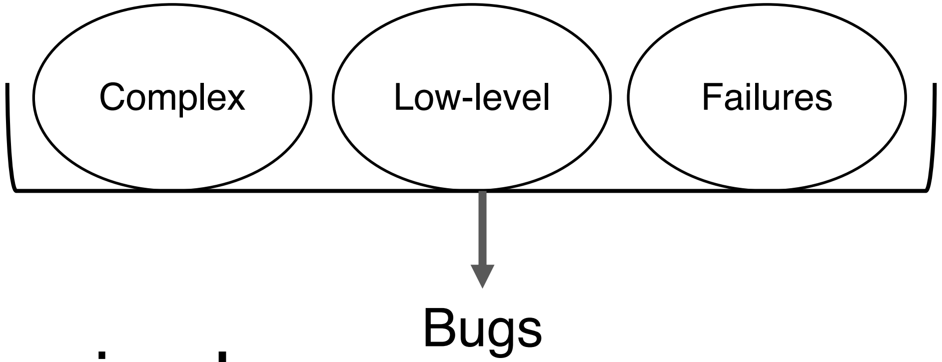
Survey in Network Config Analytics

siiba

General background

Debugging a network is hard !

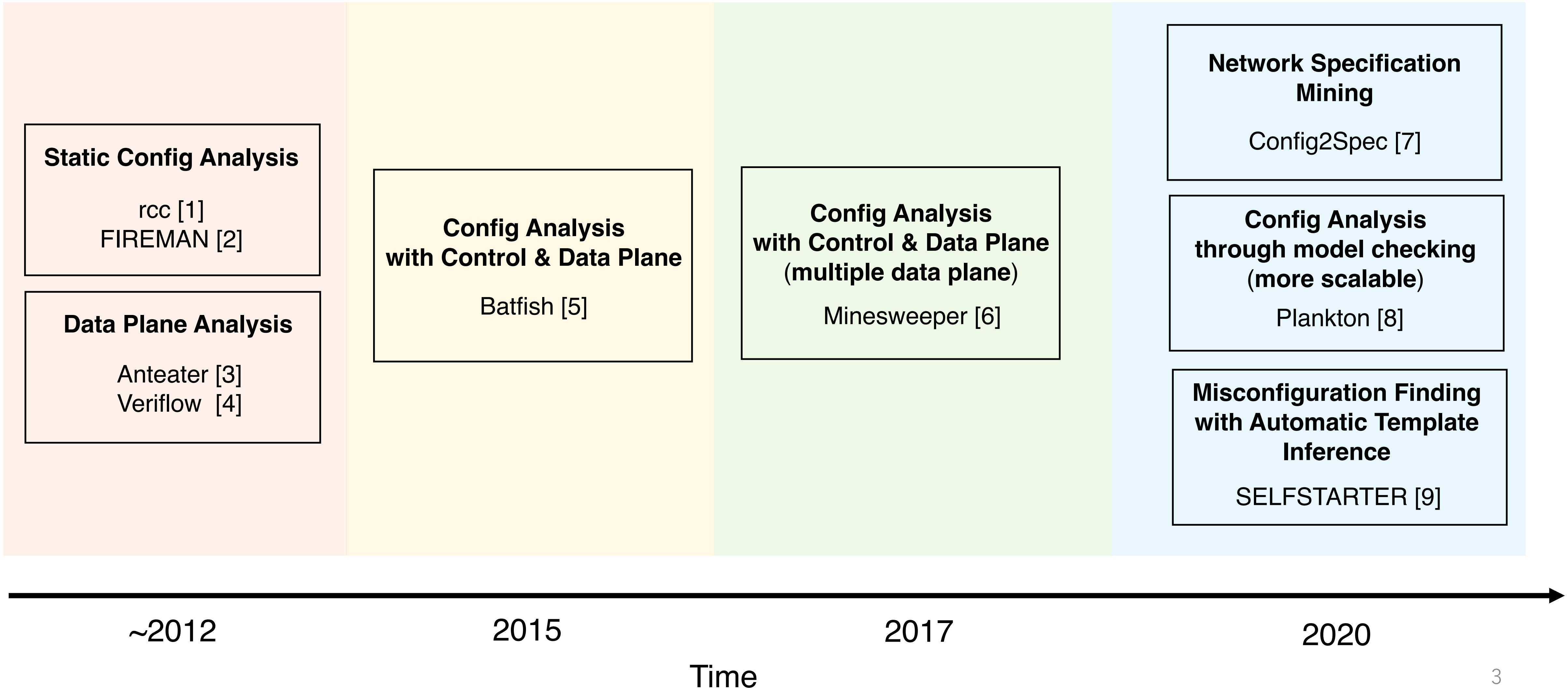
- Complexity of the multiple protocol interactions
- Low level configuration
- Failures



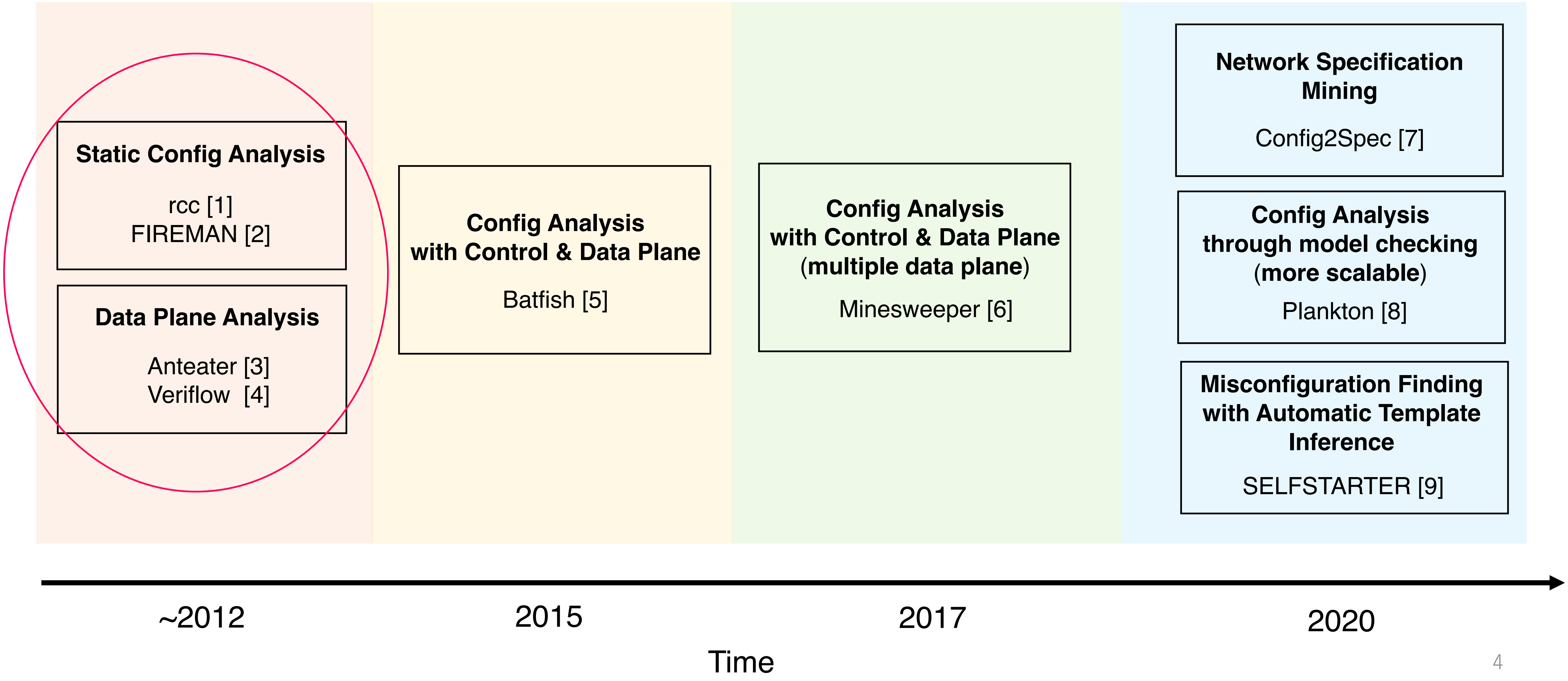
Misconfiguration of the network is so expensive !

- cost hundreds of thousands of dollars for every hour of downtime

Historical progress



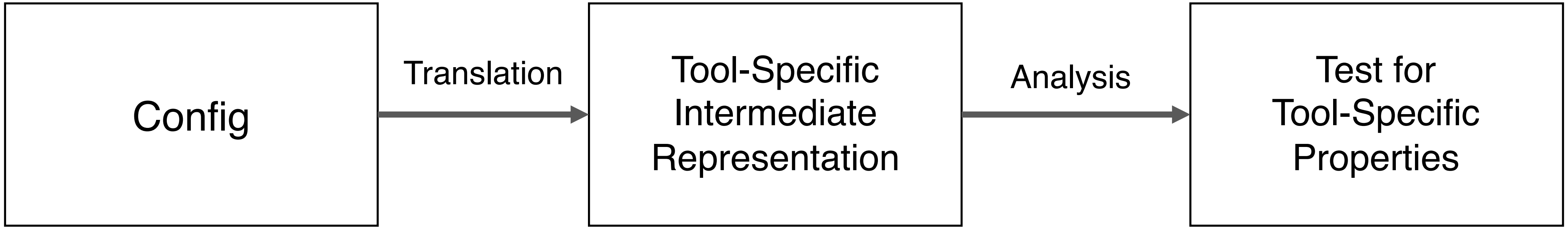
Historical progress



Static Configuration Analysis (~2012s)

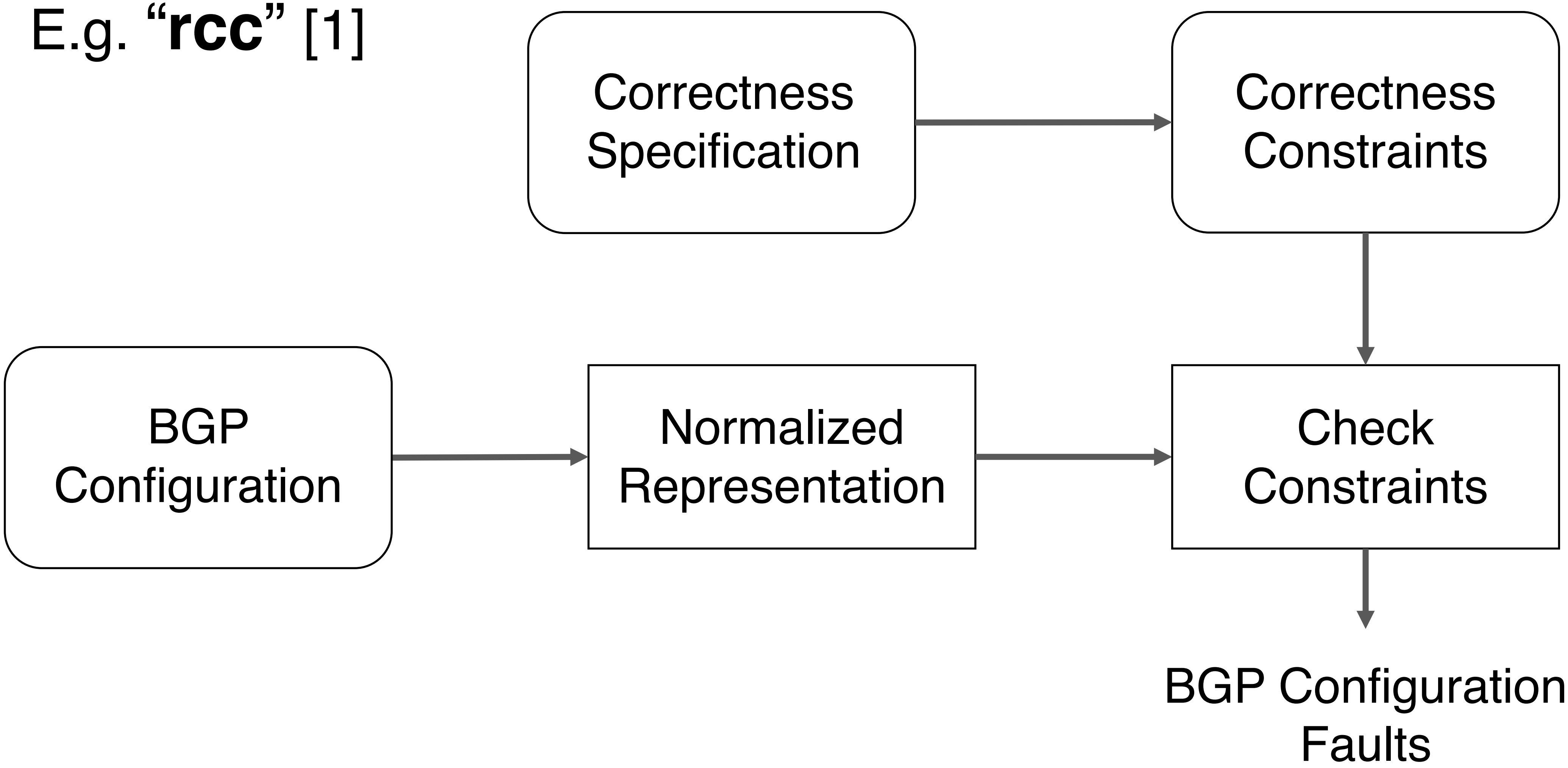
Directly analyzing network configuration files

+ Detecting configuration errors proactively !
+ Can do ”what-if” !



Static Configuration Analysis (~2012)

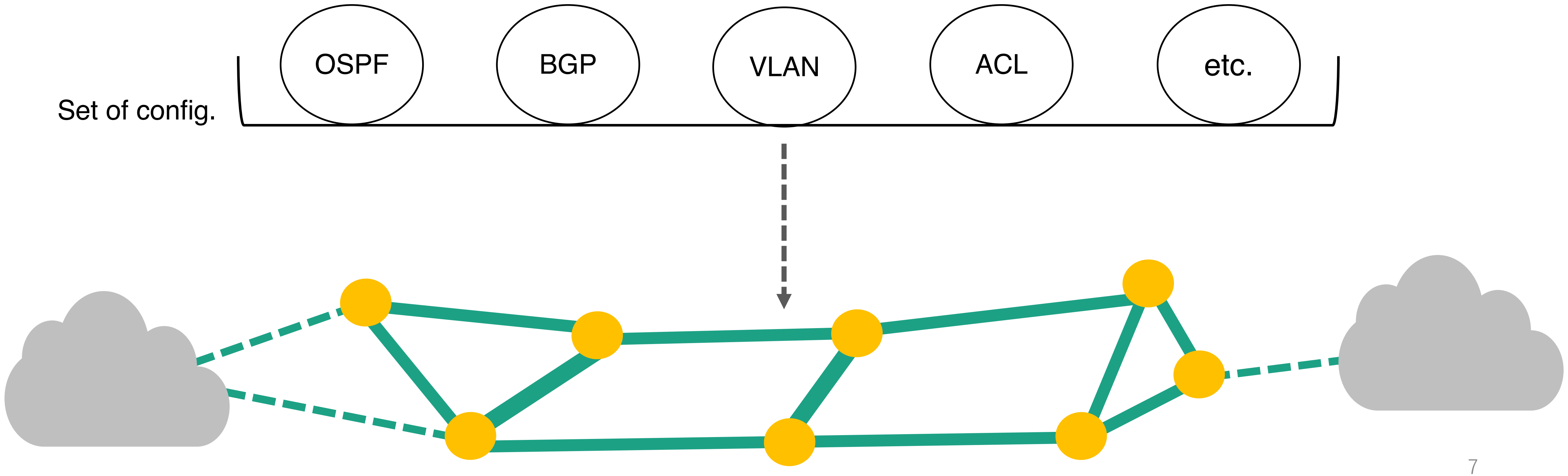
E.g. “**rcc**” [1]



Static Configuration Analysis (~2012)

– Customized model for specific aspects of config. or properties

→ Unable to handle interactions of the many protocols

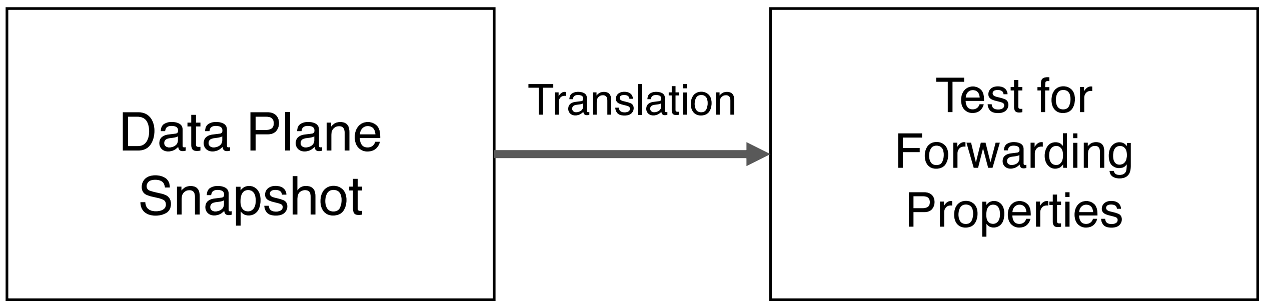


Data Plane Analysis (~2012)

Directly analyzing network data plane (Anteater[3], Veriflow[4])

+ Detecting configuration error precisely !
+ Good structure for encoding in various logic !

  → scalable checking with constraint solvers

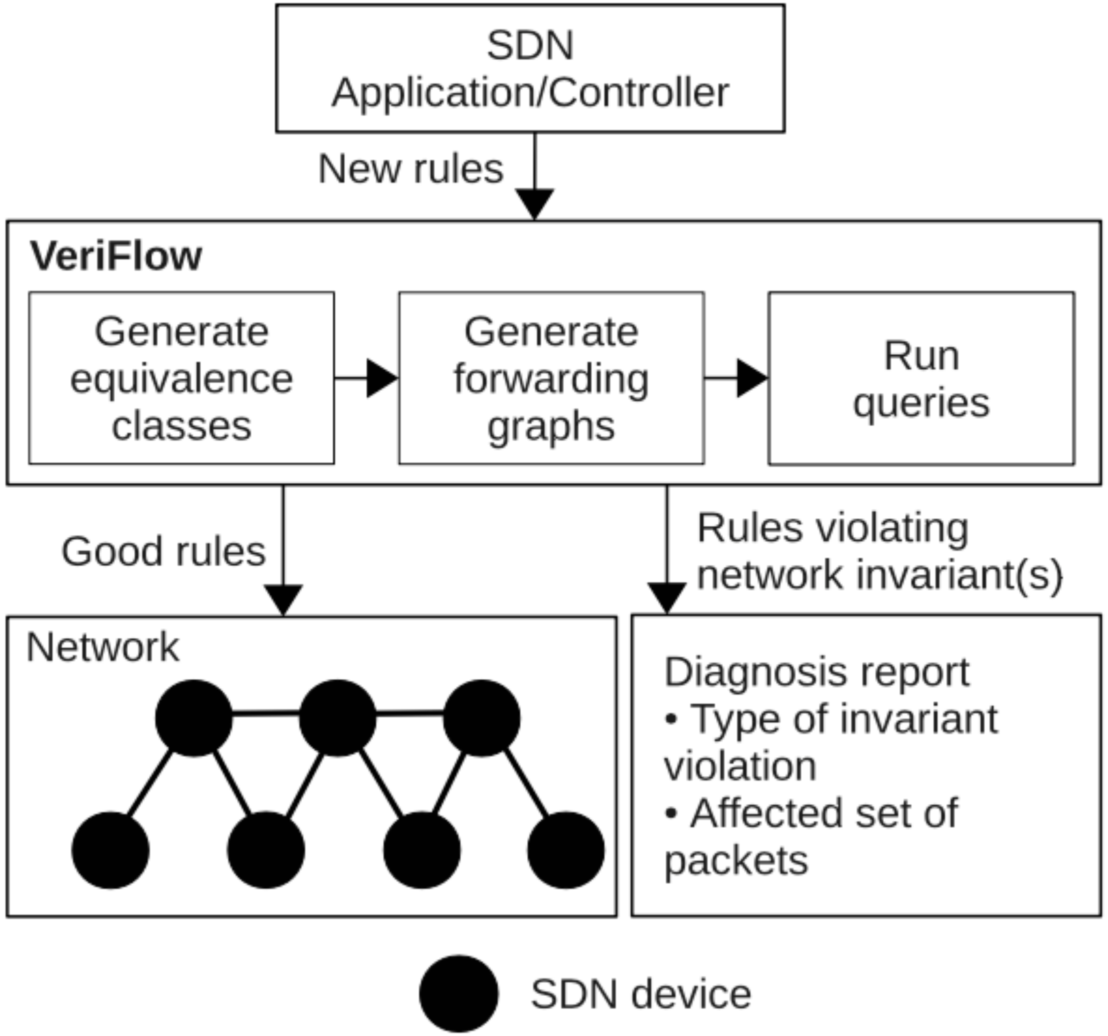


Data Plane Analysis (~2012)

E.g. "**Veriflow**" [4]

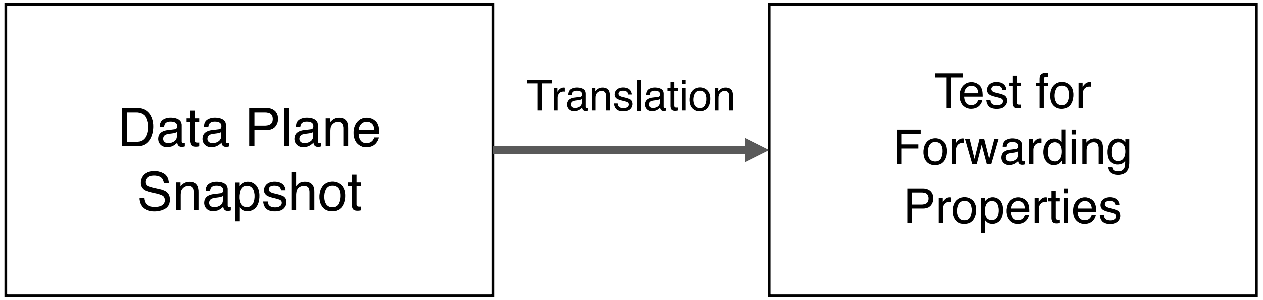
A proxy between

a SDN controller and device



Data Plane Analysis (~2012s)

- Can not prevent the errors proactively
- Need to localize the responsible snippets of configuration

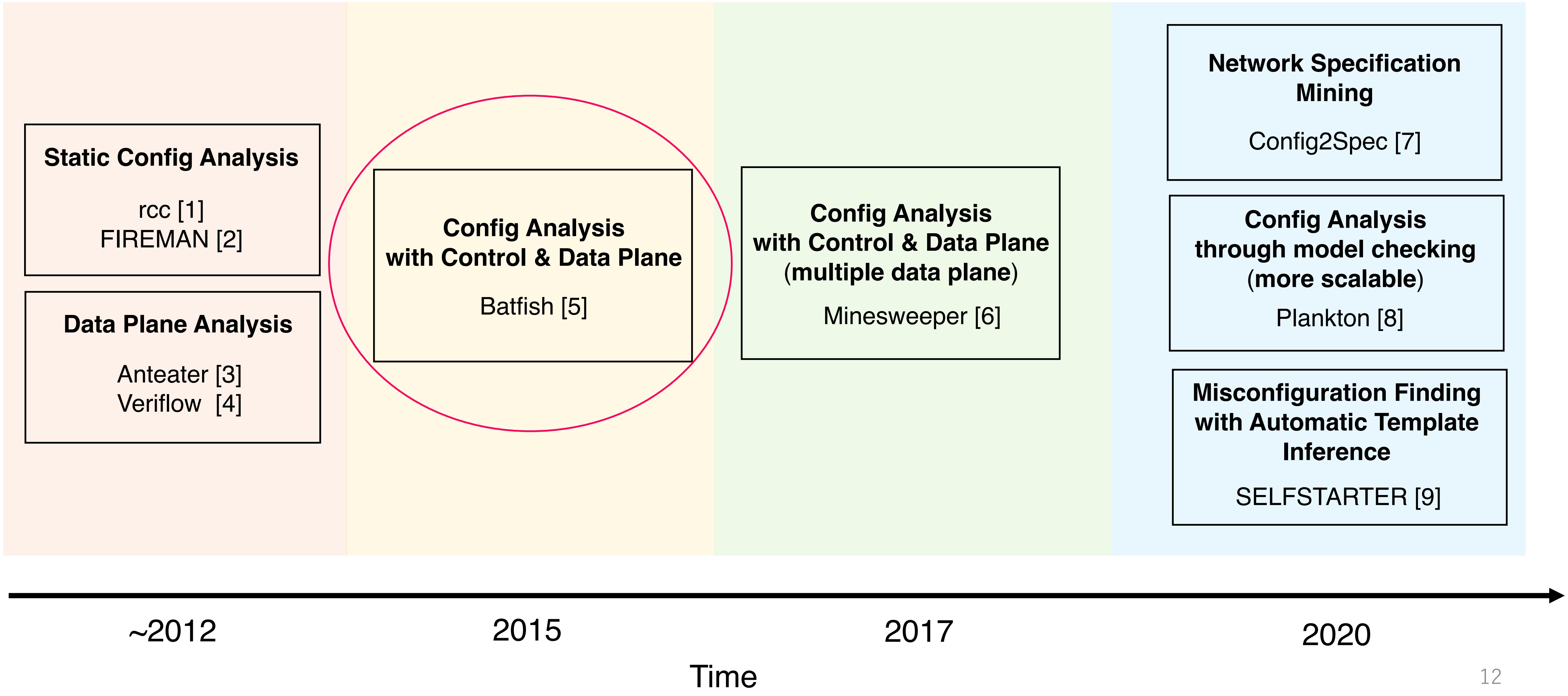


Summary (~2012)

- The approaches have both advantage and disadvantage

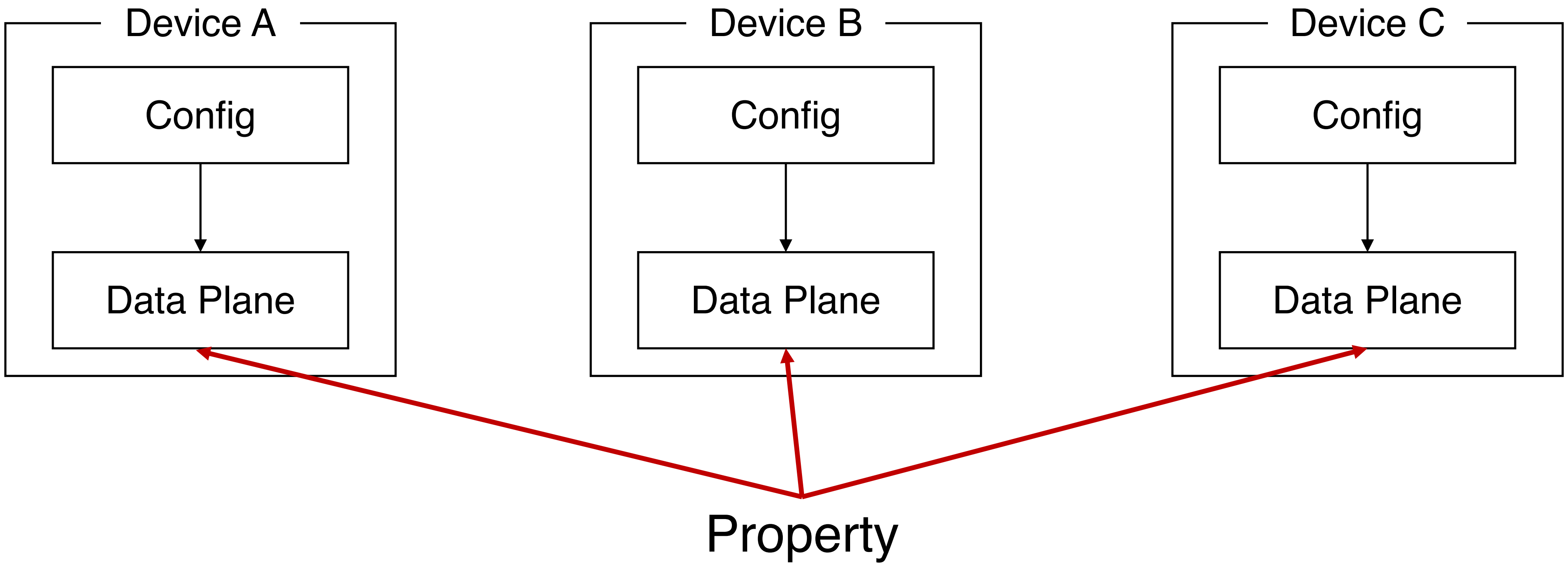
	Proactive error analysis	**Interaction of multiple protocols**
Static configuration analysis	○	✕
Data plane analysis	✕	○

Historical progress



Background and Motivation

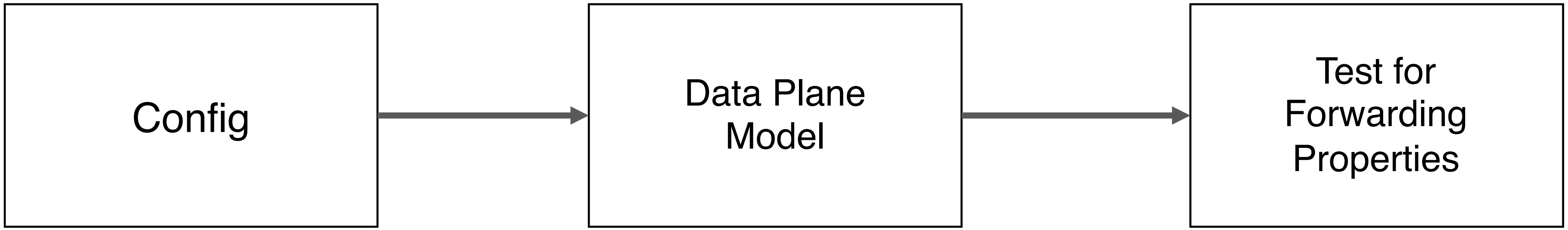
- Can we achieve proactive error analysis and checking of any forwarding property simultaneously?



Approach

Combining the strength of the two prior approaches

1. Deriving a data plane model given a config. and environment
2. Checking the correctness properties with data-plane analysis



Approach

- Combining the strength of the two approaches

	Proactive error finding	**Interaction of multiple protocols**
Static configuration analysis (~2013)	○	✕
Data plane analysis (~2013)	✕	○
Batfish	○	○

Approach

Stage 1: Configuration file + Topology → Control Plane Model

Stage 2: CP. Model + Environment → Data Plane Model

Stage 3: DP. Model + Safety Property → Counterexample

Stage 4: DP. Model + User Input + Counterexample
→ Misconfiguration line

Stage 1: Generating control plane model

- Transforming the configuration with the network topology into a control plane model

- The model is defined in LogiQL

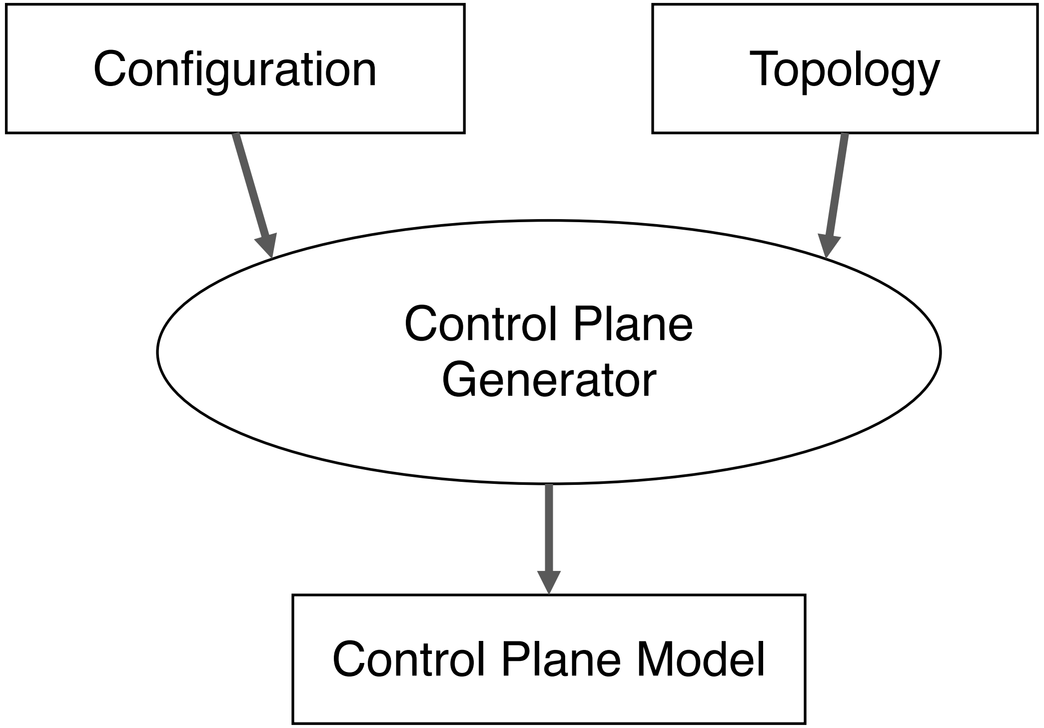
BestOspfRoute(node, network, nextHop, nhIp, cost)

Rule

← OspfRoute(node, network, nextHop, nhIp, cost),
MinOspfCost[node,network] = cost

Body

One of the example: A best route of OSPF

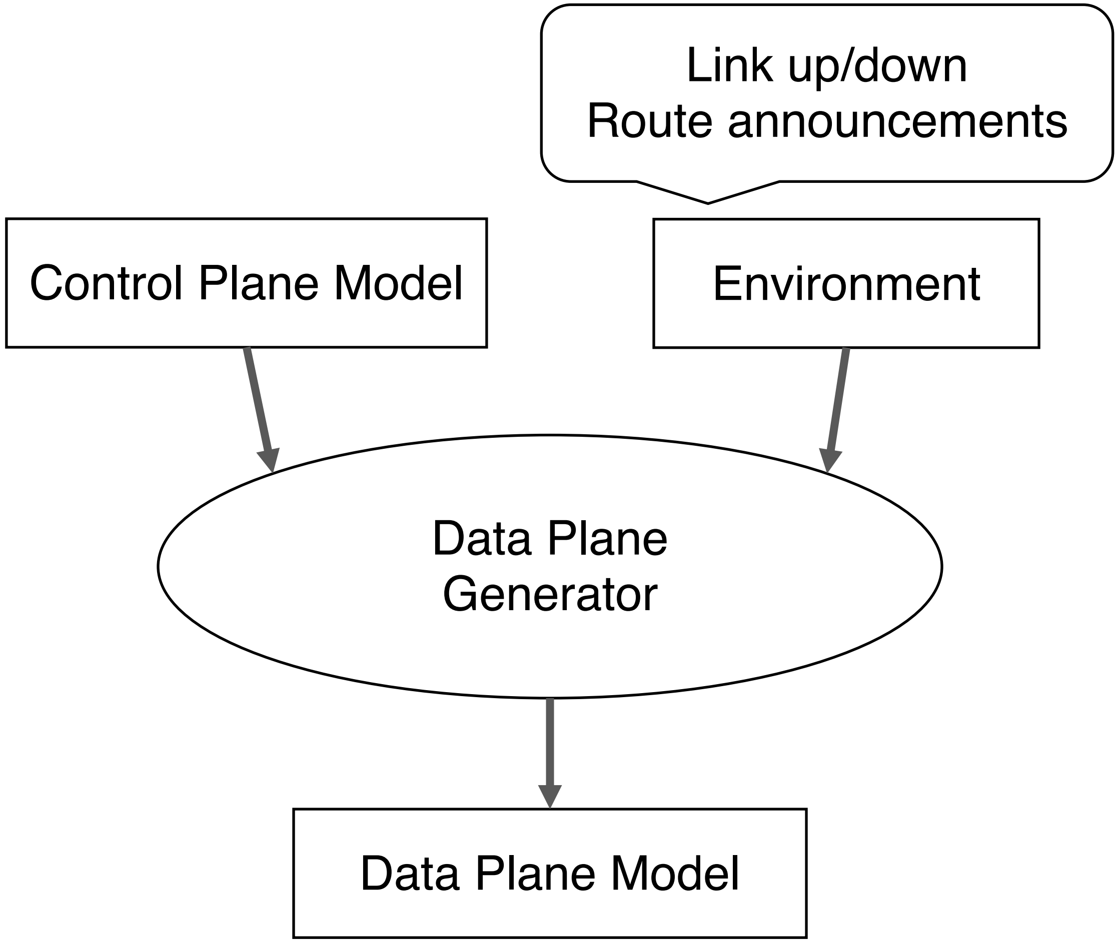


Stage 2: Generating data plane model

- Transforming the control plane model and environment into a data plane model
- The model includes the forwarding behavior as logical facts

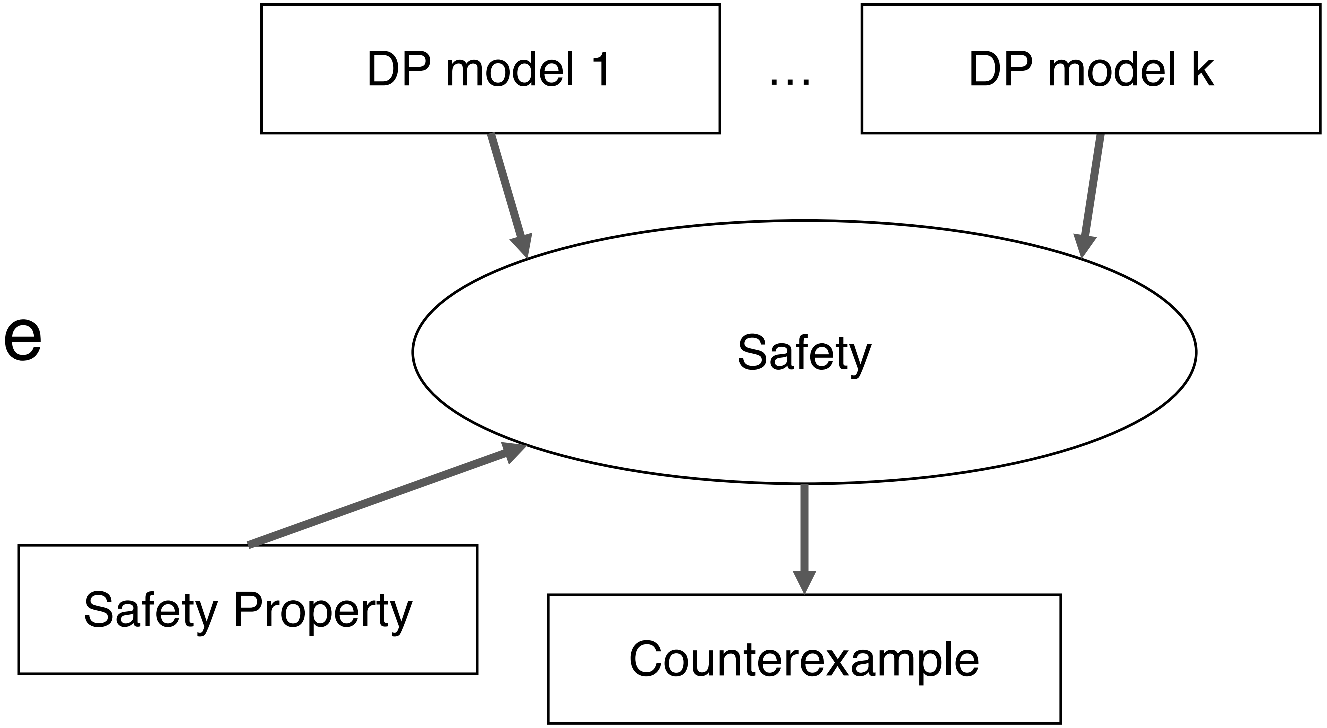
Drop(node, flow)
Forward(node, flow, neighbor)

One of the example: Drop and Forward



Stage 3: Checking safety properties

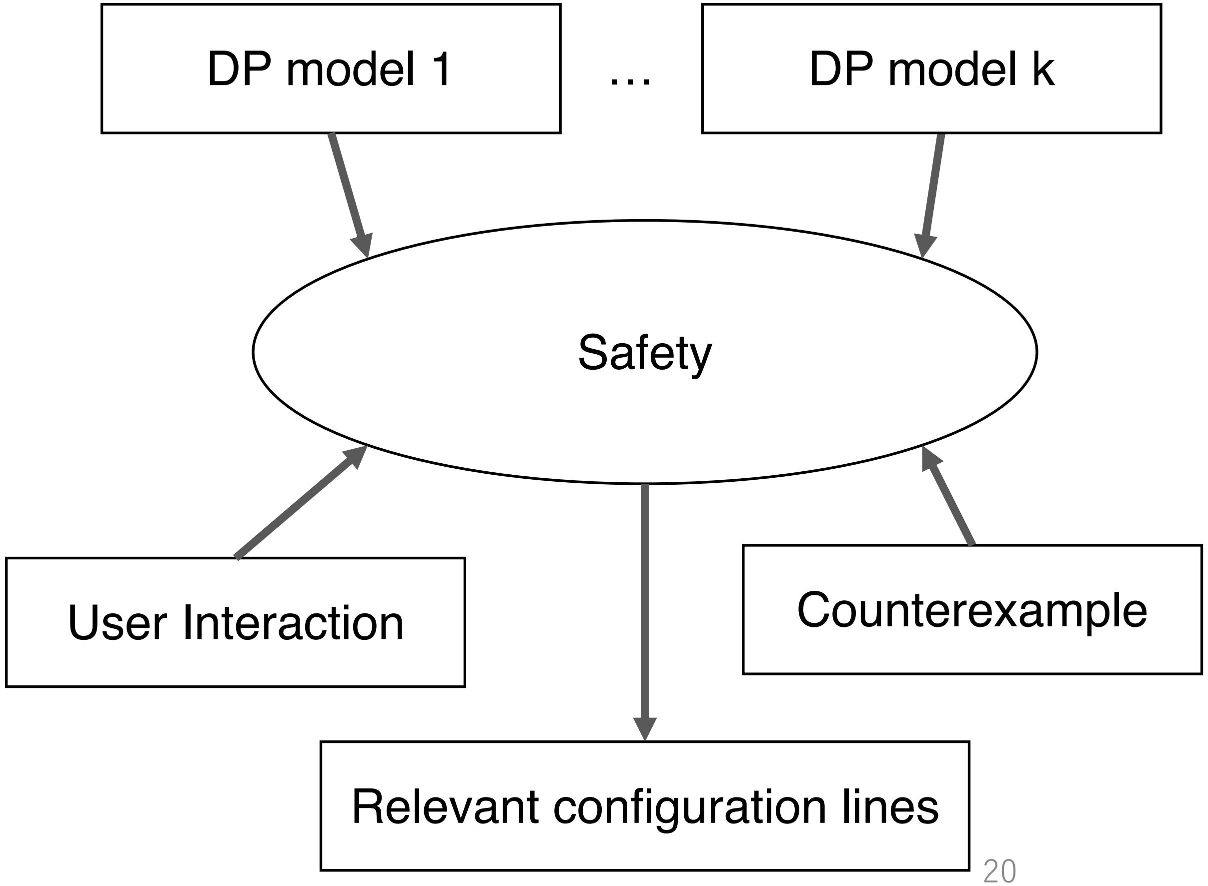
- Translating the data-plane relations and the correctness property to the language of the Z3 constraint solver



Stage 4: Finding error-relevant configuration lines

- Searching the facts in the following order

Counterexample → DP model
→ CP model. → Configuration lines



Evaluation

Net1: 21 routers, 52 AS, Net2: 17 routers, 1AS

Computation time

・Net 1: 238 min, Net 2: 37min

Error detection

		Total violations	Undesired behaviors	Fixed violations
Net1	Multipath	32 (4)	32 (4)	21 (3)
	Failure	16 (7)	3 (2)	0 (0)
	Destination	55 (6)	55 (6)	1 (1)
Net2	Multipath	11 (3)	11 (3)	11 (3)
	Failure	77(26)	18(7)	0(0)

Summary

- Batfish combines the benefits of prior works to achieve both ....
  proactive analysis & **any forwarding property checking !**

	Proactive error finding	**Interaction of multiple protocols**
Static configuration analysis (~2013)	○	✕
Data plane analysis (~2013)	✕	○
Batfish	○	○

Historical progress

Static Config Analysis

rcc [1]
FIREMAN [2]

Data Plane Analysis

Anteater [3]
Veriflow [4]

Config Analysis with Control & Data Plane

Batfish [5]

Config Analysis with Control & Data Plane (multiple data plane)

Minesweeper [6]

Network Specification Mining

Config2Spec [7]

Config Analysis through model checking (more scalable)

Plankton [8]

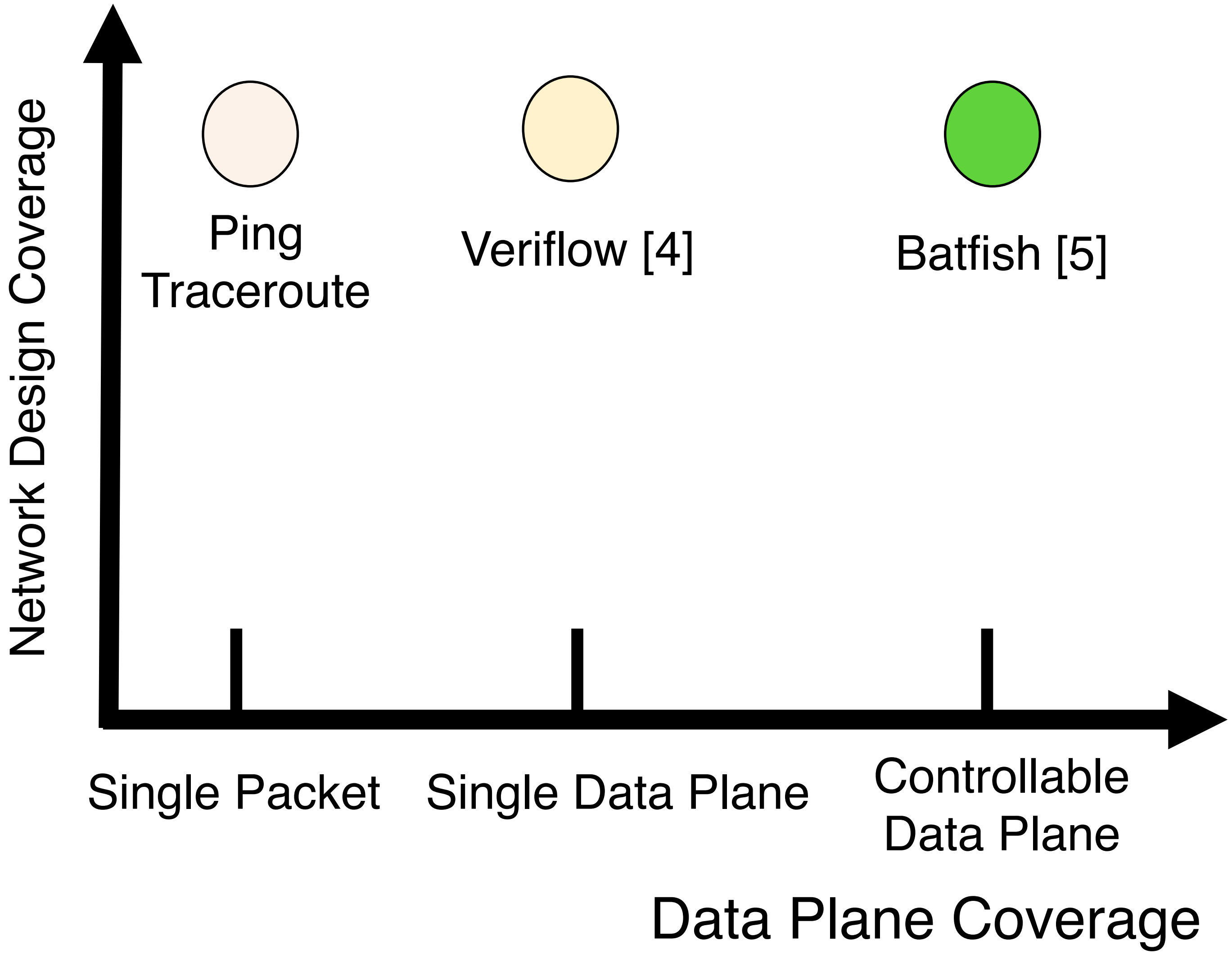
Misconfiguration Finding with Automatic Template Inference

SELFSTARTER [9]

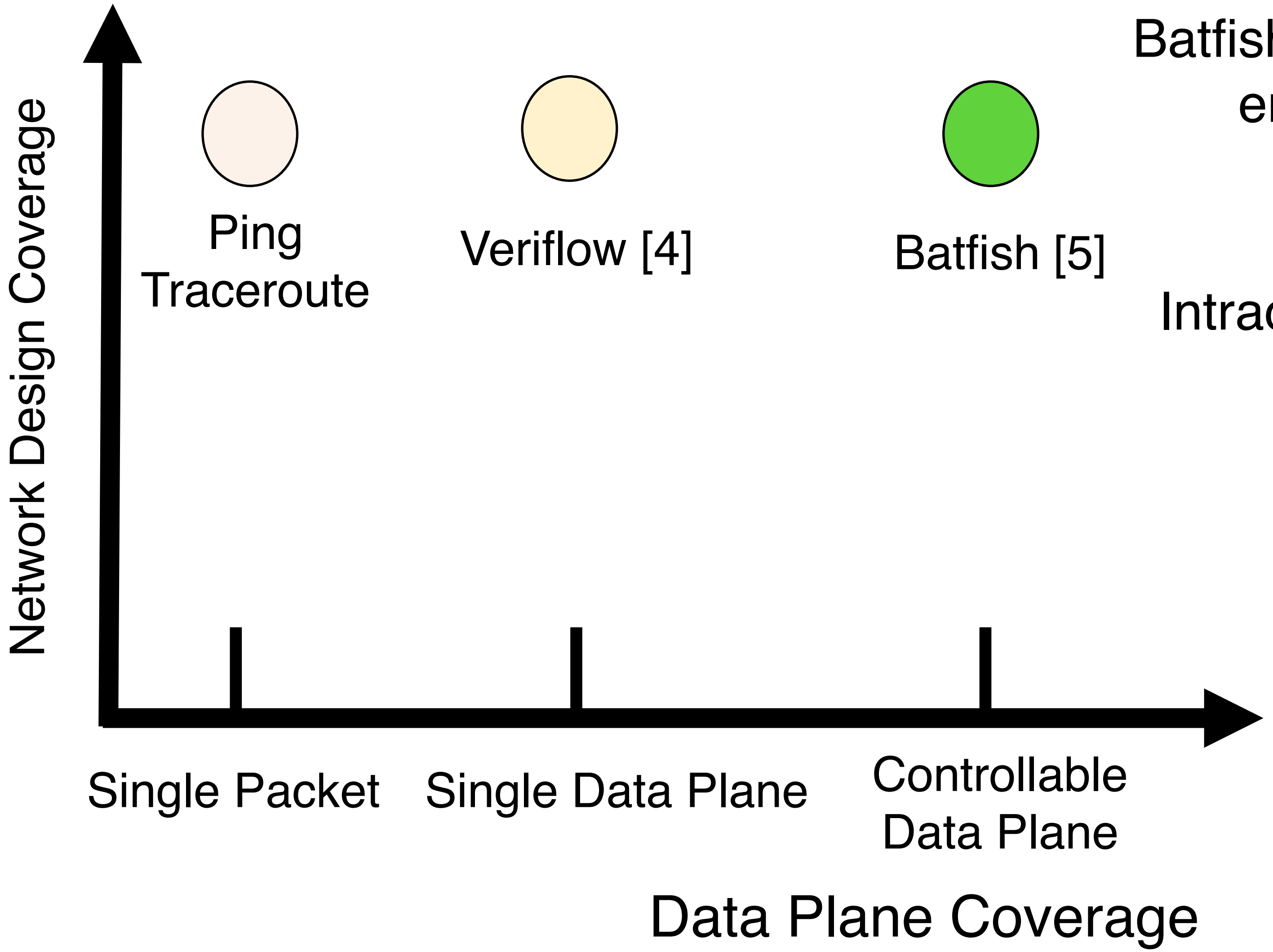
~2012 2015 2017 2020

Time

Progress in Network verification



Progress in Network verification

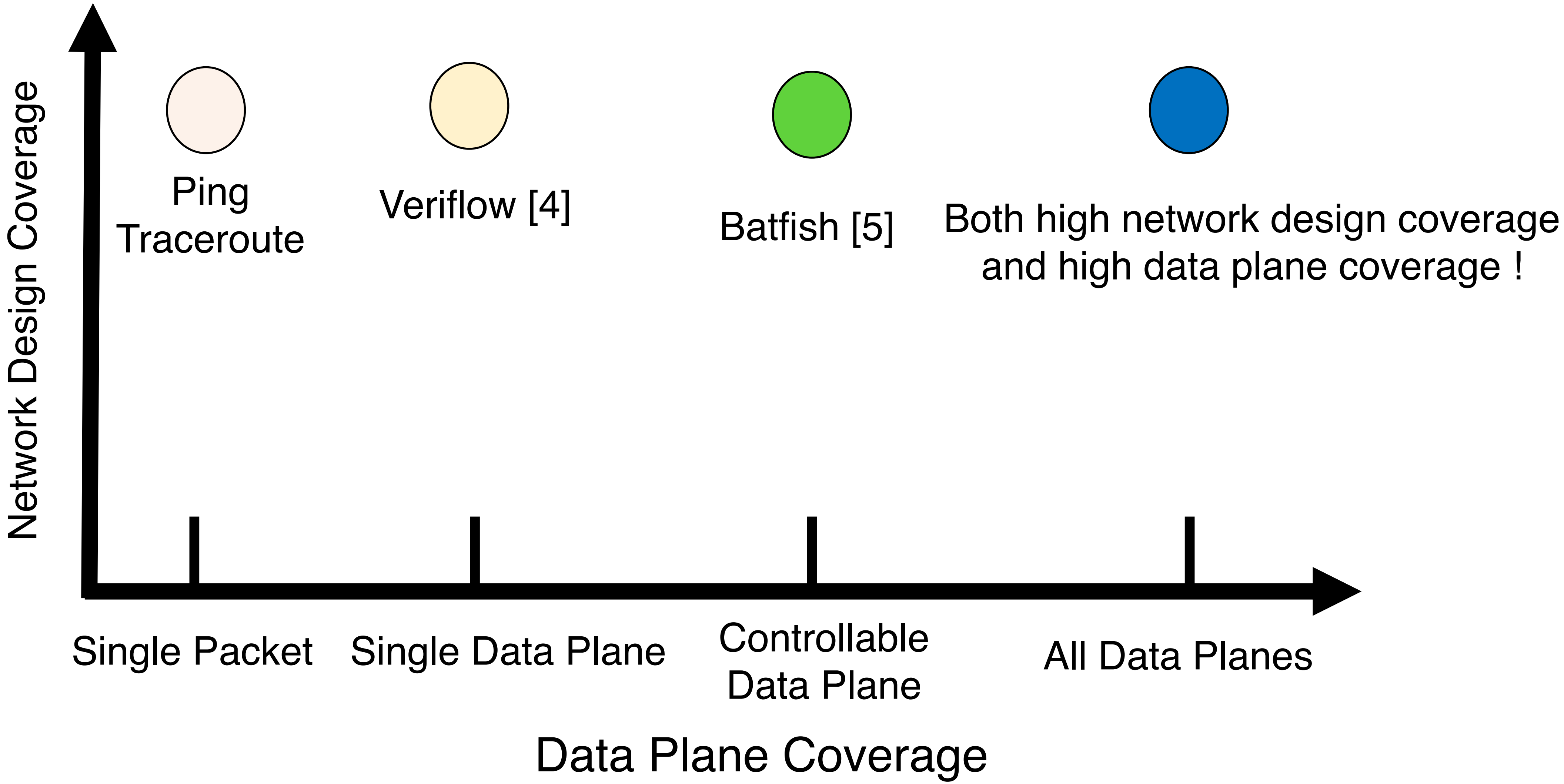


Batfish only explores data plane under the environment from the user input !

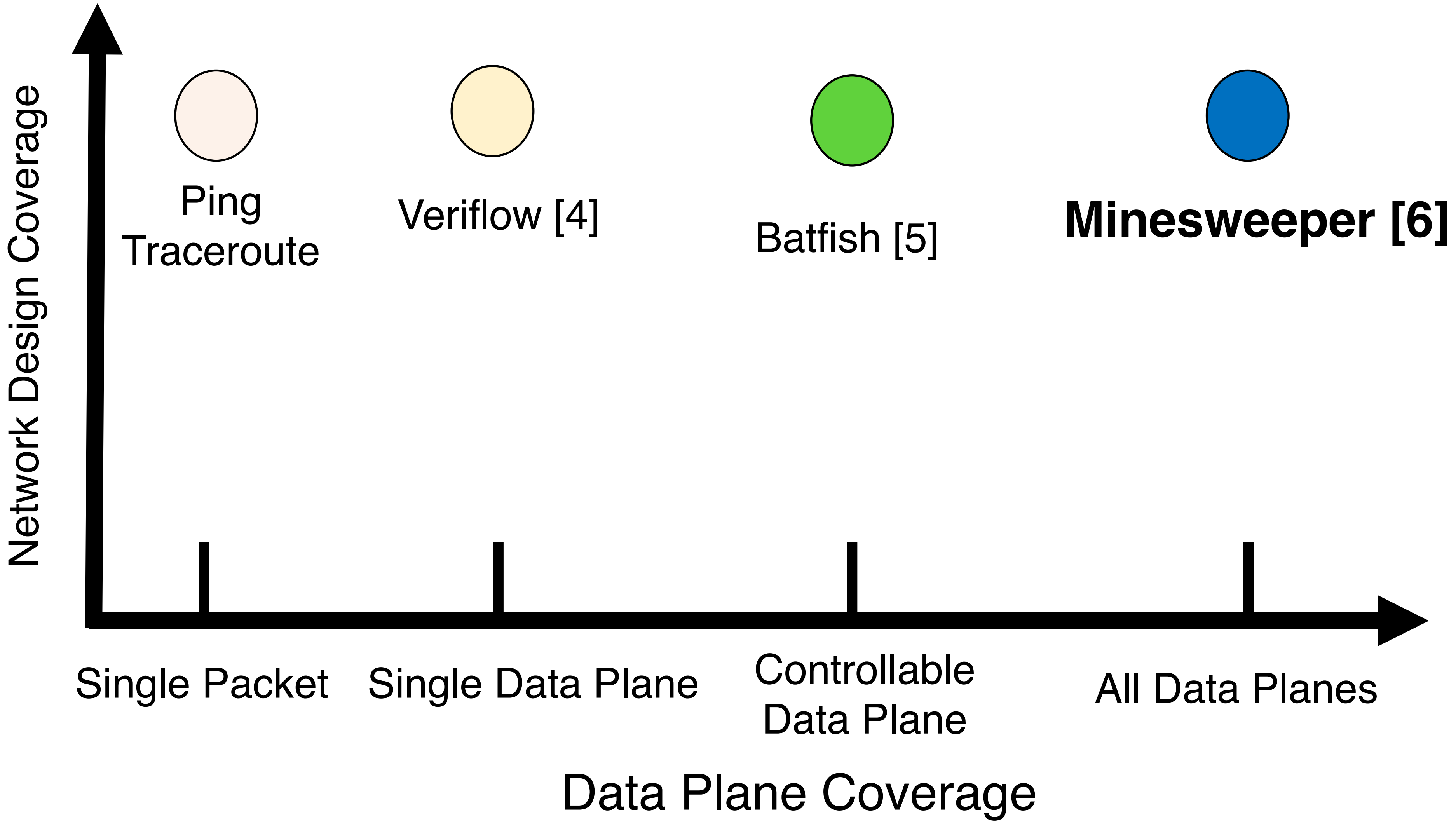
↓

Intractable to guarantee correctness for all possible data plane

Motivation


Network Design Coverage
Ping
Traceroute
Veriflow [4]
Batfish [5]
Both high network design coverage
and high data plane coverage !
Single Packet
Single Data Plane
Controllable
Data Plane
All Data Planes
Data Plane Coverage

Motivation



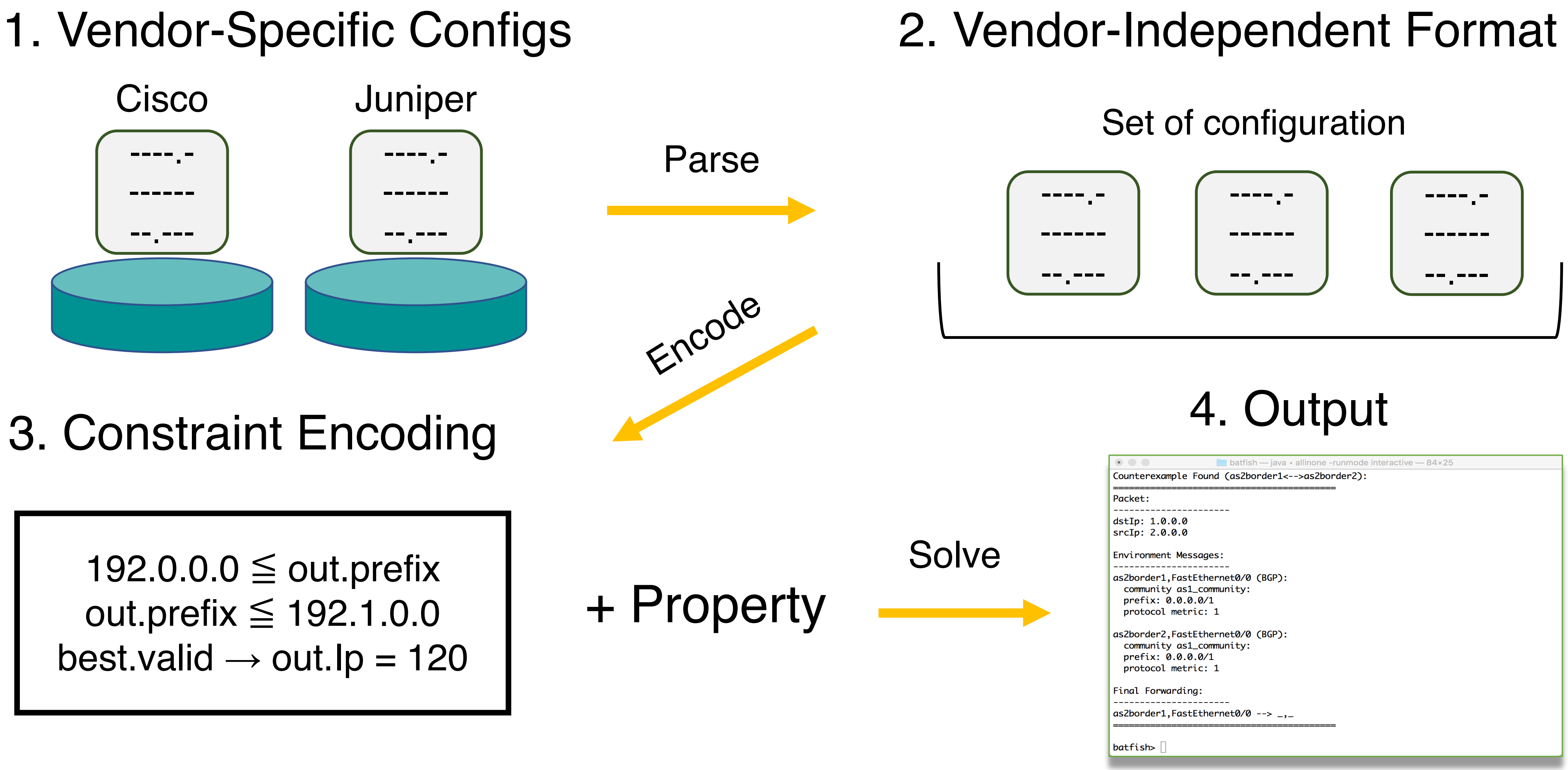
Network Design Coverage
Ping
Traceroute
Veriflow [4]
Batfish [5]
Minesweeper [6]
Single Packet
Single Data Plane
Controllable
Data Plane
All Data Planes
Data Plane Coverage

Minesweeper's key idea

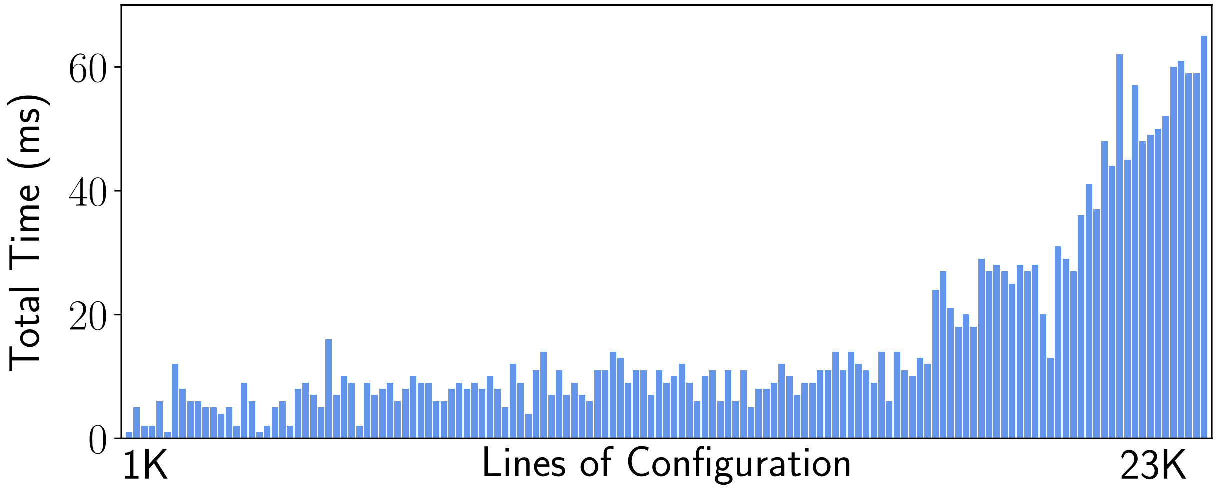
- Encoding the network as a collection of logical constraints
- Solving the constraints leveraging the off-the-shelf solvers

→ Can check many properties for all data planes !

Minesweeper workflow

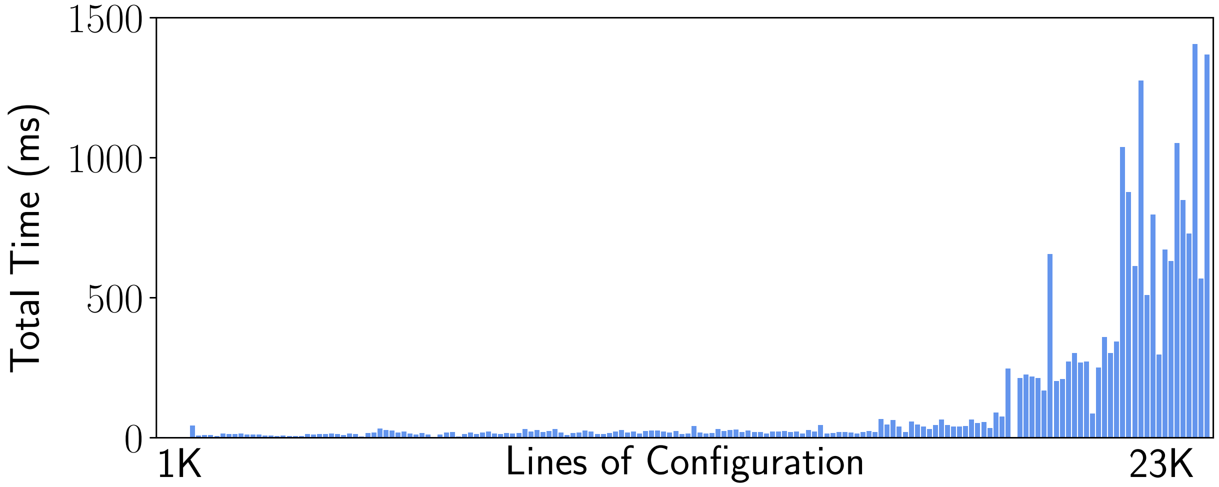


Performance: Scalability



Management interface reachability

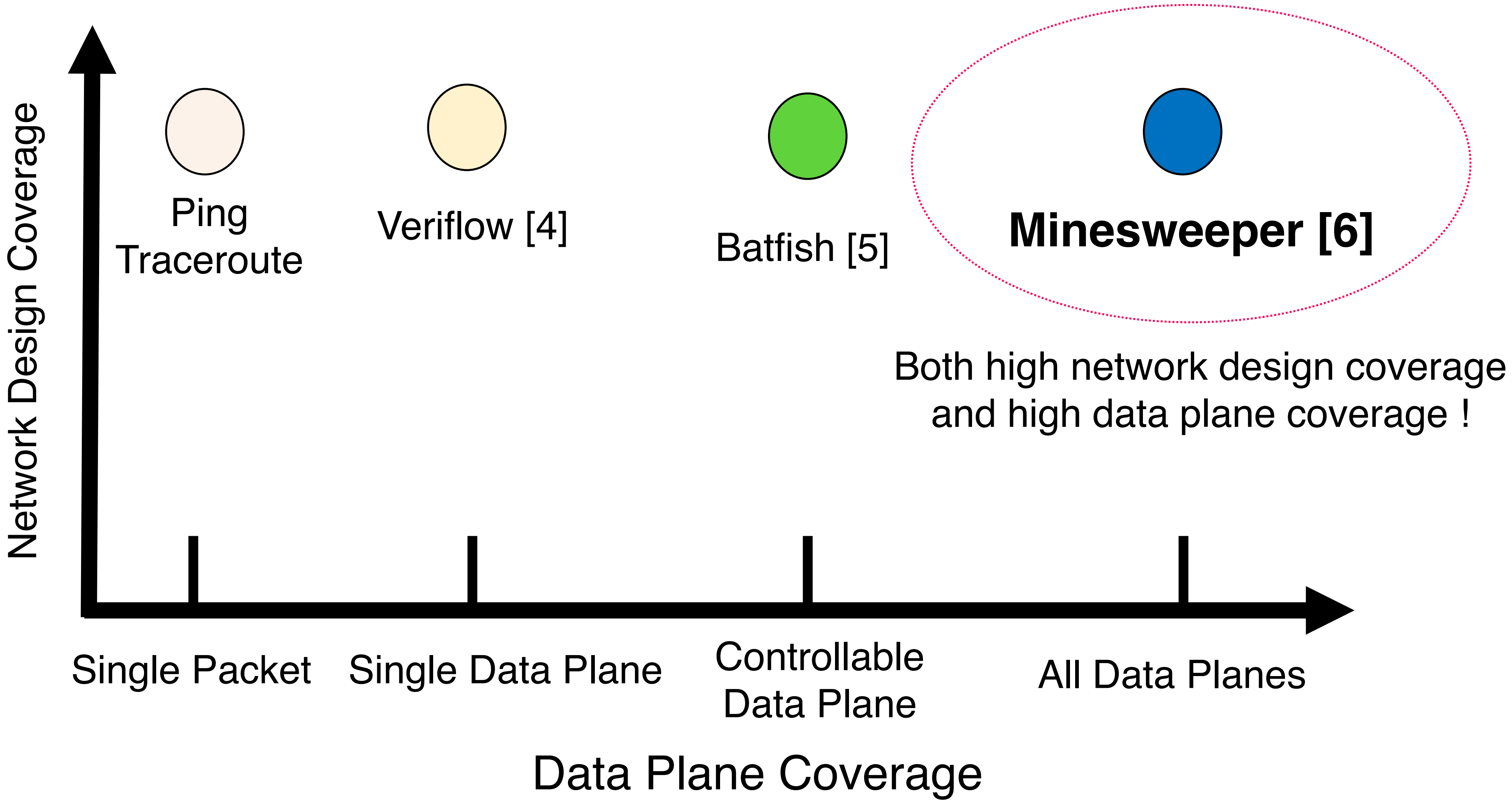
less than 60ms



Black holes only occur at the network edge

less than 1.5sec

Summary



Network Design Coverage
Ping
Traceroute
Veriflow [4]
Batfish [5]
Minesweeper [6]
Both high network design coverage
and high data plane coverage !
Single Packet
Single Data Plane
Controllable
Data Plane
All Data Planes
Data Plane Coverage

Historical progress

~2012

Static Config Analysis

rcc [1]
FIREMAN [2]

Data Plane Analysis

Anteater [3]
Veriflow [4]

2015

Config Analysis with Control & Data Plane

Batfish [5]

2017

Config Analysis with Control & Data Plane (multiple data plane)

Minesweeper [6]

2020

Network Specification Mining

Config2Spec [7]

Config Analysis through model checking (more scalable)

Plankton [8]

Misconfiguration Finding with Automatic Template Inference

SELFSTARTER [9]

Time

Short summary of the three works

Plankton [8]

- A verification tool to scale better than the SMT solver approach utilizing a model checker and domain-specific optimizations

Config2Spec [7]

- Automatically synthesizing a formal specification of a network given its configuration and a failure model

Short summary of the other works

SELFSTARTER [9]

- Automatically identifying configuration outliers by template inference from the configuration

Discussion

- Research such as knowledge extraction from configuration has been popular (not just about scalability and coverage)
- So many research focuses on the reachability
- IPv6 …?

My future Work

- Read the three papers
- Skim through previous papers that have not been yet read

Reference

[1] Detecting BGP Configuration Faults with Static Analysis
Nick Feamster and Hari Balakrishna (NSDI'05)

[2] FIREMAN: A Toolkit for FIREwall Modeling and Analysis
Lihua Yuan et al., (S&P'06)

[3] Debugging the Data Plane with Anteater
Haohui Mai et al., (Sigcomm'2011)

Reference

[4] Veriflow : Verifying Network-Wide Invariants in Real Time
Ahmed Khurshid et al.,(NSDI'13)

[5] A General Approach to Network Configuration Analysis
Ari Fogel et al., (NSDI'15)

[6] A General Approach to Network Configuration Verification
Ryan Beckett et al., (Sigcomm'2017)

Reference

[7] Config2Spec: Mining Network Specifications from Network Configurations Rüdiger Birkner et al.,(NSDI'20)

[8] Plankton: Scalable network configuration verification through model checking, Santhosh Prabhu et al., (NSDI'20)

[9] Finding Network Misconfigurations by Automatic Template Inference, Siva Kesava Reddy Kakarla et al., (NSDI'20)