to-date: useful, but hard

- **Why useful?**
  - Contains aggregated AS-level BGP information
  - Important for BGP monitoring and incident detection tools in general [1] [2] [3]
- **Why hard?**
  - The network operator has to manually fill it in and update it for every change in network topology and/or routing policy
  - Not practical for large networks (complex policies, MOAS, rich peerings, etc.)
  - Even if we “extract” this information from public sources (such as [2], [3] do) → not reliable, still needs manual verification

[1] https://github.com/forth-ics-inspire/artemis
[2] https://github.com/nttgin/BGPalerter
[3] https://bgpmon.net/

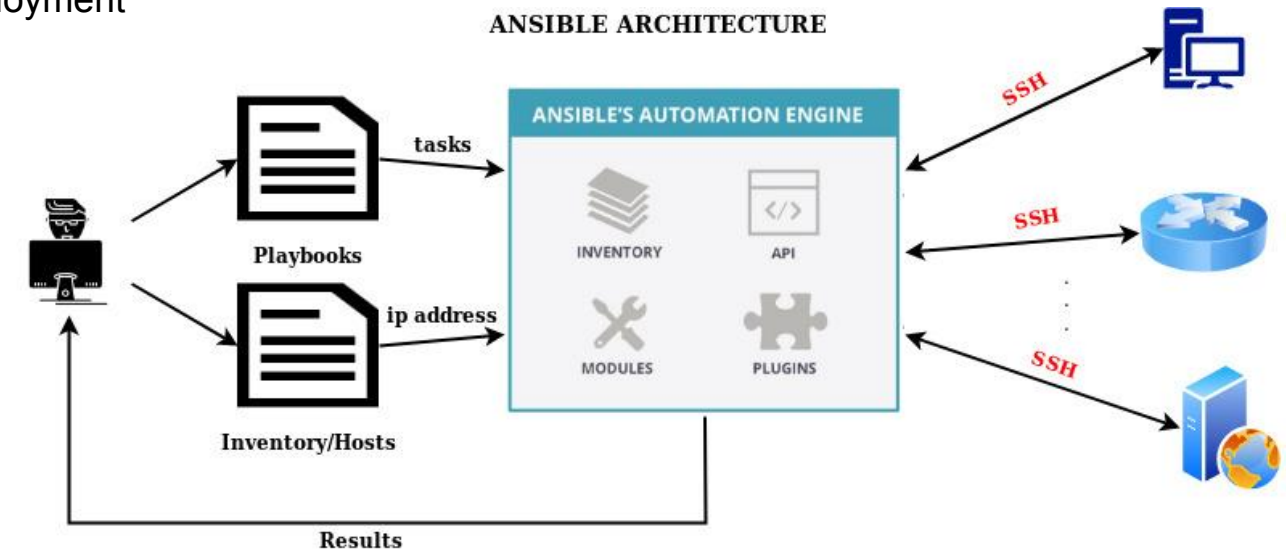
Auto-configuration using Ansible (router-specific, polling-based approach)

https://github.com/georgeepta/artemis-ansible

Ansible in a nutshell

*“A Powerful, **Agentless** ,open source **IT automation tool** for:”*

- Configuration Management
- Application Deployment
- Provisioning



Ansible playbooks

- YAML format
- Contain lists of tasks that tell Ansible what to execute on a particular machine
- Tasks in playbook run sequentially
- Use host's file hierarchy

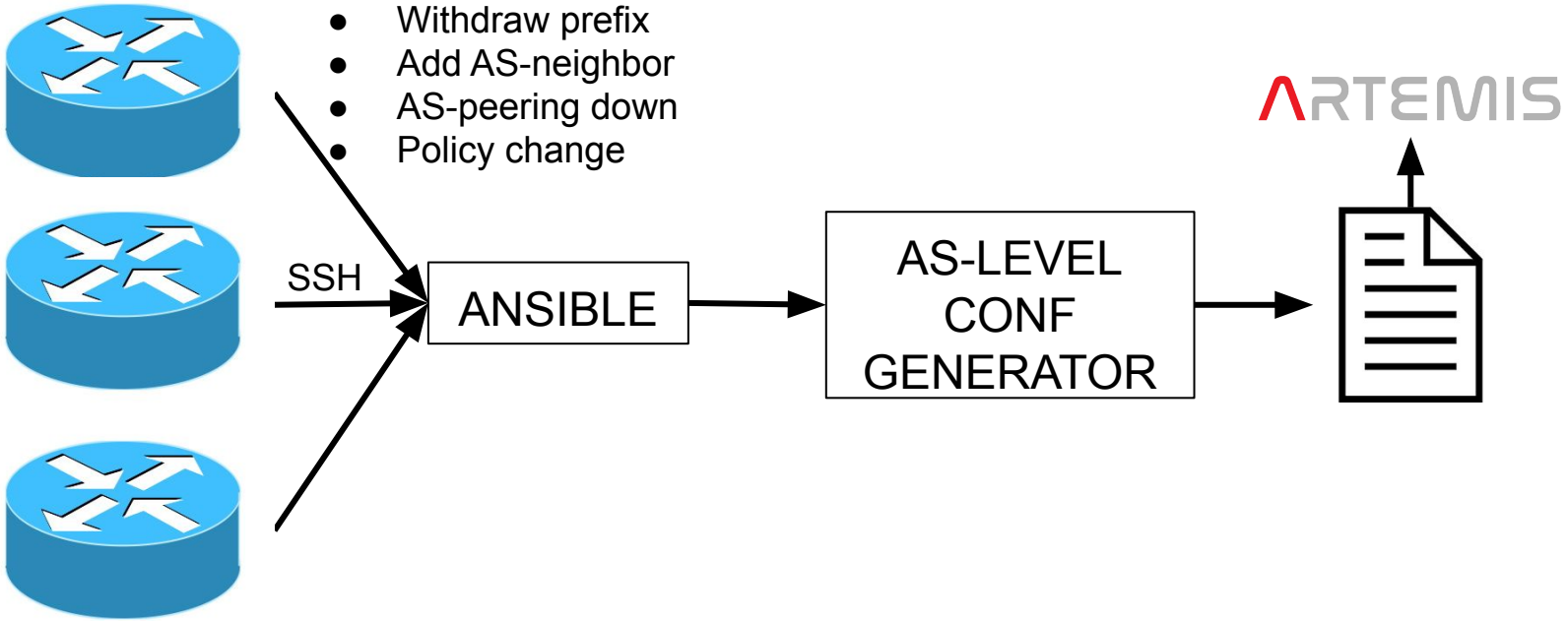
```
- name: EXECUTE TASKS FOR EACH CONNECTED ROUTER
  hosts: all
  connection: network_cli
  gather_facts: false

  tasks:
     - name: Get IOS router configuration
       ios_command:
          commands:
               - show run
       register: output
```

Playbook execution command:

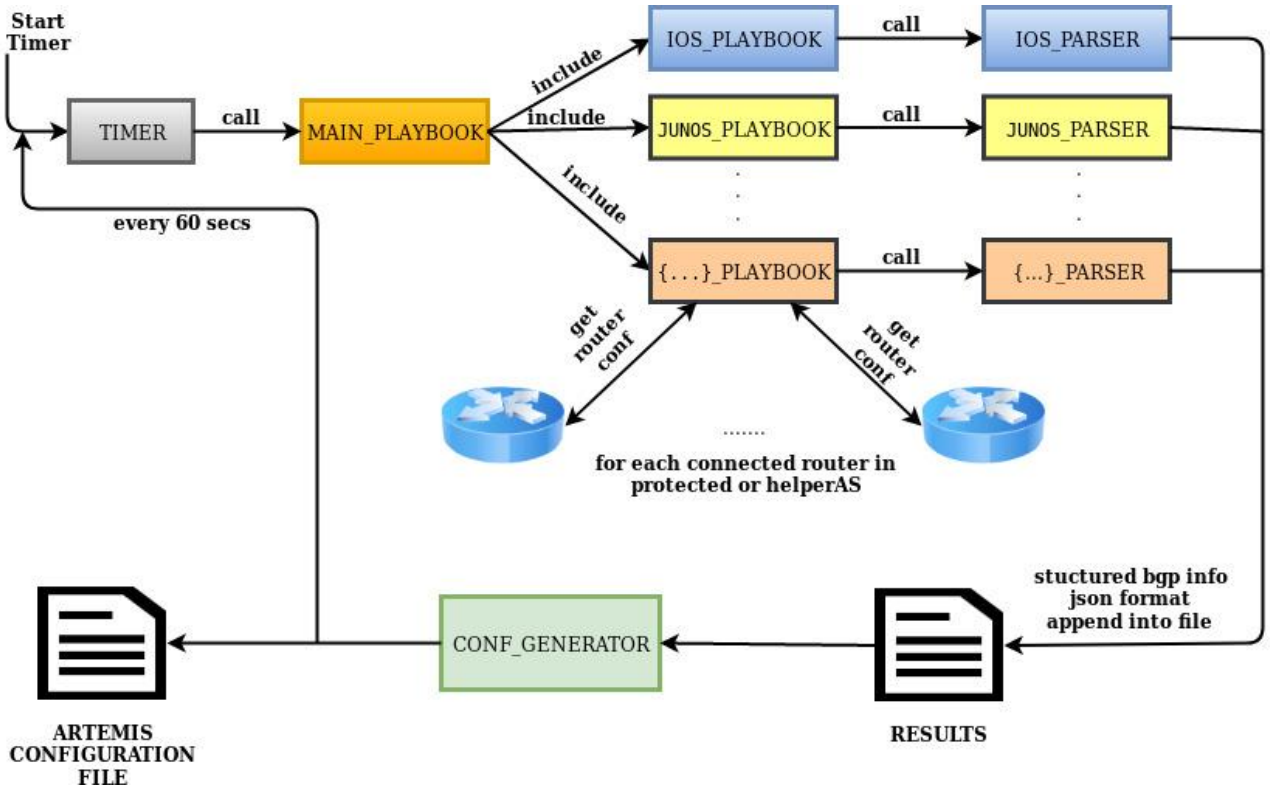
ansible-playbook [options] playbook.yaml [playbook2 ...]

Basic idea of Ansible-based auto-configuration

- Originate new prefix
- Withdraw prefix
- Add AS-neighbor
- AS-peering down
- Policy change



System architecture



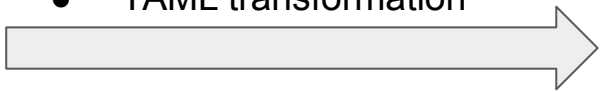
Parsing router configurations

- **Ciscoconfparse** Python library
- Parses Cisco IOS-style configurations
  - Cisco IOS/IOS-XR
  - Arista EOS
  - HP Switches
  - Juniper Networks

Router's conf file

```
router bgp 65001
 bgp router-id 192.168.10.1
 bgp log-neighbor-changes
 network 130.10.0.0 mask 255.255.248.0
 neighbor 2.2.1.2 remote-as 65002
 neighbor 2.2.1.2 route-map PROV-OUT out
```

- AS-level aggregation
- Conf primitive transforms
- YAML transformation



ARTEMIS conf file

```
prefixes:
   prefix_1: &prefix_1
   - 130.10.0.0/21
...
asns:
  AS_65001: &AS_65001
  - 65001
  AS_65002: &AS_65002
  - 65002
...
rules:
- prefixes:
  - *prefix_1
  origin_asns:
  - *AS_65001
  neighbors:
  - *AS_65002
  mitigation:
  -   /root/mitigation_trigger.py
…
```

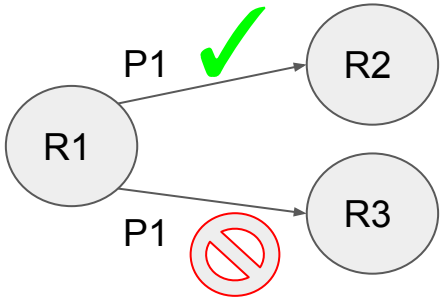
Supported configuration primitives for Cisco IOS

- ✓ **BGP router-id**
- ✓ **BGP announced prefixes**
- ✓ **BGP origin asn**
- ✓ **BGP neighbor asns**
- ✓ **BGP peer-groups**

- ✓ **Router interfaces**
- ✓ **BGP route-maps**
- ✓ **Prefix lists**
- ✓ **Access control lists (numbered + standard)**

}

E.g., for selective prefix announcement



Challenges with Ansible-based approach

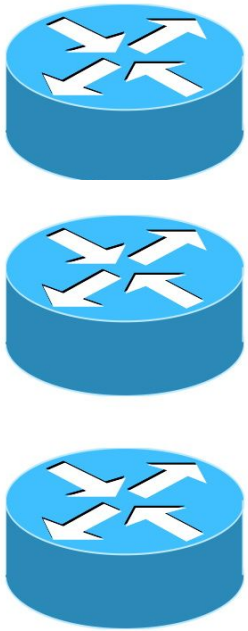
- SSH access required
  - Tricky to give to an application, needs proper credential management
  - Accountability w.r.t. any actions taken on the router-level
- Agentless: not asynchronous, requires polling interval
  - During the polling interval, non-learned changes may trigger hijack alerts!
  - Change quicker than configuration update, “pseudo-real-time”
- Need different parsers for different router types
  - Currently CISCO IOS is supported (has been tested)
- Can overwrite manually induced conf changes (in current implementation)

Auto-configuration using local BGP feeds (passive async approach)

https://github.com/forth-ics-inspire/artemis

Basic idea

- Originate new prefix
- Withdraw prefix
- Add AS-neighbor
- AS-peering down
- Policy change



BGP

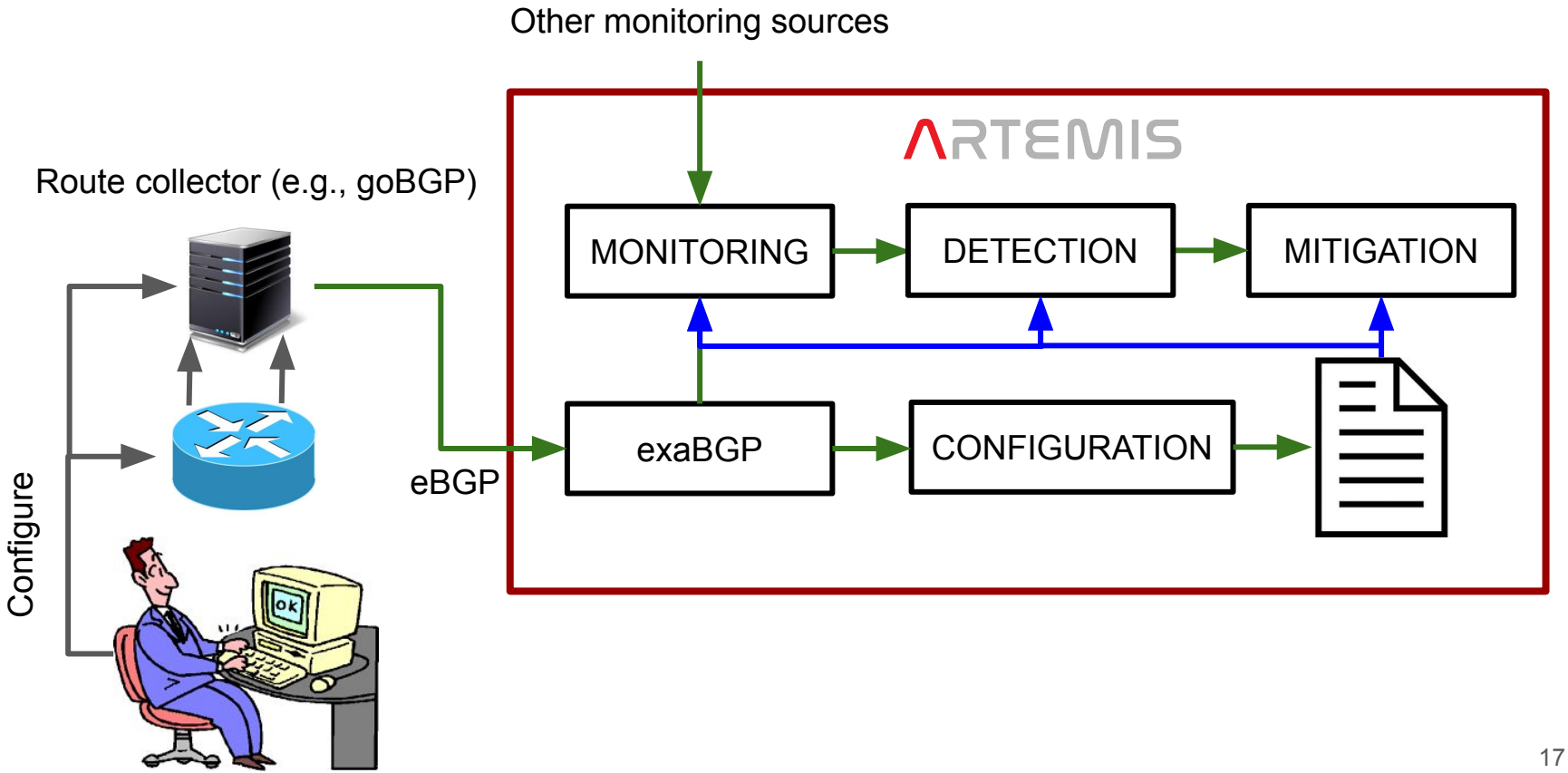
ROUTE COLLECTOR

BGP API (exaBGP)

AS-LEVEL CONF GENERATOR

ARTEMIS

System architecture



Min. requirements: route maps on RC’s side

```
...
router bgp 1
    bgp router-id 1.1.1.1

    ! announced networks
    network 192.168.1.0/24
    ...
    ! inbound/outbound policy
    ...
    neighbor MONITOR peer-group
    neighbor MONITOR route-map RM-MONITOR-IN in
    neighbor MONITOR next-hop-self
    ...
    ! monitors
    neighbor 192.168.10.2 remote-as <MONITOR_AS>
    neighbor 192.168.10.2 peer-group MONITOR
    neighbor 192.168.10.2 ebgp-multihop 2
    neighbor 192.168.10.2 description Local Exabgp RC
...
! Route map for monitors.
! Block all incoming advertisements
route-map RM-MONITOR-IN deny 10
...
```

Min. requirements: exaBGP API configuration

```
group r1 {
    router-id <PUBLIC_IP>;

    process message-logger {
        encoder json;
        receive {
            parsed;
            update;
            neighbor-changes;
        }
        run /usr/lib/python2.7.14/bin/python /home/server.py;
    }

    neighbor <NEIGHBOR_IP> {
        local-address <LOCAL_LAN_IP>;
        local-as <LOCAL_ASN>;
        peer-as <PEER_ASN>;
    }
}
```

Min. requirements: ARTEMIS configuration

```
...
monitors:
  ...
  exabgp:
    - ip: exabgp
      port: 5000
      autoconf: "true"
...


# run with:
docker-compose -f docker-compose.yaml -f docker-compose.exabgp.yaml up -d
```

Auto prefix and origin AS learning: originate

```
prefixes: {}
monitors:
  riperis: ['']
  bgpstreamlive:
    - routeviews
    - ris
    - caida
  exabgp:
    - ip: exabgp
      port: 5000
      autoconf: "true"
asns: {}
rules: []
```

(configuration before)

Origination of 192.168.1.0/24 from AS1

```
prefixes:
  AUTOCONF_P_192_168_1_0_24: &AUTOCONF_P_192_168_1_0_24
  - 192.168.1.0/24
monitors:
  riperis: ['']
  bgpstreamlive:
  - routeviews
  - ris
  - caida
  exabgp:
  - ip: exabgp
    port: 5000
    autoconf: "true"
asns:
  AUTOCONF_AS_1: &AUTOCONF_AS_1
  - 1
rules:
- prefixes:
  - *AUTOCONF_P_192_168_1_0_24
  origin_asns:
  - *AUTOCONF_AS_1
  mitigation: manual
```

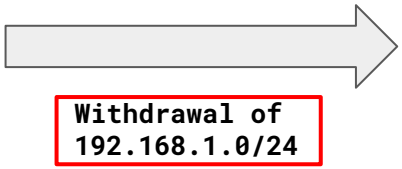
This enables detection of fake origin + sub-prefix hijacks!

(configuration after)

Auto prefix and origin AS learning: withdraw

```
prefixes:
  AUTOCONF_P_192_168_1_0_24: &AUTOCONF_P_192_168_1_0_24
  - 192.168.1.0/24
monitors:
  riperis: ['']
  bgpstreamlive:
  - routeviews
  - ris
  - caida
  exabgp:
  - ip: exabgp
    port: 5000
    autoconf: "true"
asns:
  AUTOCONF_AS_1: &AUTOCONF_AS_1
  - 1
rules:
- prefixes:
  - *AUTOCONF_P_192_168_1_0_24
  origin_asns:
  - *AUTOCONF_AS_1
  mitigation: manual
```

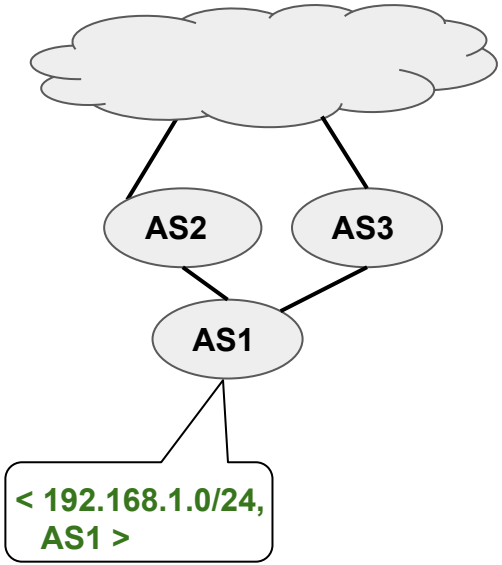
(configuration before)

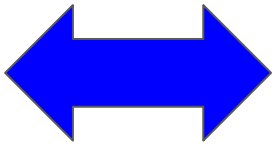



```
prefixes: {}
monitors:
  riperis: ['']
  bgpstreamlive:
    - routeviews
    - ris
    - caida
  exabgp:
    - ip: exabgp
      port: 5000
      autoconf: "true"
asns: {}
rules: []
```

(configuration after)

Auto 1st-hop neighbor learning: getting neighbor info





Annotate prefix origination with communities
[1:2] ⇔ AS1 announces prefix to AS2
[1:3] ⇔ AS1 announces prefix to AS3

```
...
route-map RM-MONITOR-OUT permit 10
    match community selforig
    set community 1:2 additive
    on-match next
route-map RM-MONITOR-OUT permit 20
    match community selforig
    set community 1:3 additive
...
```

Auto 1st-hop neighbor learning: originate

```
prefixes: {}
monitors:
  riperis: ['']
  bgpstreamlive:
    - routeviews
    - ris
    - caida
  exabgp:
    - ip: exabgp
      port: 5000
      autoconf: "true"
asns: {}
rules: []
```

(configuration before)

Origination of 192.168.1.0/24 from AS1 with communities [1:2, 1:3]

This enables detection of fake origin + sub-prefix + fake neighbor hijacks!

```
prefixes:
  AUTOCONF_P_192_168_1_0_24: &AUTOCONF_P_192_168_1_0_24
  - 192.168.1.0/24
monitors:
  riperis: ['']
  ...
  exabgp:
  - ip: exabgp
    port: 5000
    autoconf: "true"
asns:
  AUTOCONF_AS_1: &AUTOCONF_AS_1 1
  AUTOCONF_AS_2: &AUTOCONF_AS_2 2
  AUTOCONF_AS_3: &AUTOCONF_AS_3 3
rules:
- prefixes:
  - *AUTOCONF_P_192_168_1_0_24
  origin_asns:
  - *AUTOCONF_AS_1
  neighbors:
  - *AUTOCONF_AS_2
  - *AUTOCONF_AS_3
  mitigation: manual
```

(configuration after)

Challenges

- Asynchronous (real-time), but needs pre-configuration on netops' side
  - Setup eBGP session between tool (via exaBGP) and RC (or router)
  - Configure route maps properly
  - Route map integration into production configs might be complex

- RCs should -ideally- export all visible paths, instead of the best one
  - BGP additional paths
  - `adj-RIB-in` via BMP
  - alternative: several eBGP sessions with routers

- Scalability when 100s of AS-peerings
  - E.g., IXP setup, information hidden behind IXP RS
  - Large transit networks with several customers

Status and next steps

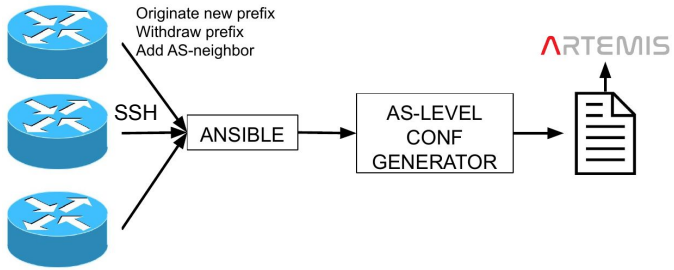
- Experimental Ansible prototype available (`artemis-ansible`) [1]
  - Working with ARTEMIS devs to integrate this in [2] as another microservice
- Local feed-based autoconfiguration available in latest ARTEMIS [2]
  - Release: 1.4.0
- Next steps
  - Get feedback
  - Quantify trade-offs
  - Revise approaches where needed

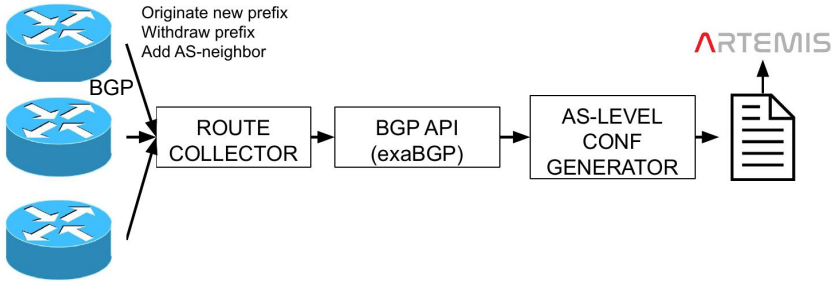
[1] https://github.com/georgeepta/artemis-ansible
[2] https://github.com/forth-ics-inspire/artemis

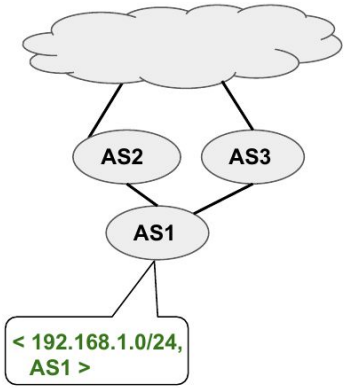
Feedback needed

- Is the route map manipulation to convey neighbor info too complex?
  - Are communities the “best” way to convey such information between your routers and ARTEMIS-like tools?
- How can we scale this up for IXP peerings?
  - Public or local feed from IXP RS?
- How about learning neighbors from reverse AS-paths?
  - From non-local origins, other prefixes
  - What about policy asymmetries?

Thank you! Questions?








```
prefixes:
  AUTOCONF_P_192_168_1_0_24: &AUTOCONF_P_192_168_1_0_24
  - 192.168.1.0/24
monitors:
  riperis: ['']
  ...
  exabgp:
  - ip: exabgp
    port: 5000
    autoconf: "true"
asns:
  AUTOCONF_AS_1: &AUTOCONF_AS_1 1
  AUTOCONF_AS_2: &AUTOCONF_AS_2 2
  AUTOCONF_AS_3: &AUTOCONF_AS_3 3
rules:
- prefixes:
  - *AUTOCONF_P_192_168_1_0_24
  origin_asns:
  - *AUTOCONF_AS_1
  neighbors:
  - *AUTOCONF_AS_2
  - *AUTOCONF_AS_3
  mitigation: manual
```

Useful links

- Official Github repository + wiki: https://github.com/forth-ics-inspire/artemis
- Discord channel(s): https://discord.gg/8UerJvh
- My email: vkotronis[at]ics[dot]forth[dot]gr

BACKUP

Hijacks: dimensions

BACKUP

Type	Examples	ARTEMIS-Supported
Prefix	Sub(S)-/Exact(E)-prefix, squatting (Q)	S, E, Q
AS-Path	Type-0/1/… (depending on hijacker AS-hop)	0, 1
Data plane	Blackholing, Imposture, MitM	- (control-plane tool)
Policy	No-export route leak (L), ...	L (based on AS-path length)

Example 1: Invalid origin, advertising a configured prefix: **E|0|-|-**
Example 2: Valid origin, fake neighbor, leaking a sub-prefix of a configured prefix: **S|1|-|L**

ARTEMIS configuration file as ground truth info

- Define prefix, ASN, monitor groups

- Declare ARTEMIS rules:
  - “My ASes ASX and ASY originate prefix P”
  - “And they advertise it to ASZ”
  - “When a hijack occurs → mitigate manually”

BACKUP

Sample Rule	Sample Incoming BGP update	Hijack
prefixes: - *my_prefix origin_asns: - *my_origin neighbors: - *my_neighbor mitigation: manual	[..., <subprefix_of_my_prefix>]	S|-|-|-
	[..., <not_my_origin>, <my_prefix>]	E|0|-|-
	[..., <not_my_neighbor>, <my_origin>, <my_prefix>]	E|1|-|-
prefixes: - *my_prefix mitigation: manual	[..., <my_prefix>]	Q|0|-|-

Auto 1st-hop neighbor learning: getting neighbor info

BACKUP

```
...
router bgp 1
    bgp router-id 1.1.1.1

    ! announced networks
    network 192.168.1.0/24 route-map SET-SELF-COMM
    ...
    ! inbound/outbound policy
    ...
    neighbor MONITOR peer-group
    neighbor MONITOR route-map RM-MONITOR-IN in
    neighbor MONITOR route-map RM-MONITOR-OUT out
    neighbor MONITOR next-hop-self
    ...
    ! monitors
    neighbor 192.168.10.2 remote-as <MONITOR_AS>
    neighbor 192.168.10.2 peer-group MONITOR
    neighbor 192.168.10.2 ebgp-multihop 2
    neighbor 192.168.10.2 description Local Exabgp RC
...
```

```
! Route map for locally originated networks
route-map SET-SELF-COMM permit 10
    set community 1:1 additive
...
! Route map for monitors.
! Block all incoming advertisements
route-map RM-MONITOR-IN deny 10

! Here declare also the neighbors
! to whom these prefixes are advertised
route-map RM-MONITOR-OUT permit 10
    match community selforig
    set community 1:2 additive
    on-match next
route-map RM-MONITOR-OUT permit 20
    match community selforig
    set community 1:3 additive
    on-match next
route-map RM-MONITOR-OUT permit 30

! community list matching self-originated route entries
ip community-list standard selforig permit 1:1
...
```

Ansible-based auto-configuration mechanism

BACKUP

- Communicates directly with routers via SSH
- Every **polling interval** it receives feed from directly connected routers
- Updates ARTEMIS configuration file **only if one or more changes occurred w.r.t. network topology or routing-policy on the AS-level, e.g.,:**
  - Router/link is down/up (AS-peering down/up)
  - New BGP prefix announcement/withdrawal
  - Selective BGP announcements (policy change)