NV

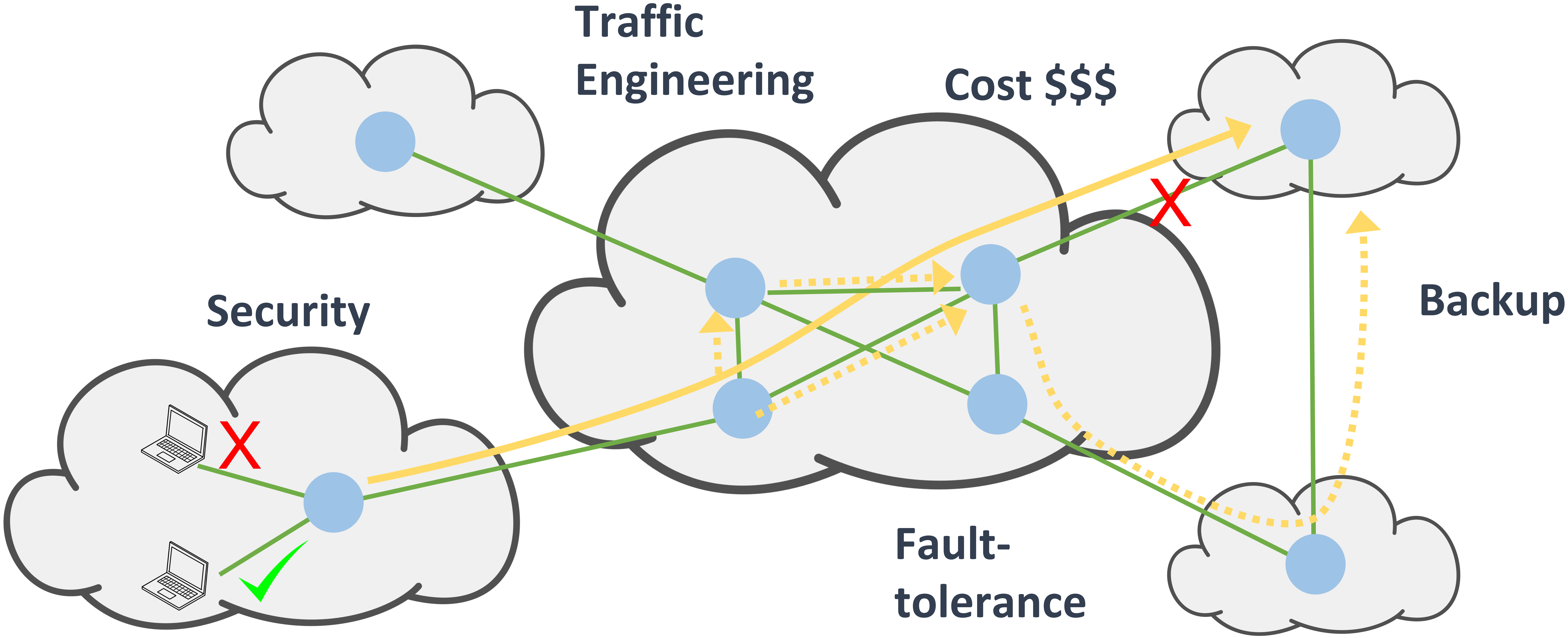
A platform for modelling and verifying routing protocols

Ryan Beckett*+ Nick Giannarakis* Devon Loehr*
Aarti Gupta* Ratul Mahajan! David Walker*

* Princeton University
\+ Microsoft Research
! Intentionet/UW

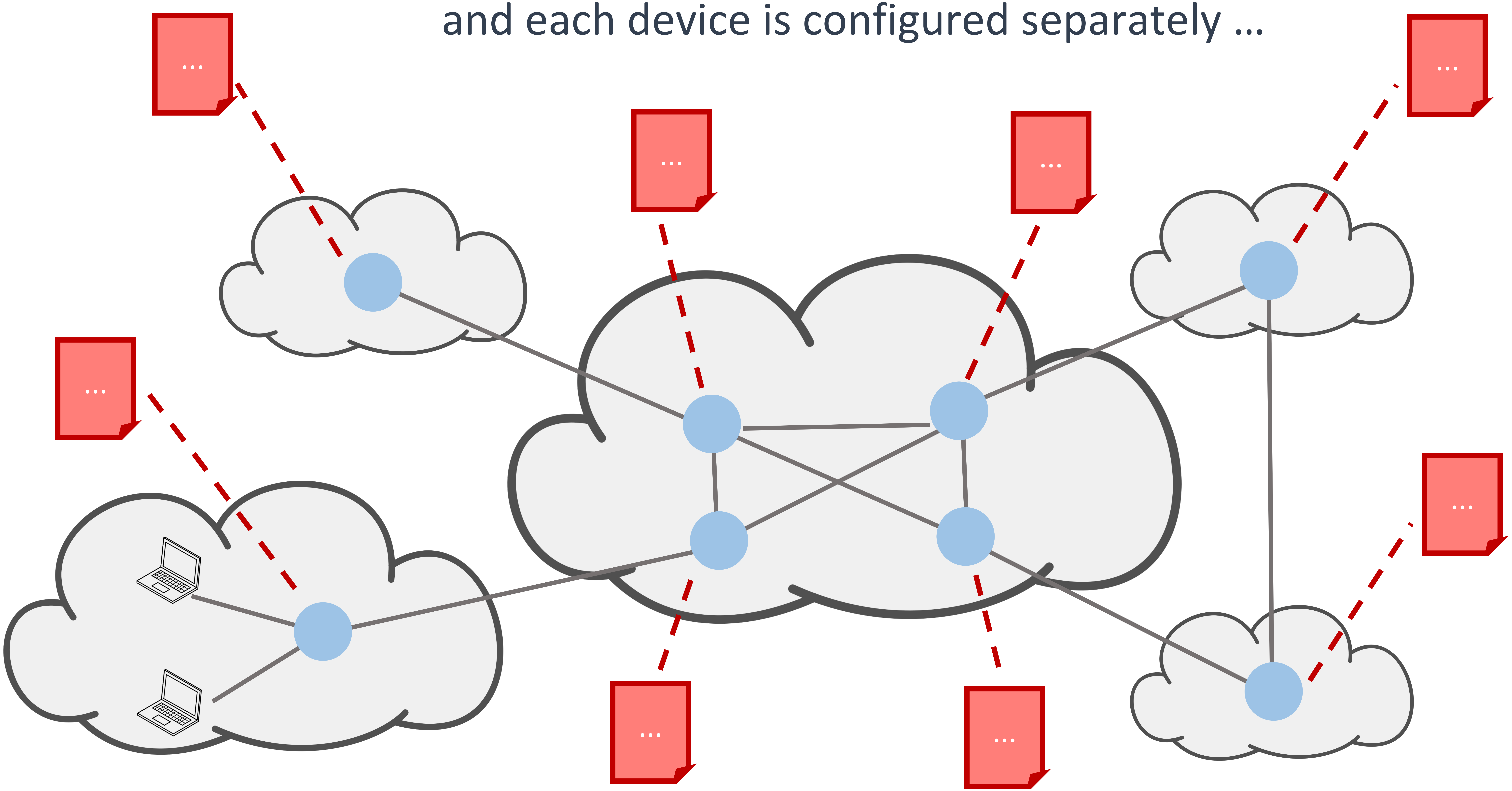
Modern Networks

are complicated things …



Modern Networks

and each device is configured separately …



South Africa: FNB solves crippling connectivity issues

July 25, 2016 • Finance, Southern Africa, Top Stories

Microsoft: misconfigured network device led to Azure outage

30 July 2012 | By Yevgeniy Sverdlik

BGP errors are to blame for Monday's Twitter outage, not DDoS attacks

No, your toaster didn't kill Twitter, an engineer did

Router Crashes Trigger Major Southwest IT System Failure

By: Chris Preimesberger | July 21, 2016

Unions want Southwest CEO removed after IT outage

BlackBerry outage could cost RIM $100 million

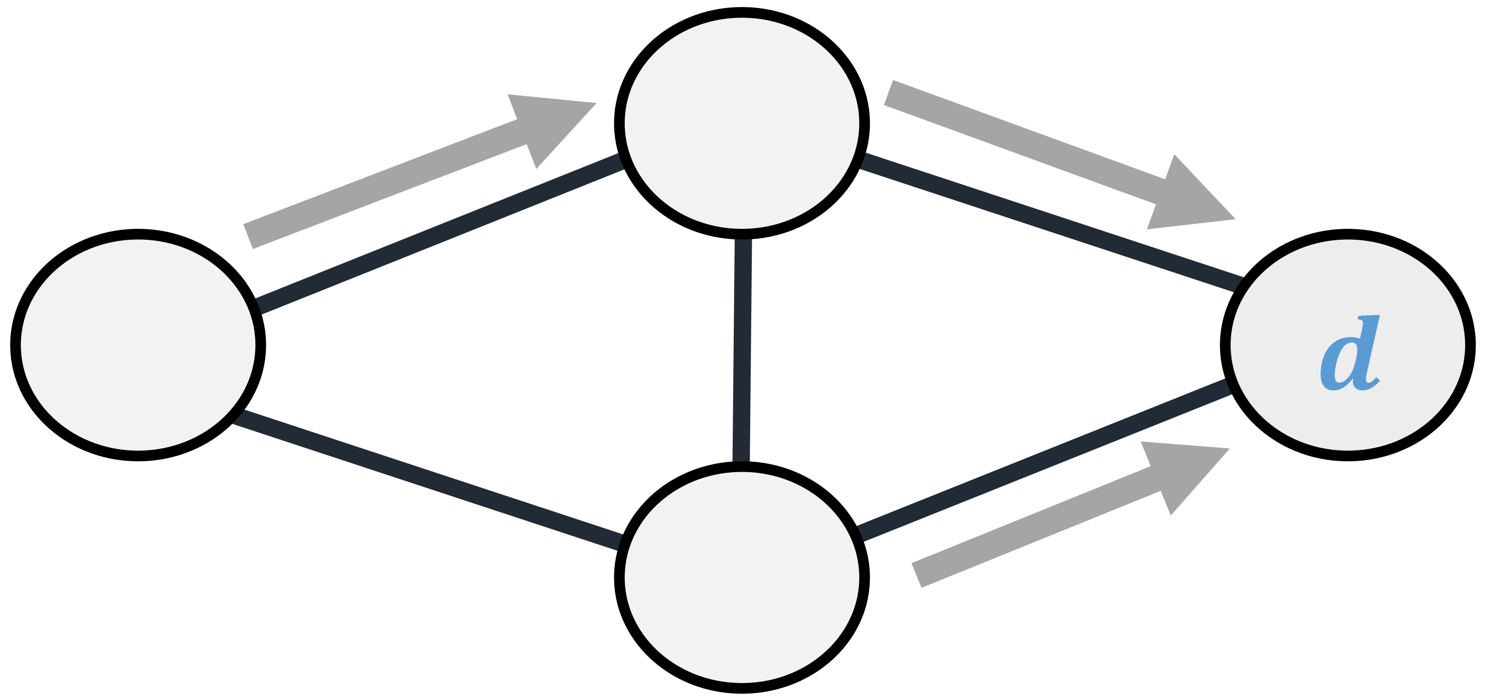
Massive route leak causes Internet slowdown

Posted by Andree Toonk – June 12, 2015 – BGP instability – No Comments

Xbox Live outage caused by network configuration problem

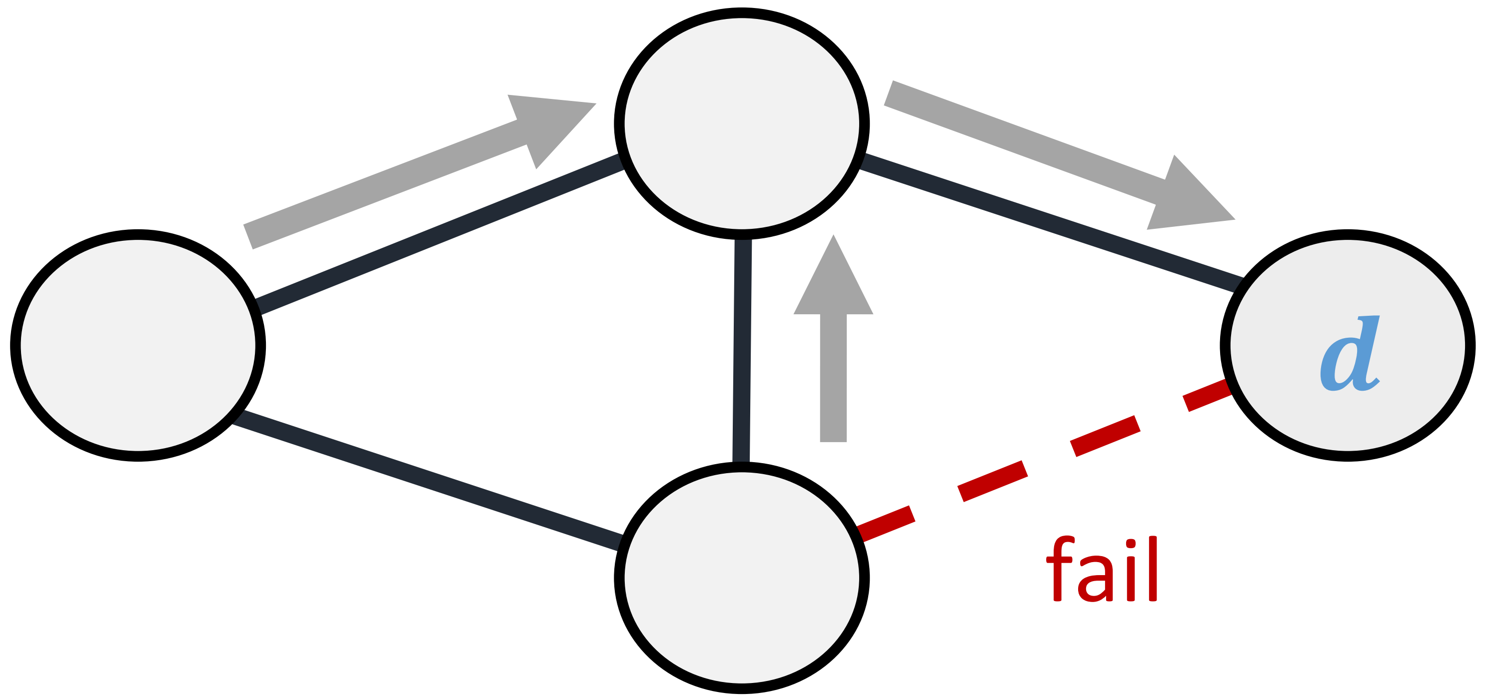
BY TODD BISHOP on April 15, 2013 at 9:27 am

Good news! Some Solutions



The data plane:

A snapshot at one instant in time of how a network forwards traffic.



The control plane:

The algorithms that figure out which routes to use and react to environmental changes over time, producing a series of data planes.

Good news! Some Solutions

Data Plane Verification

Anteater	[Mai 2011]
HSA	[Kazemian 2012]
Veriflow	[Kurshid 2013]
NetKAT	**[Anderson 2014]**
NoD	[Lopes 2015]
Symmetries	[Plotkin 2016]
…	

Good news! Some Solutions

Data Plane Verification

Anteater	[Mai 2011]
HSA	[Kazemian 2012]
Veriflow	[Kurshid 2013]
NoD	[Lopes 2015]
Symmetries	[Plotkin 2016]
…	

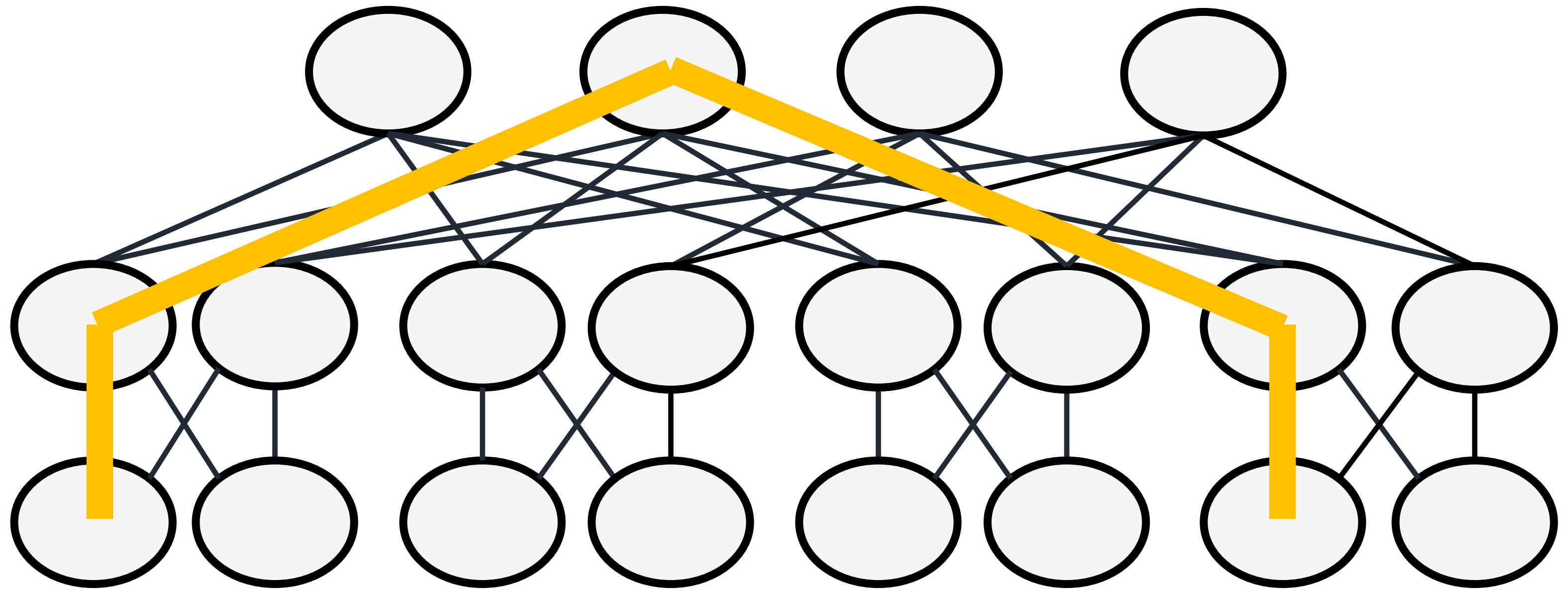
Control Plane Simulation

C-BGP	[Quotin 2005]
Batfish	[Fogel 2015]
…	

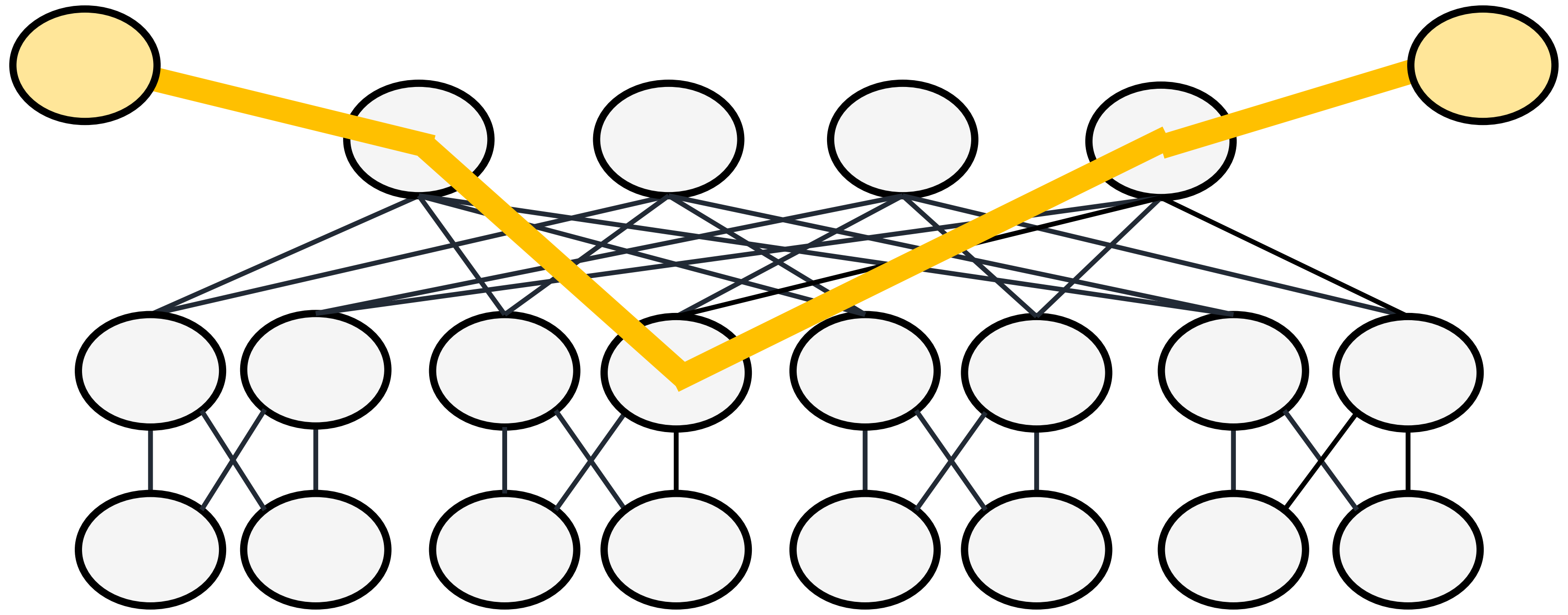
Control Plane Verification

Bagpipe	[Weitz 2016]
ARC	[Gember-Jacobsen 2016]
ERA	[Fayaz 2017]
MineSweeper	[Beckett 2017]
…	

Properties (*for all* data planes produced)

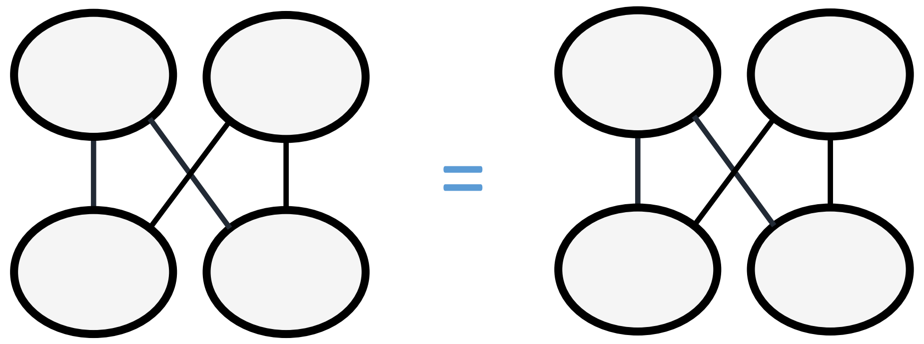


reachability



no transit

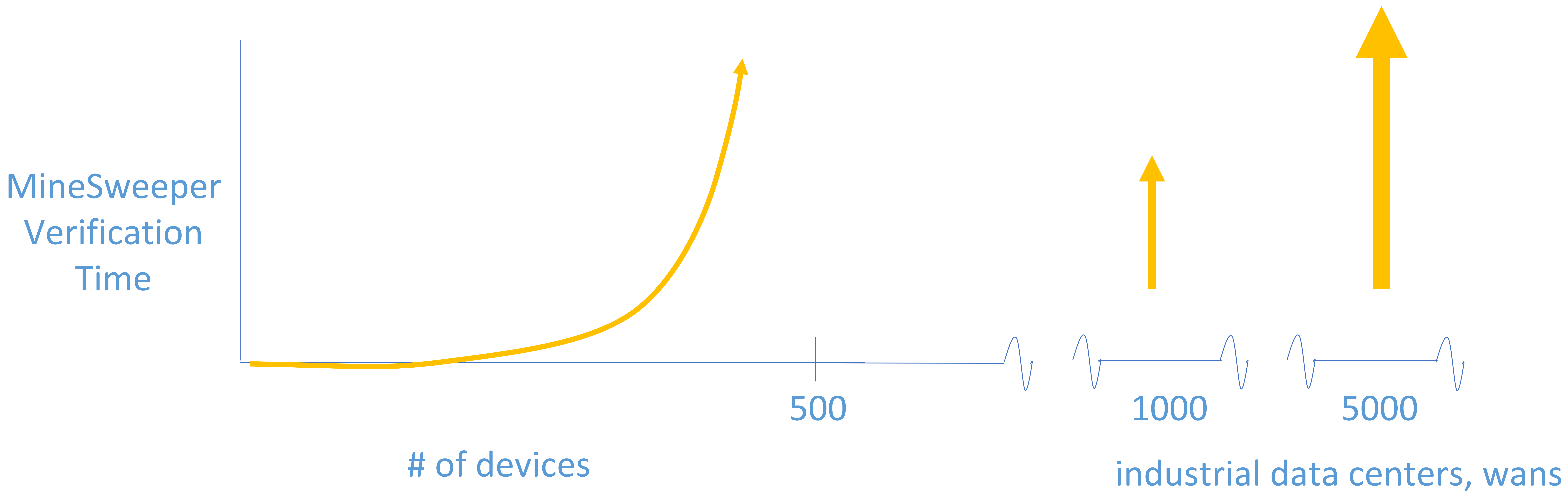
no black holes



router or subnet equivalence

no congestion

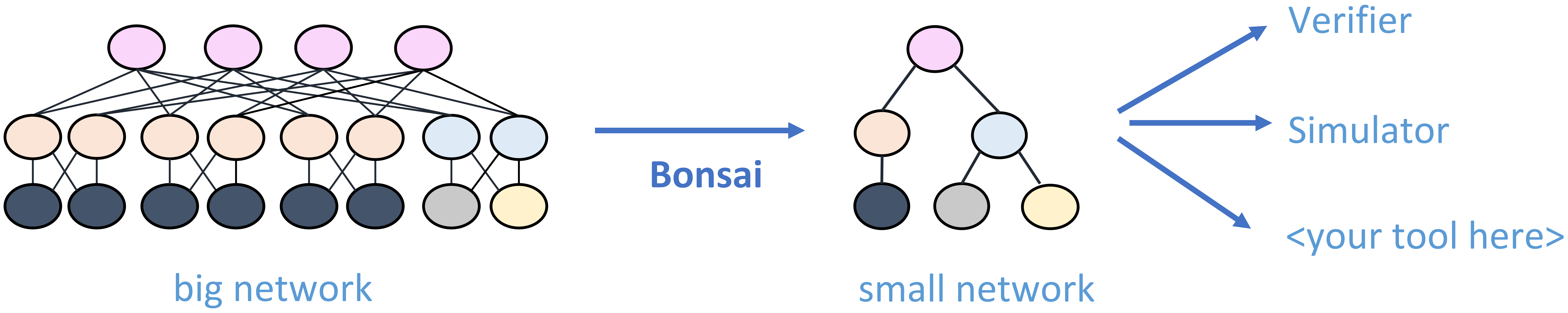
A Problem of Scale



Other technologies, such as simulation, suffer similar, though less severe trends.

To Cope with Scale

Implement transformations that collapse symmetries or abstract away details



Bonsai

big network

small network

Verifier

Simulator

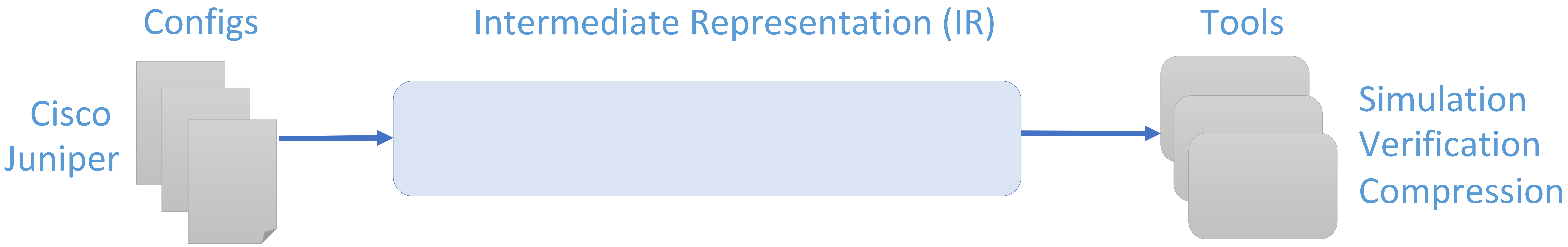
<your tool here>

Transformations:

- Topological transformations: Bonsai [SIGCOMM 18], Origami [CAV 19]
- Message abstractions: ShapeShifter
- Divide and conquer tactics
- Conventional optimization [dead code, constant folding, slicing]
- Specialization [per destination, source-dest]

How should we build a tool suite for network reliability?

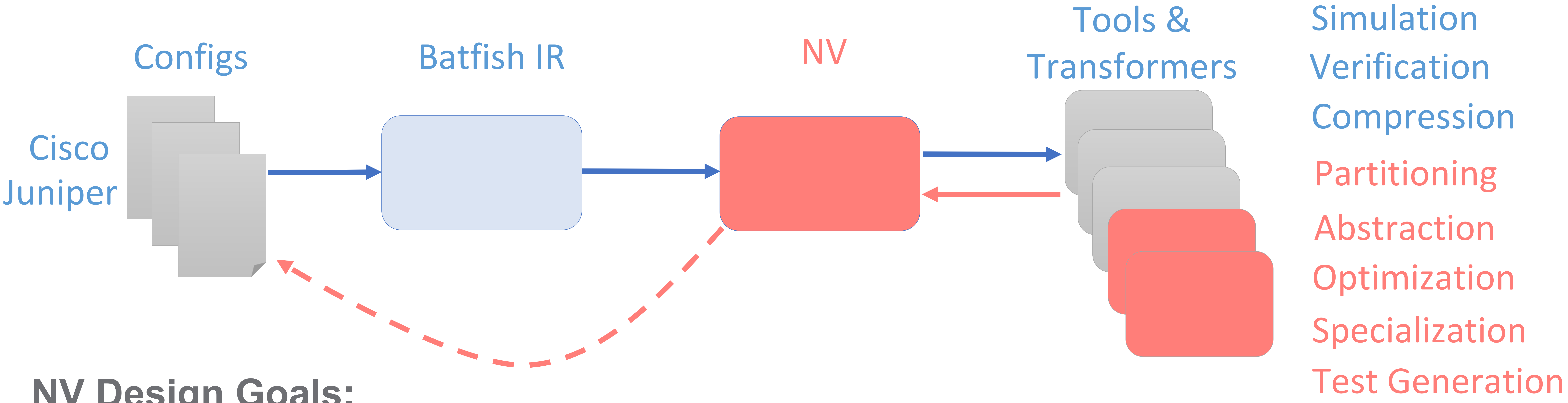
Batfish [Fogel et al, 2015]



IR Characteristics:

- ***Indispensible***: represents a very wide range of configs
- ***Massive***: for route maps: 105 expressions, 23 statements (30-100LOC/class)
- ***Specialized, not orthogonal***: 19 different expressions to "set" things: tags, AS path, …
- ***Non-compositional***: can't build complex structures from simpler ones
- ***Not reuseable***: hard to reuse optimizations from one tool to another
- ***Inexpressive***: new config features often need extensions; not designed as a tool target
- ***Designed for experts***: deep knowledge of networks needed to grok it
- ***Semi-implicit semantics***: some effects happen implicitly (need to look at simulator)

Batfish → NV



Configs

Cisco
Juniper

Batfish IR

NV

Tools &
Transformers

Simulation
Verification
Compression
Partitioning
Abstraction
Optimization
Specialization
Test Generation

NV Design Goals:

- ***Conventional***: mostly ordinary (functions, records, options, …, *dictionaries*)
- ***Minimal & Orthogonal***: one operation for record projection
- ***Compositional***: complex data from simple primitives
- ***Expressive***: new config features usually *don't* need extensions
- ***Tractable***: … but semantics can be translated into decideable logics (SMT)
- ***Designed for non-experts***: deep knowledge of networks *not* needed to grok it
- ***Well-defined semantics:*** every program has a rigorous, mathematical meaning
- ***Explicit semantics***: no implicit semantic side effects
- ***Verification support***: facilities to declare unknowns, requirements and specifications

Moral of the Story

Moral of the Story

To build reliable networking infrastructure in the 2020s,

use functional programming from the 1980s

to model network control planes.

“There are two kinds of applause: The kind you earn or ‘cheap applause,’ the kind you get by pandering to the audience .... I am a fan of both.”

-- Lady Gaga

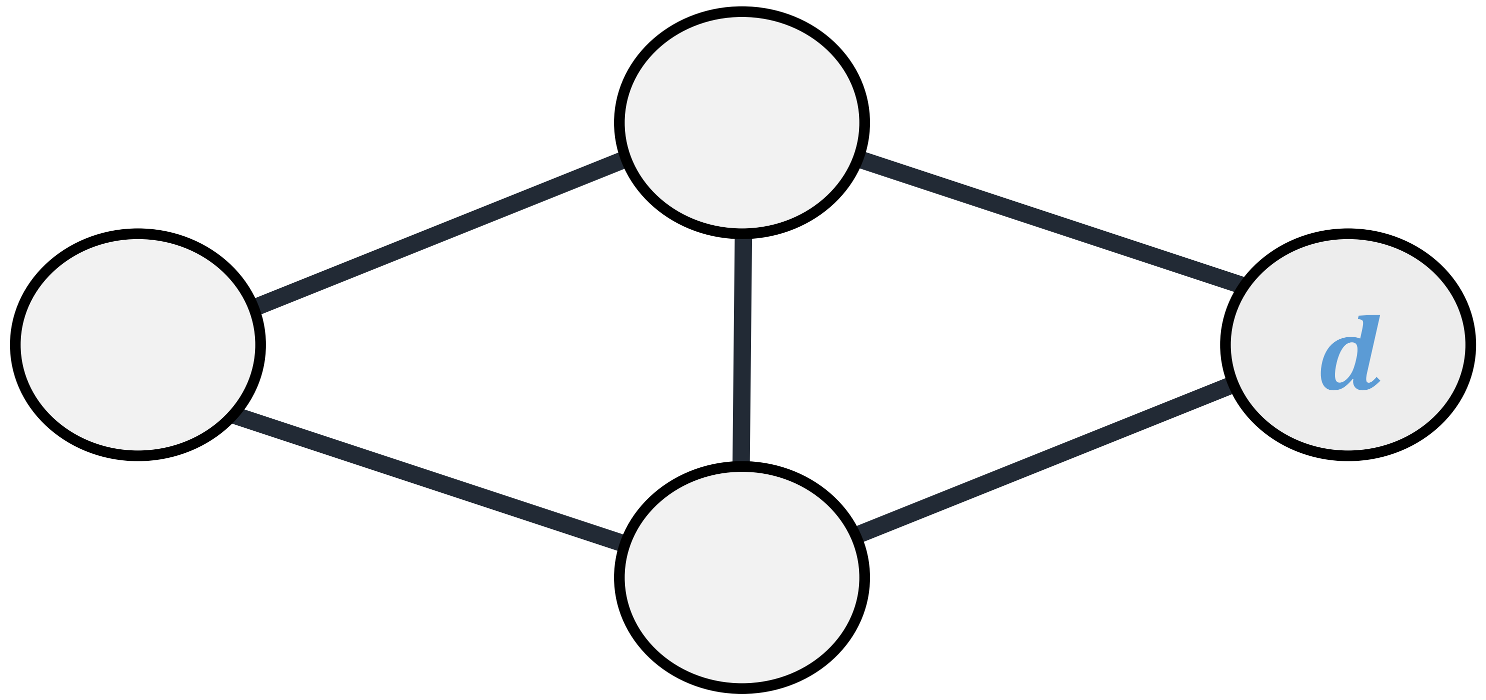
Modelling a Routing Protocol

Thanks to Griffin, Wilfong and Sobrinho's work on
Stable Paths Problem, Routing Algebras, Metarouting
2000-2005

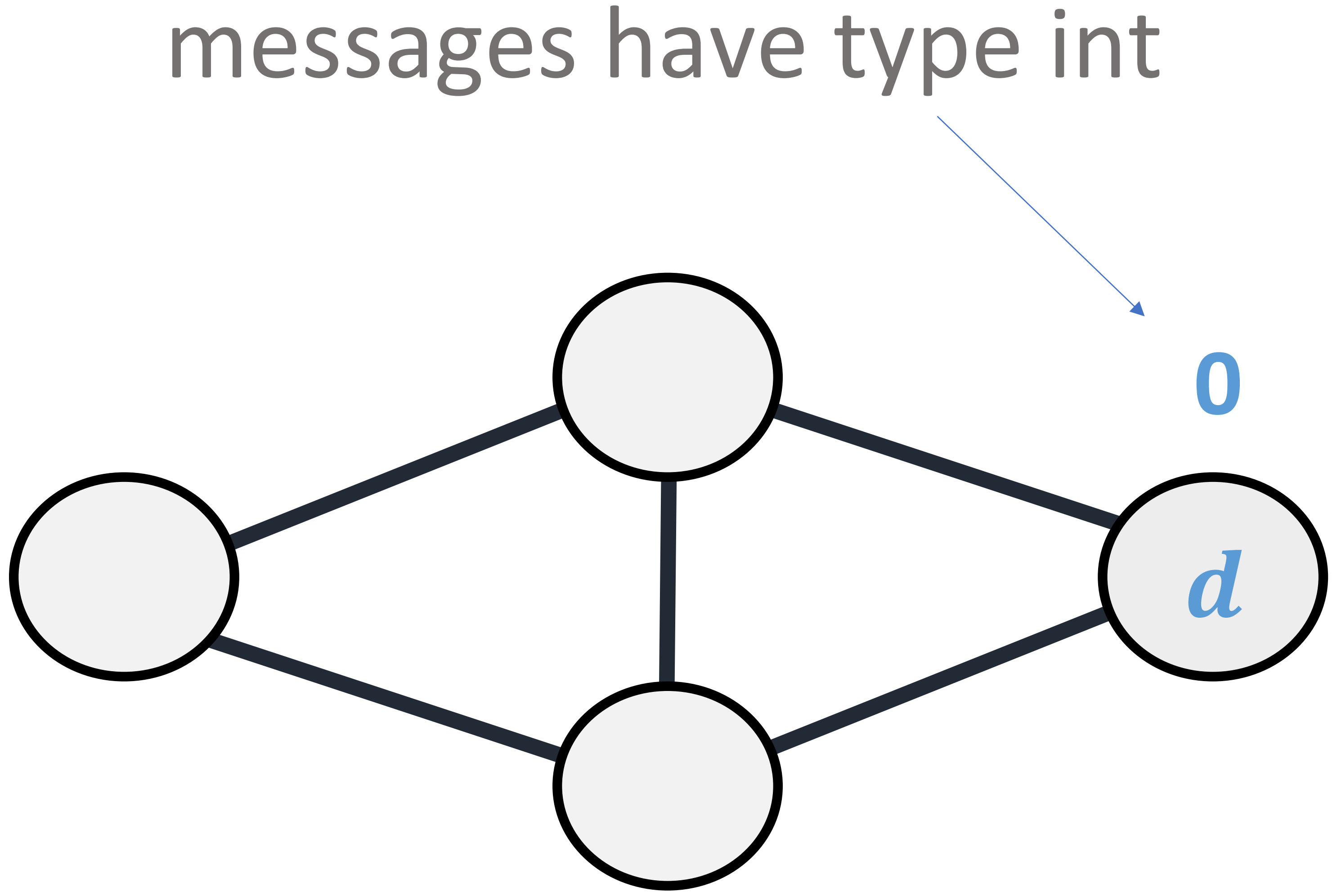
Modelling a Routing Protocol (Instance)

- Idealized RIP: shortest paths routing
- Route announcements are integers that count the number of hops to the destination

Modelling a Routing Protocol (Instance)

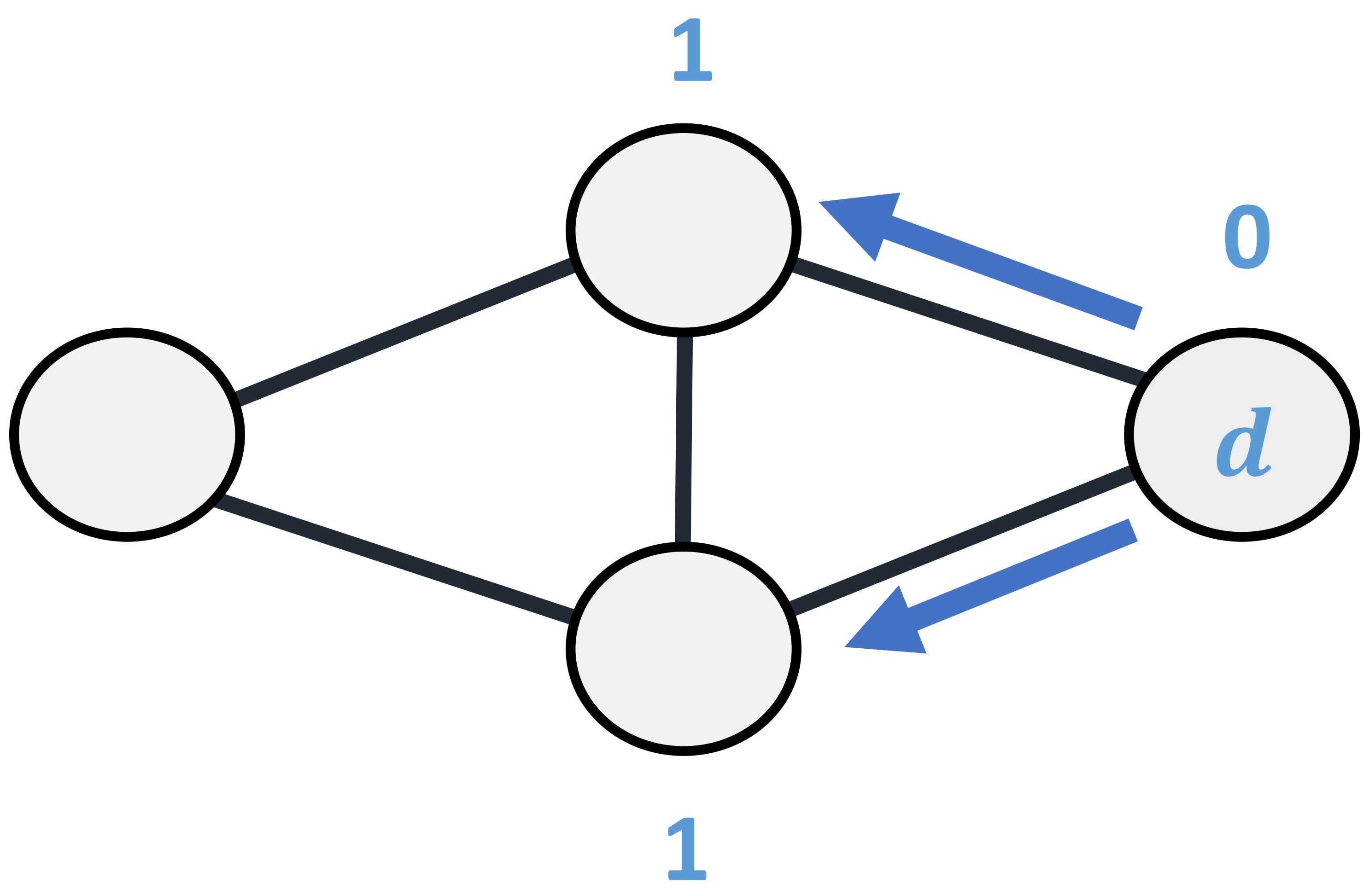


Modelling a Routing Protocol (Instance)



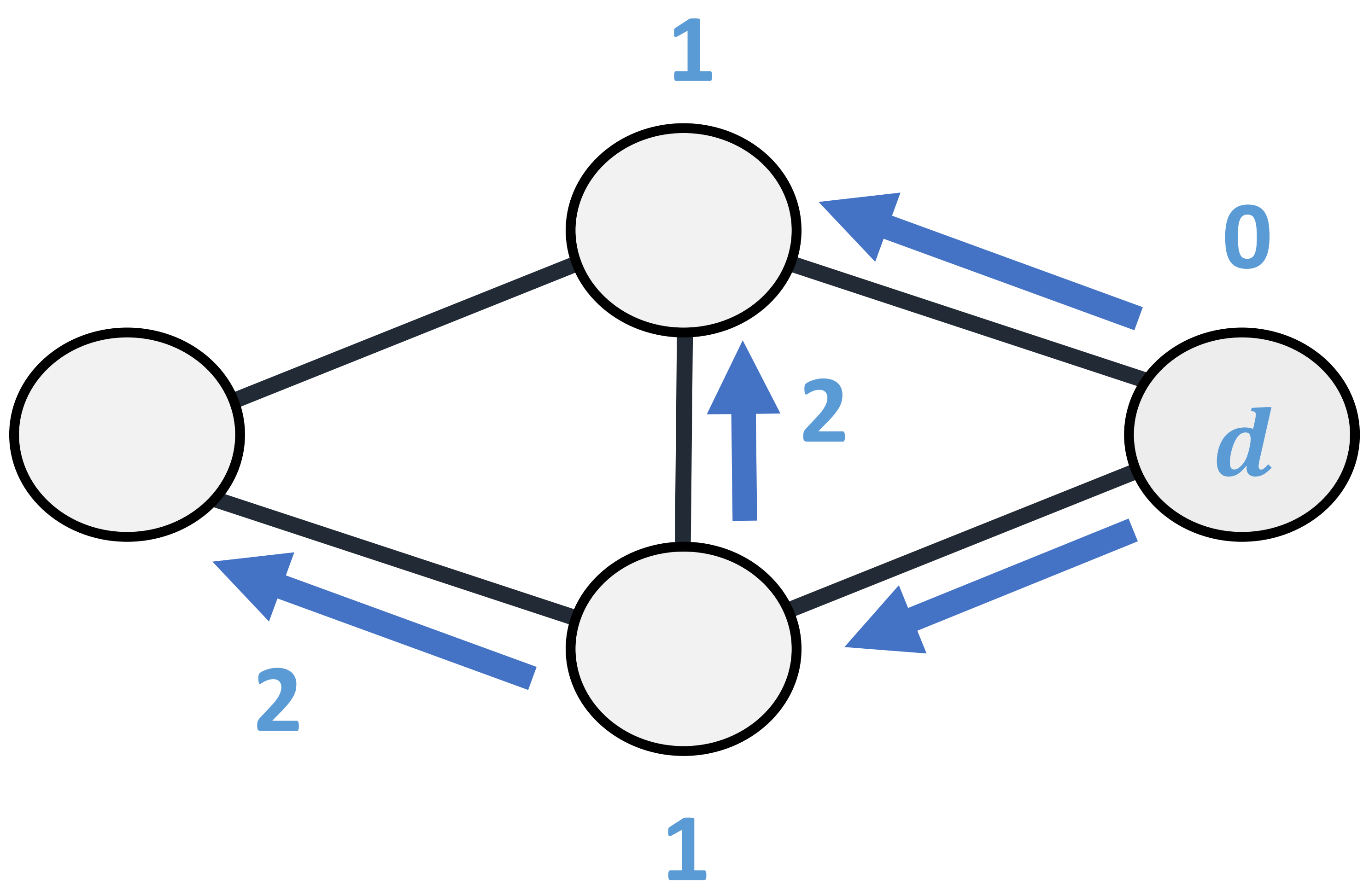
- **The origin creates an initial announcement stating it has a path to destination *d***

Modelling a Routing Protocol (Instance)



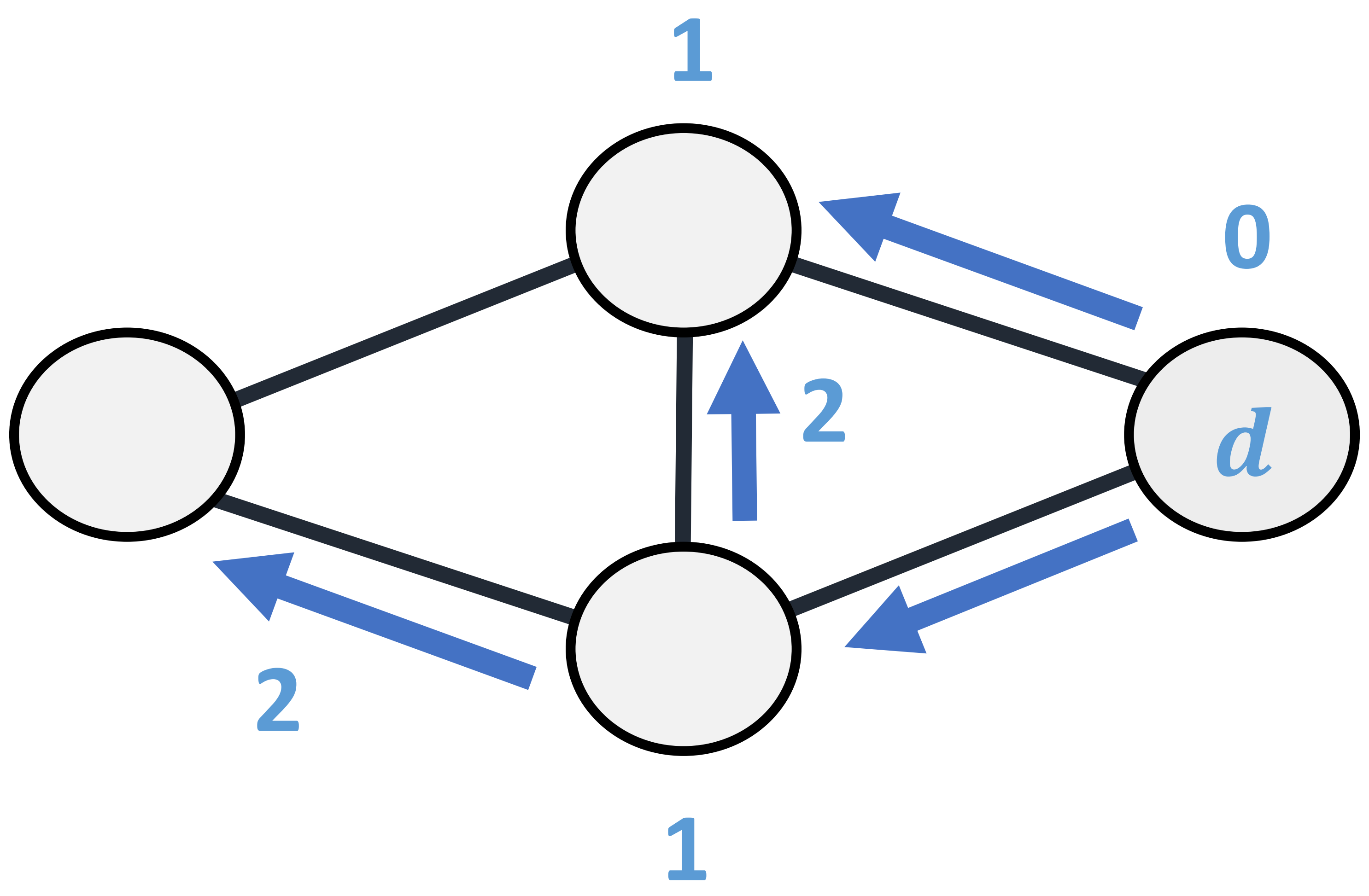
- **The origin creates an initial announcement stating it has a path to destination *d***
- **The announcement is transmitted along edges to neighbors, often modified (or dropped) as it goes.**

Modelling a Routing Protocol (Instance)



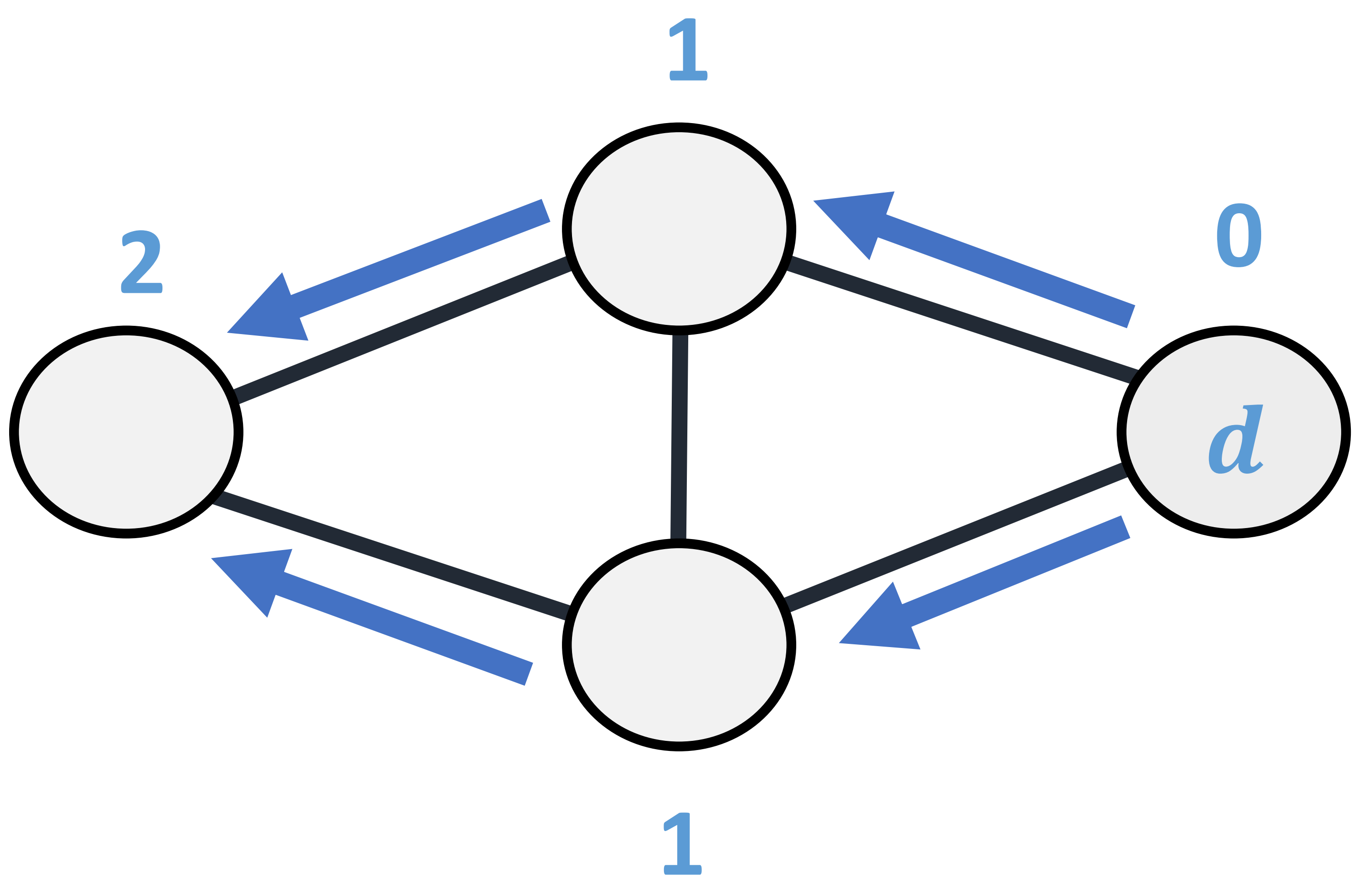
- **The origin creates an initial announcement stating it has a path to destination *d***
- **The announcement is transmitted along edges to neighbors, often modified (or dropped) as it goes.**

Modelling a Routing Protocol (Instance)



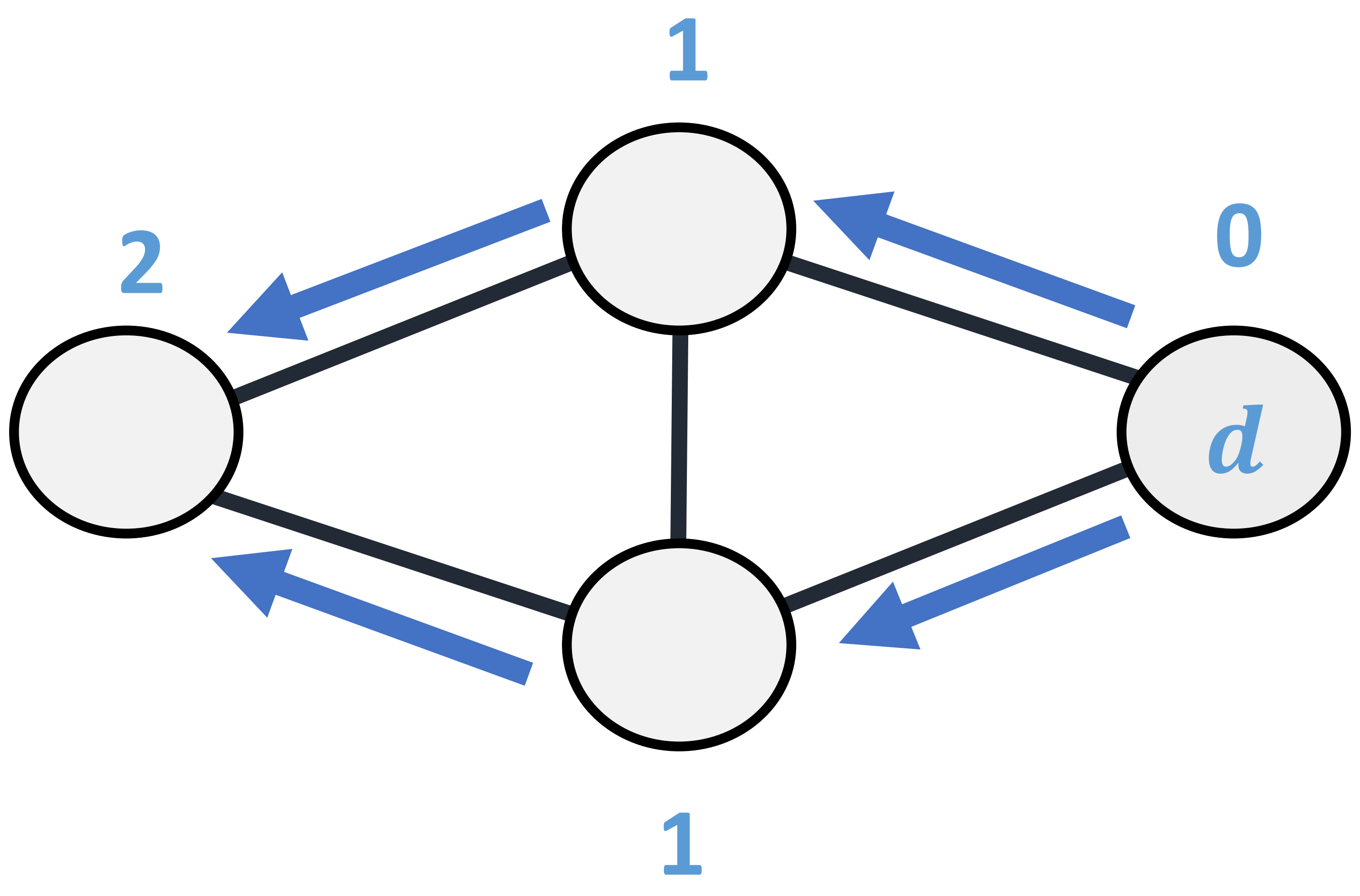
- The origin creates an initial announcement stating it has a path to destination *d*
- The announcement is transmitted along edges to neighbors, often modified (or dropped) as it goes.
- When nodes receive multiple announcements, they choose a best one

Modelling a Routing Protocol (Instance)



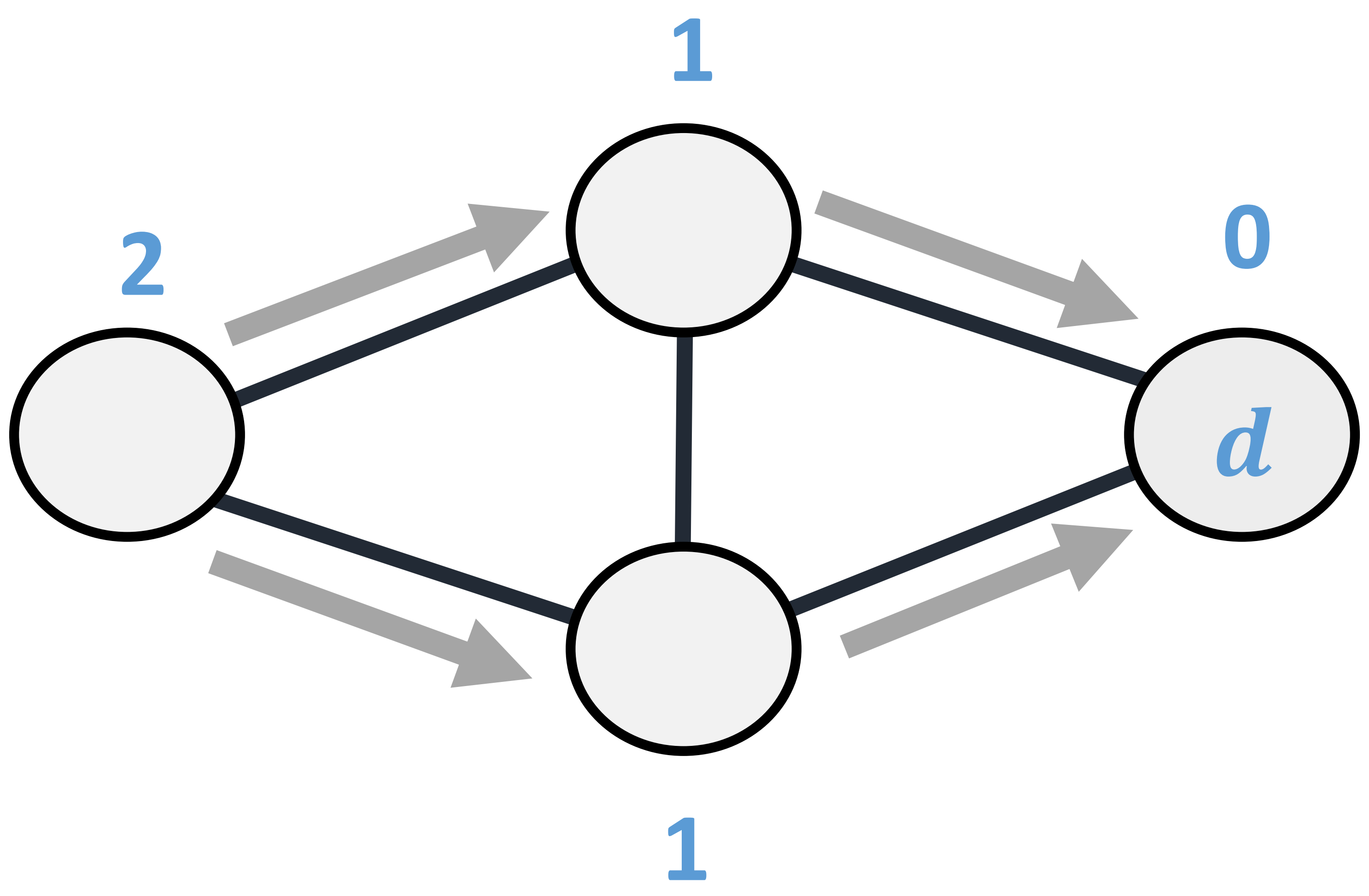
- **The origin creates an initial announcement stating it has a path to destination *d***
- **The announcement is transmitted along edges to neighbors, often modified (or dropped) as it goes.**
- **When nodes receive multiple announcements, they choose a best one**

Modelling a Routing Protocol (Instance)



- The origin creates an initial announcement stating it has a path to destination *d*
- The announcement is transmitted along edges to neighbors, often modified (or dropped) as it goes.
- When nodes receive multiple announcements, they choose a best one
- Eventually (hopefully), the system converges on a solution: all nodes have selected the best route amongst all available to them and no more changes occur; nodes forward in the opposite direction of announcements

Modelling a Routing Protocol (Instance)

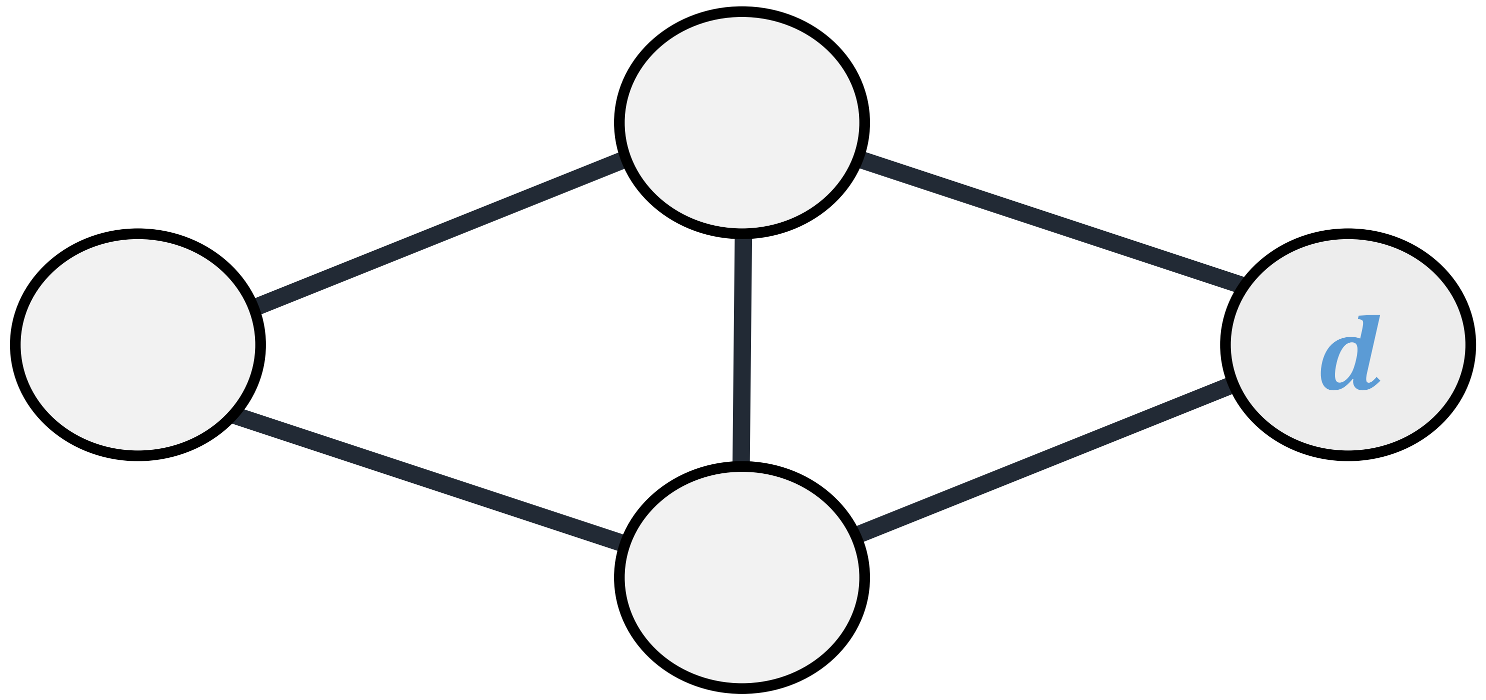


forwarding can usually be inferred from the solution

opposite to the flow of messages

- **The origin creates an initial announcement stating it has a path to destination *d***
- **The announcement is transmitted along edges to neighbors, often modified (or dropped) as it goes.**
- **When nodes receive multiple announcements, they choose a best one**
- **Eventually (hopefully), the system converges on a solution: all nodes have selected the best route amongst all available to them and no more changes occur; nodes forward in the opposite direction of announcements**

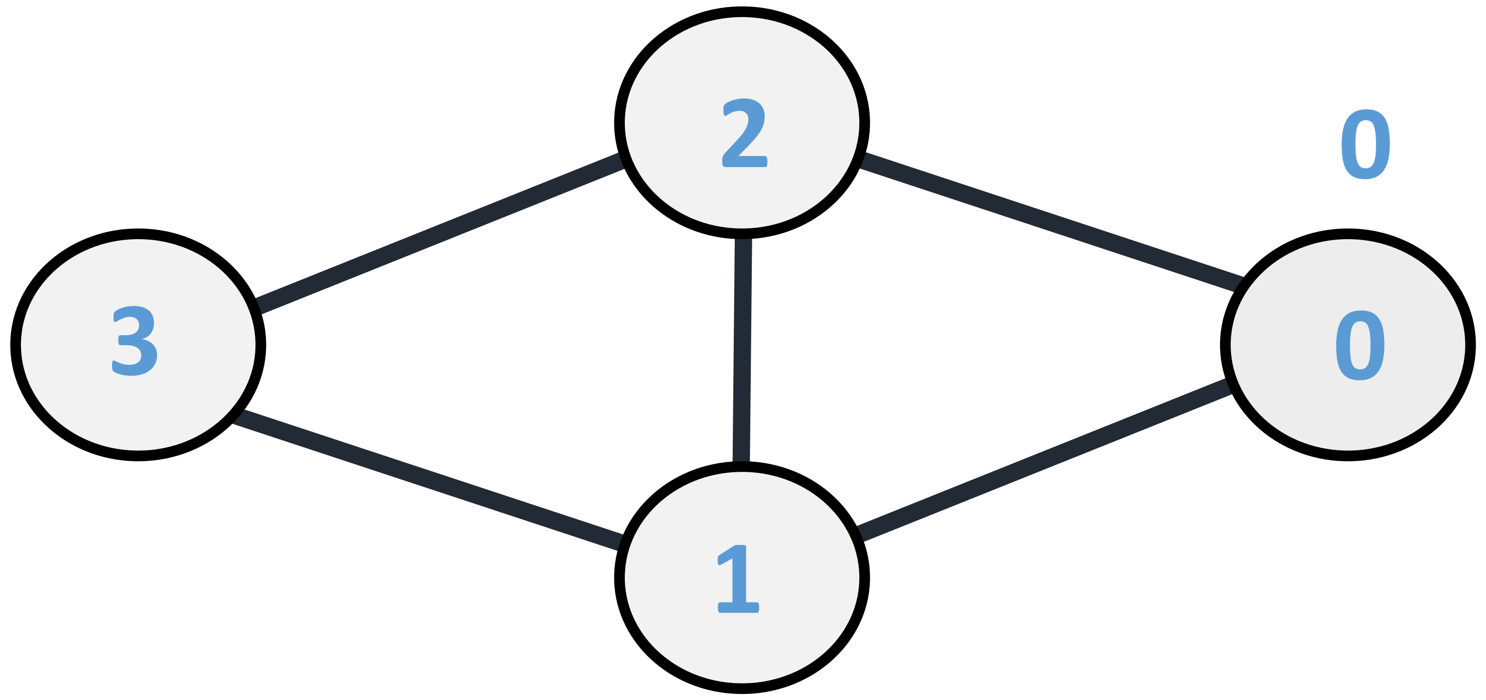
Modelling a Routing Protocol (Instance)



- The origin creates an ***initial announcement*** stating it has a path to destination *d*
- The announcement is ***transmitted along edges to neighbors, often modified*** (or dropped) as it goes.
- When nodes receive multiple announcements, they ***choose a best one***
- Eventually (hopefully), the system converges on a solution: all nodes have selected the best route amongst all available to them and no more changes occur; nodes forward in the opposite direction of announcements

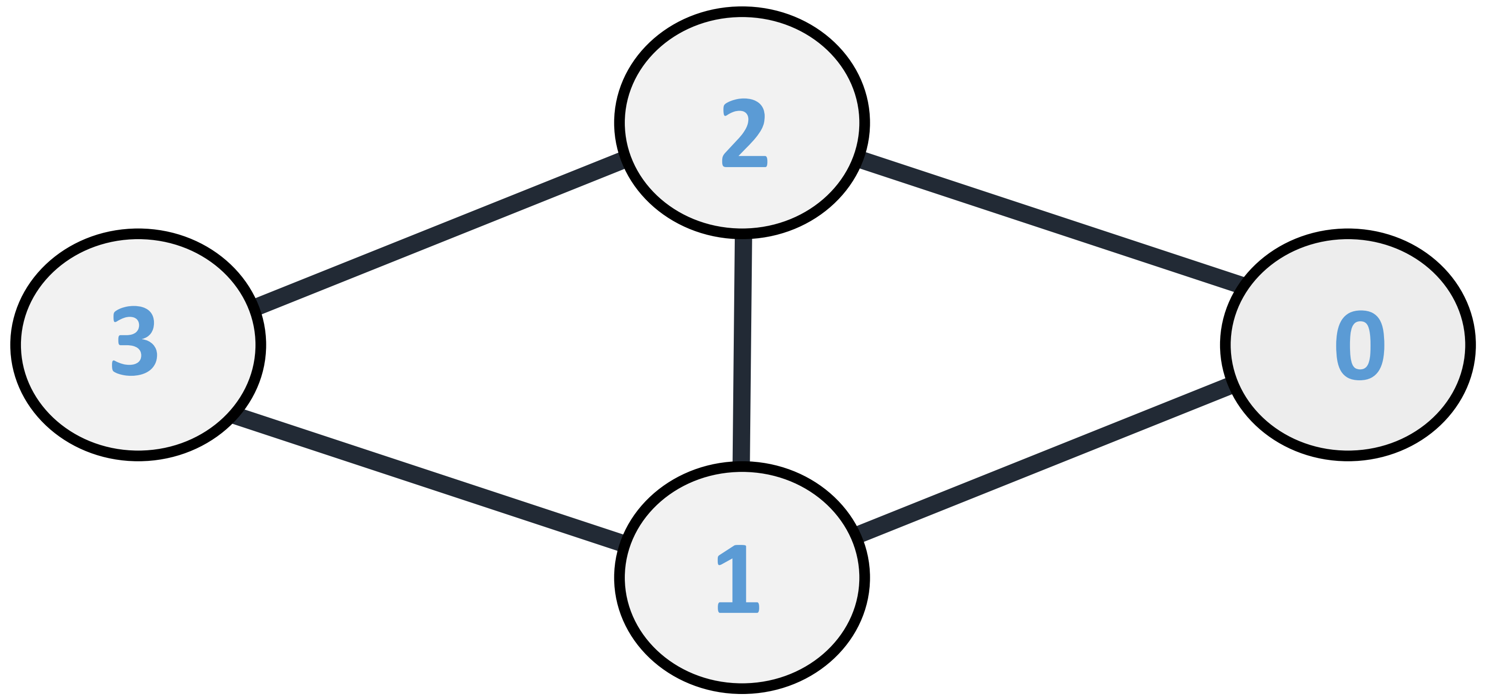
The NV Language

Idealized RIP



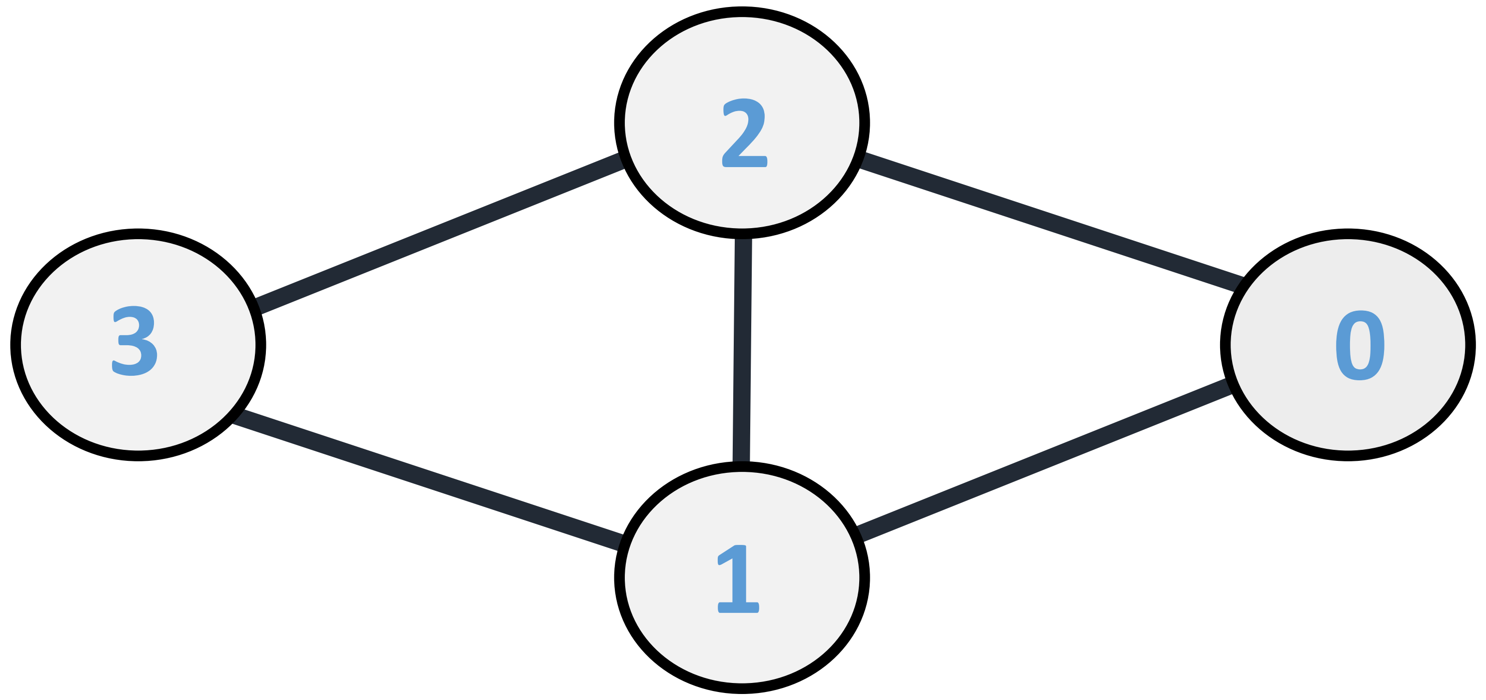
```
type attribute = option[int]
```

Idealized RIP



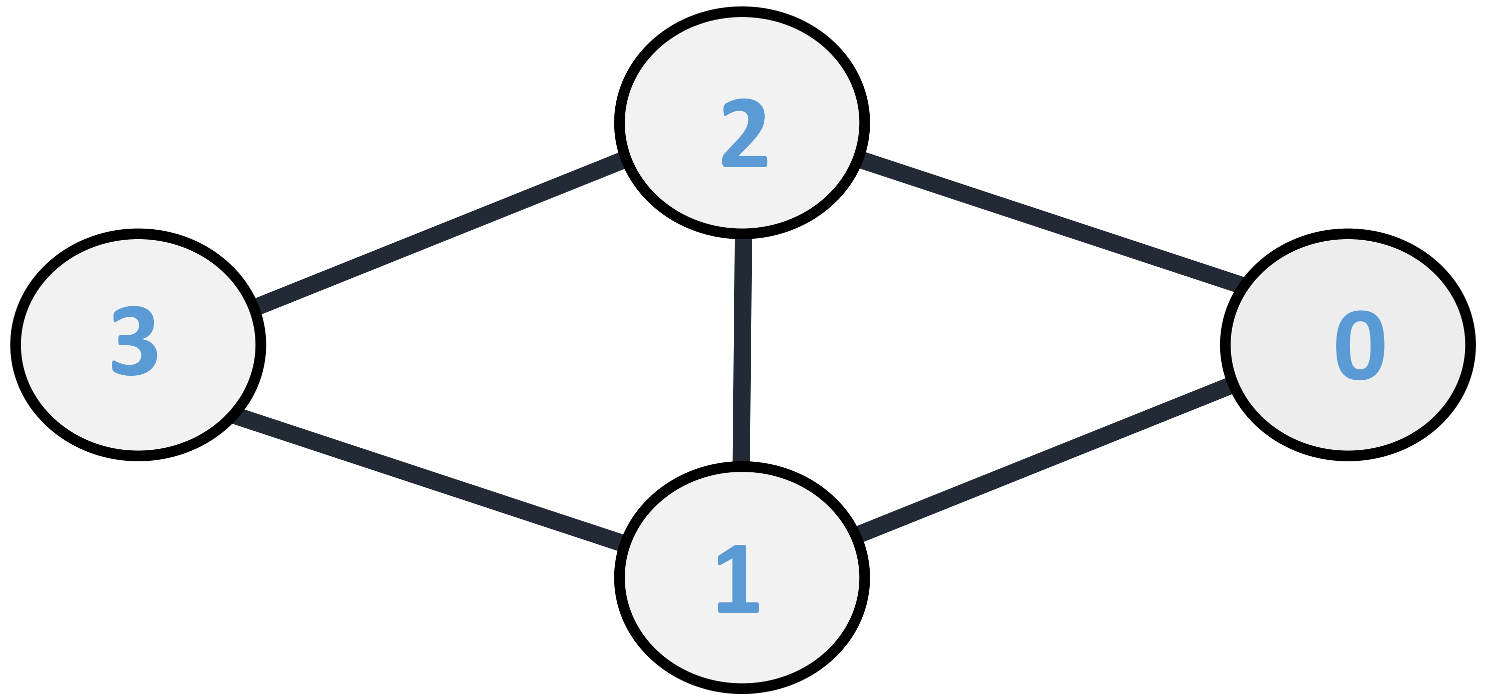
```
type attribute = option[int]

let nodes = 4

let edges = { 0=1; 0=2; 1=2; 1=3; 2=3; }
```

Idealized RIP



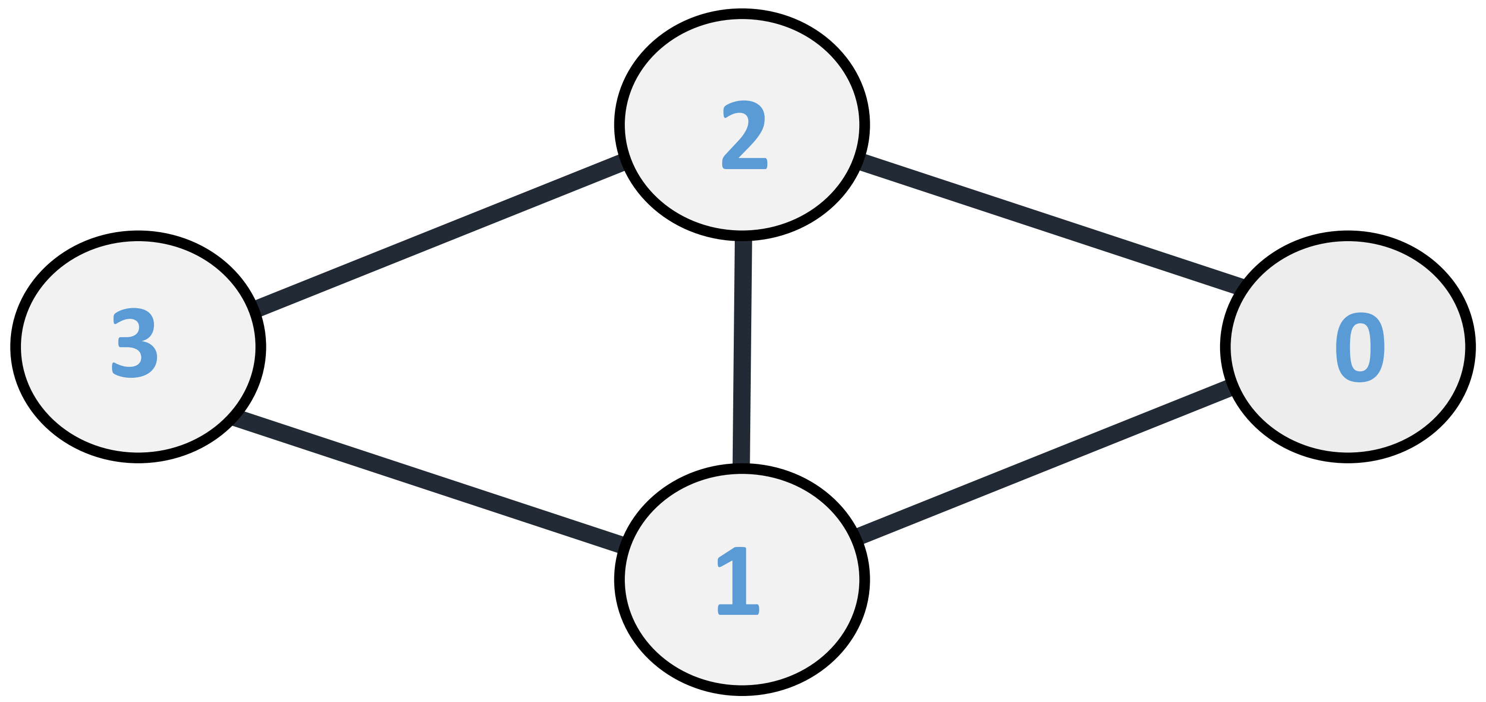
```
type attribute = option[int]

let nodes = 4

let edges = { 0=1; 0=2; 1=2; 1=3; 2=3; }

let trans edge x =
```

Idealized RIP



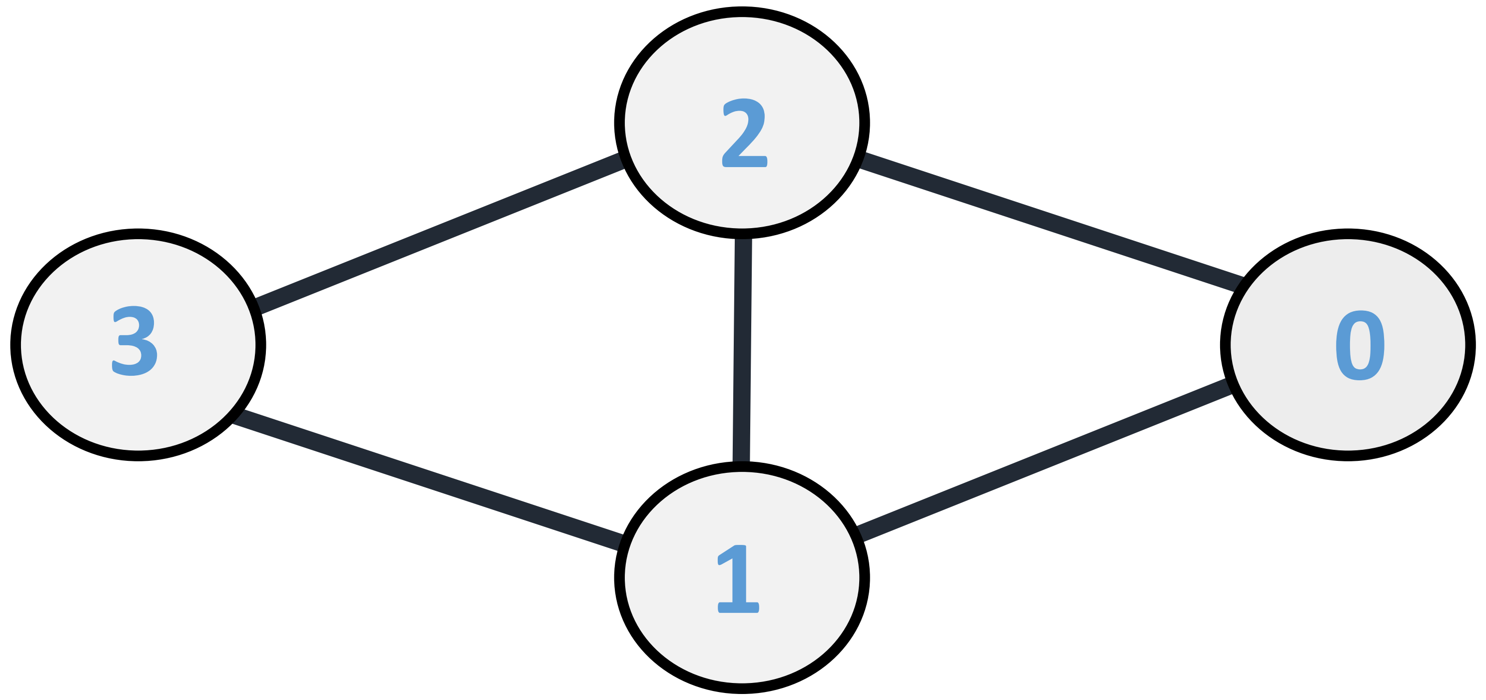
```
type attribute = option[int]

let nodes = 4

let edges = { 0=1; 0=2; 1=2; 1=3; 2=3; }

let trans edge x =
  match x with
  | None -> None
  | Some i -> Some (i+1)
```

Idealized RIP



```
type attribute = option[int]

let nodes = 4

let edges = { 0=1; 0=2; 1=2; 1=3; 2=3; }

let trans edge x =
  match x with
  | None -> None
  | Some i -> Some (i+1)

let merge node x y =
  match (x,y) with
  | (None, _) -> y
  | (_, None) -> x
  | (Some x’, Some y’) -> Some (min x’ y’)

let init node =
  if node = 0 then Some 0 else None
```

Adding Assertions

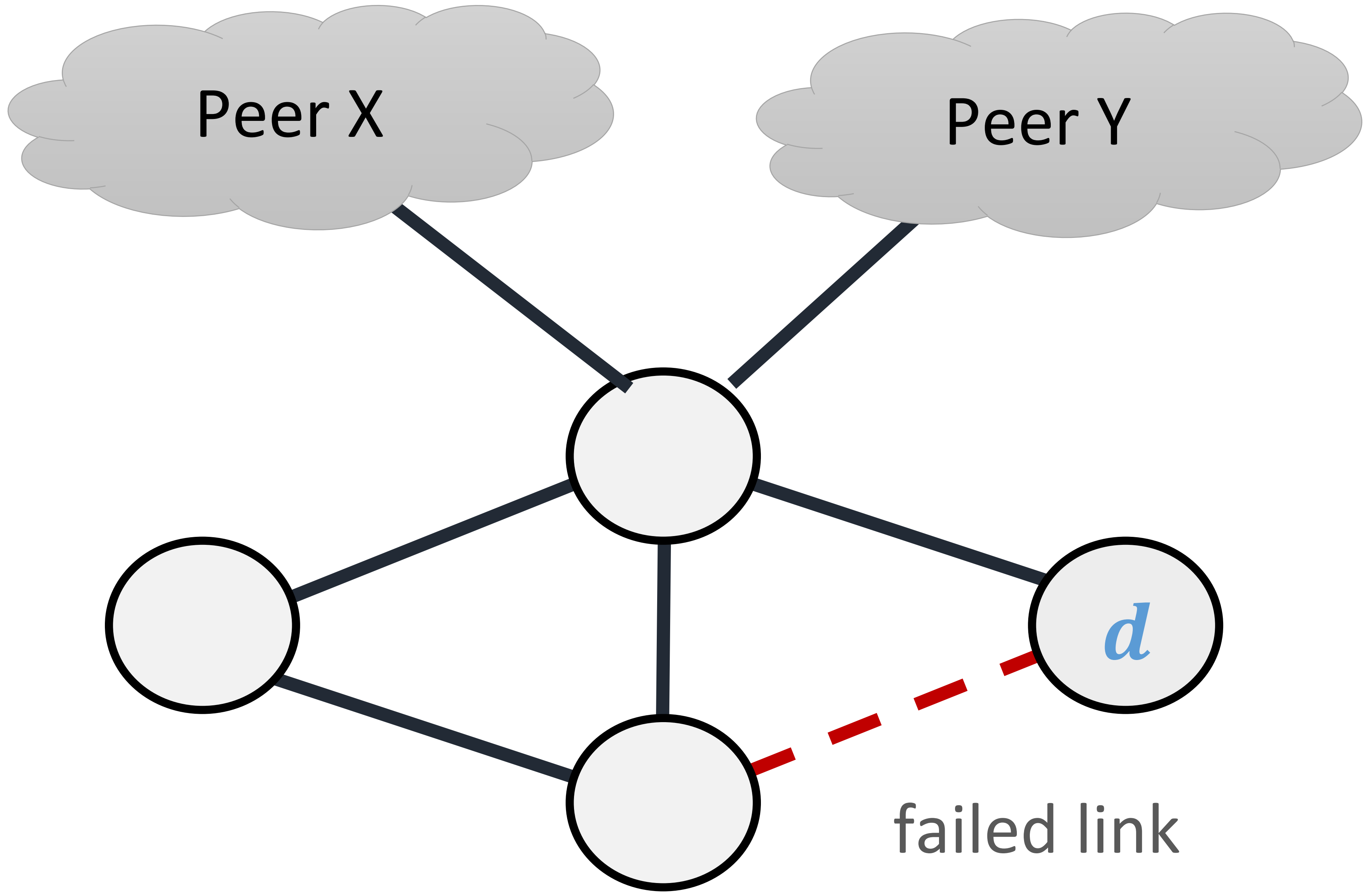


```
type attribute = option[int]

...


(* all nodes can reach the destination *)
let assert node sol =
  match sol with
    None -> false
  | Some x -> true
```

Assertion checking modes:

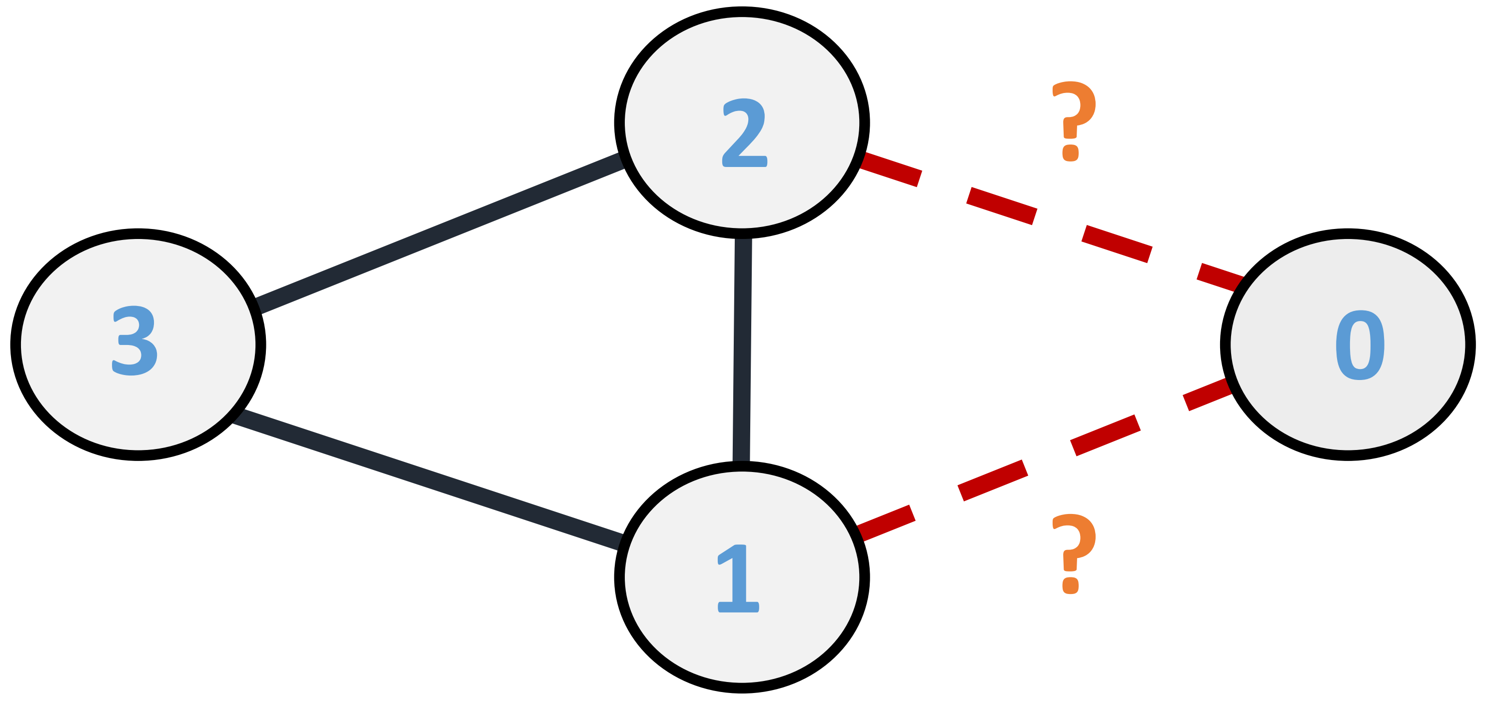
- SMT: Finds some solution that does not satisfy the assertion (or verifies all do)
- Simulation: Checks that an arbitrary solution satisfies the assertion (faster)

Managing Unknowns



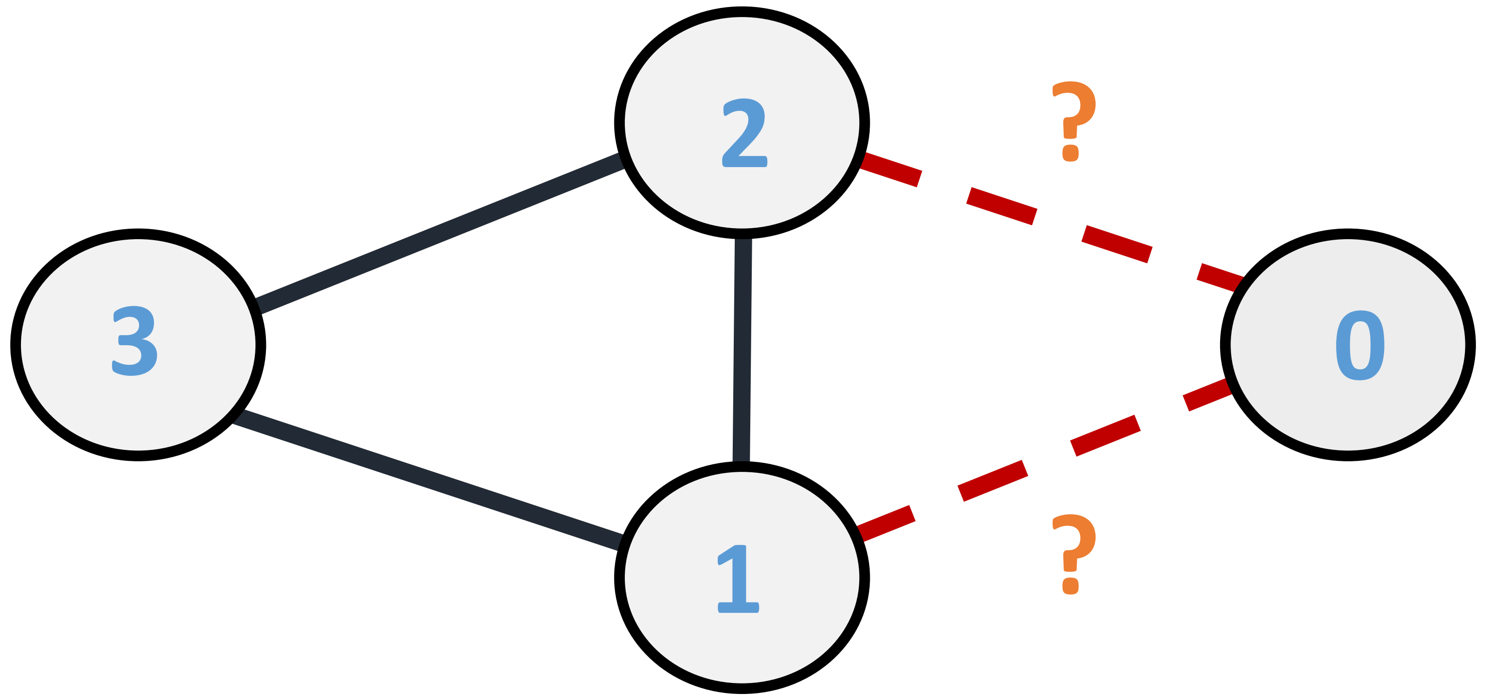
- Most networks are connected to the rest of the internet through peer networks. These peers may propagate arbitrary (well-formed) messages
- In large networks, many devices fail. Operators need to reason about network behavior in the presence of failures.
- In both cases, we need to model unknowns

Managing Unknowns: Link Failures



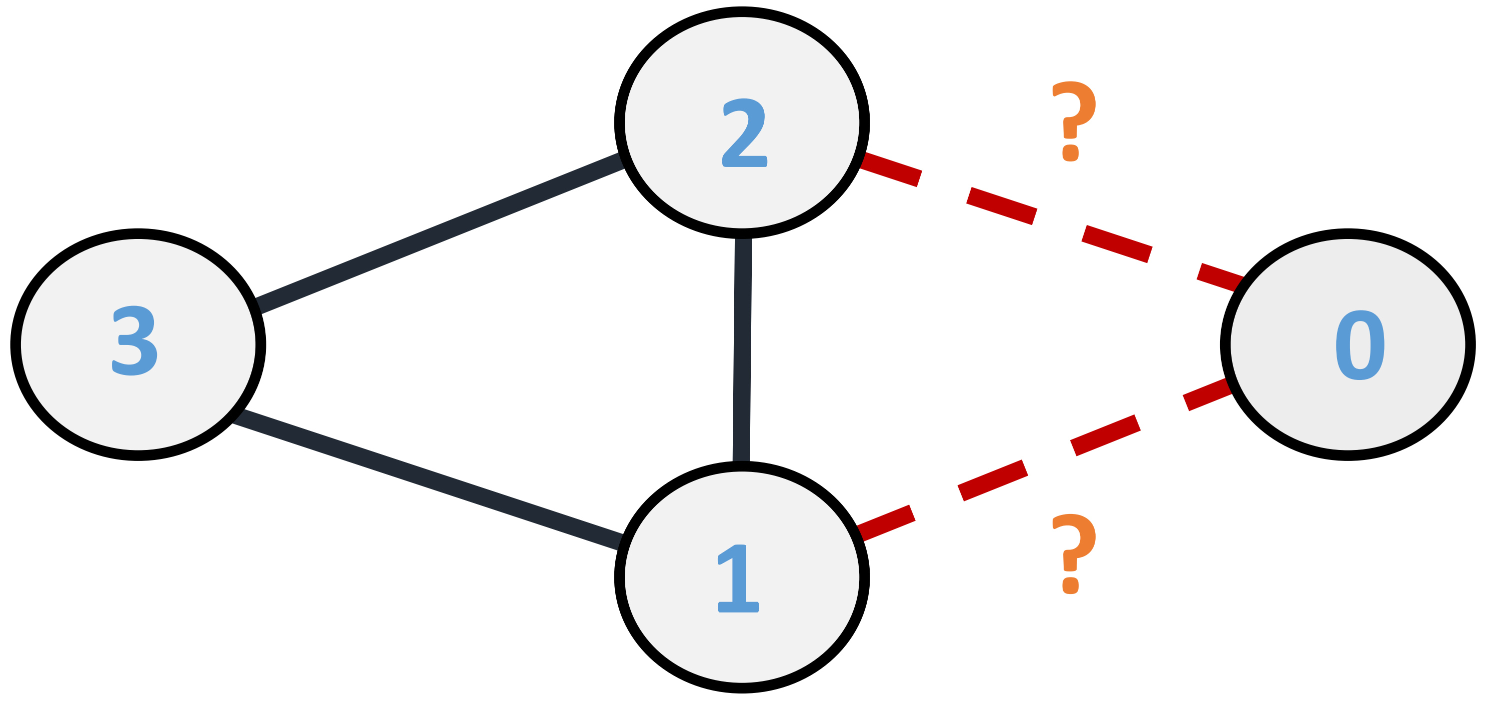
```
type attribute = option[int]

symbolic fail01 : bool
symbolic fail02 : bool
```

Managing Unknowns: Link Failures



```
type attribute = option[int]

symbolic fail01 : bool
symbolic fail02 : bool

require !(fail01 && fail02)
```

Managing Unknowns: Link Failures



```
type attribute = option[int]

symbolic fail01 : bool
symbolic fail02 : bool

require !(fail01 && fail02)

let trans edge x =
  if (edge = (0,1) && fail01)
  || (edge = (0,2) && fail02) then
     None
  else
     ...
```

More Realistic Protocols

```
type ospf =
  { ospfAd: int; weight: int; areaType: int; areaId: int; }

type bgp =
  { bgpAd: int; lp: int; aslen: int; comms:set[int]; origin:int}

type rib = {
    connected : option[int];
    static    : option[int];
    ospf      : option[ospf];
    bgp       : option[bgp];
    selected  : option[int];
}

type prefix = {ip:int32; len:int5}

type attribute = dict[prefix][rib]
```

Message Abstractions

```
type ospf = ...

type bgp = ...

type rib = ...

type prefix = ...

type attribute =
  dict[prefix][rib]
```

- **This is a lot of bits for each message. A simulator (or verifier) must process a lot of messages.**
- **For some properties, and many policies, we don’t need to keep track of all that information.**
- **Because the system is programmable, we can construct abstractions relatively easily.**
- **Not only can the abstract routing algorithms be simulated more efficiently. They can lead to new analysis ideas.**

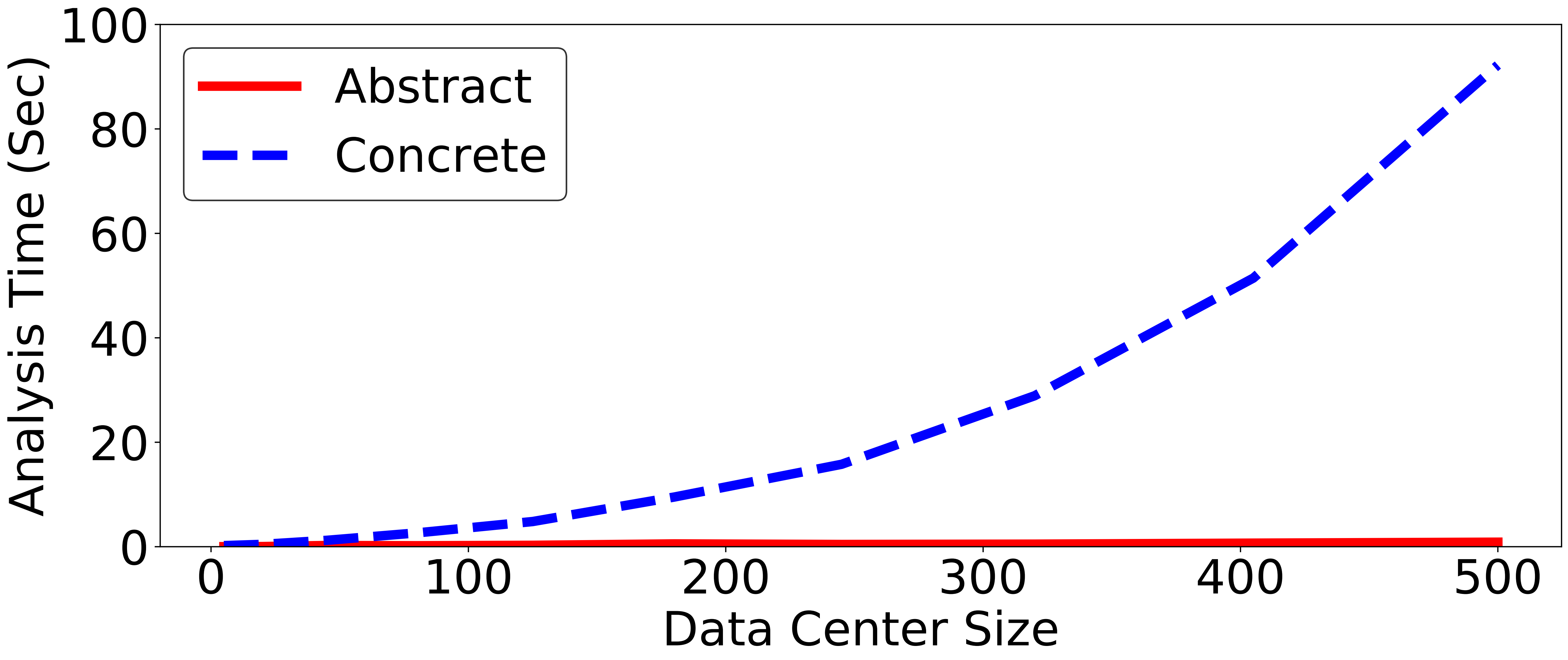
Message Abstractions

```
type bgp =
  { bgpAd. : int;
    lp     : int;
    aslen  : int;
    comms  : set[int];
    origin : int;
 }
```

```
type abstract_bgp = {
   comms   : set[int32];
   origin : int;
]
```

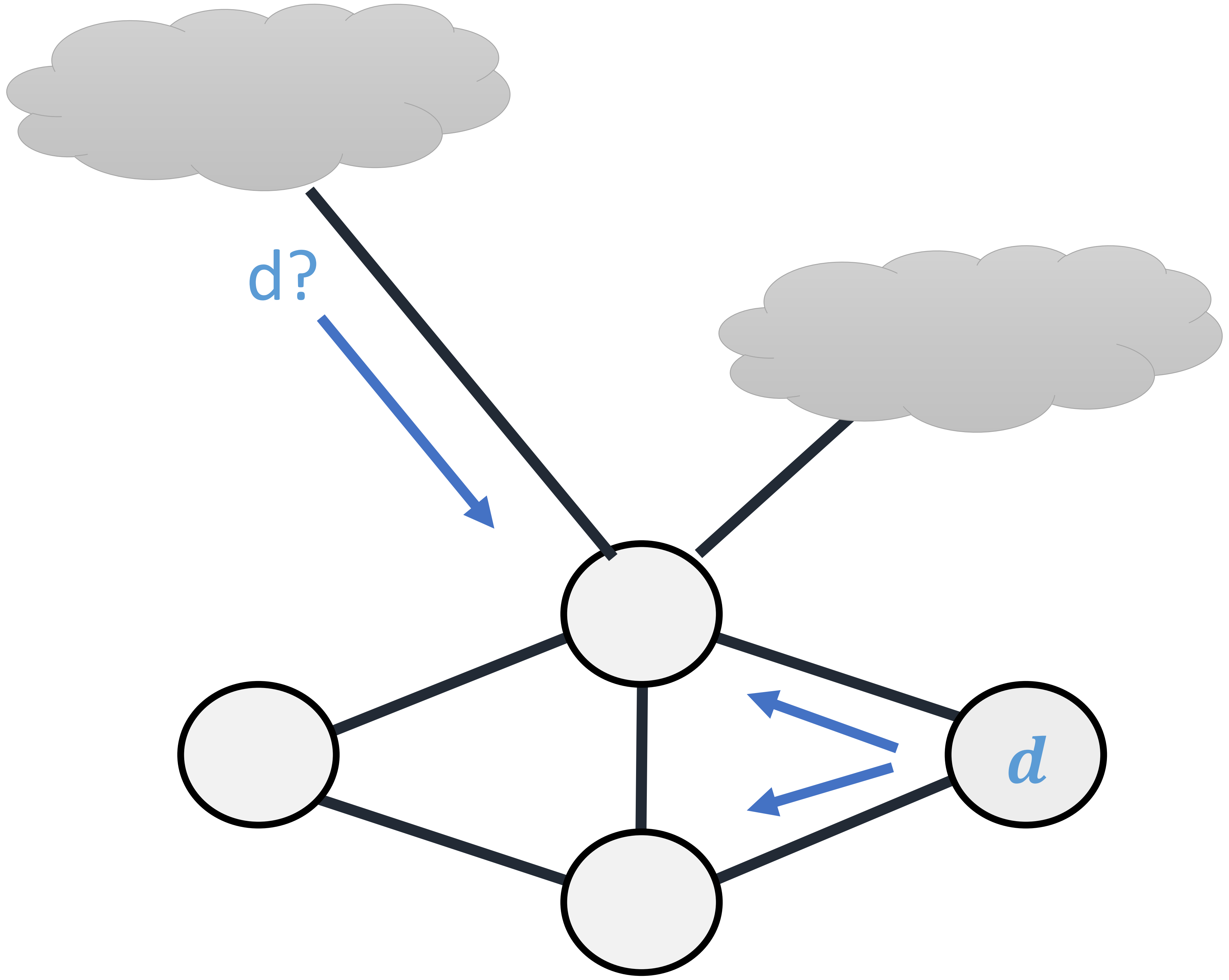
An effective abstraction for reachability
For many networks no reduction in precision;
asymptotically faster

Scaling Trends: Message Abstractions for Data Center Reachability



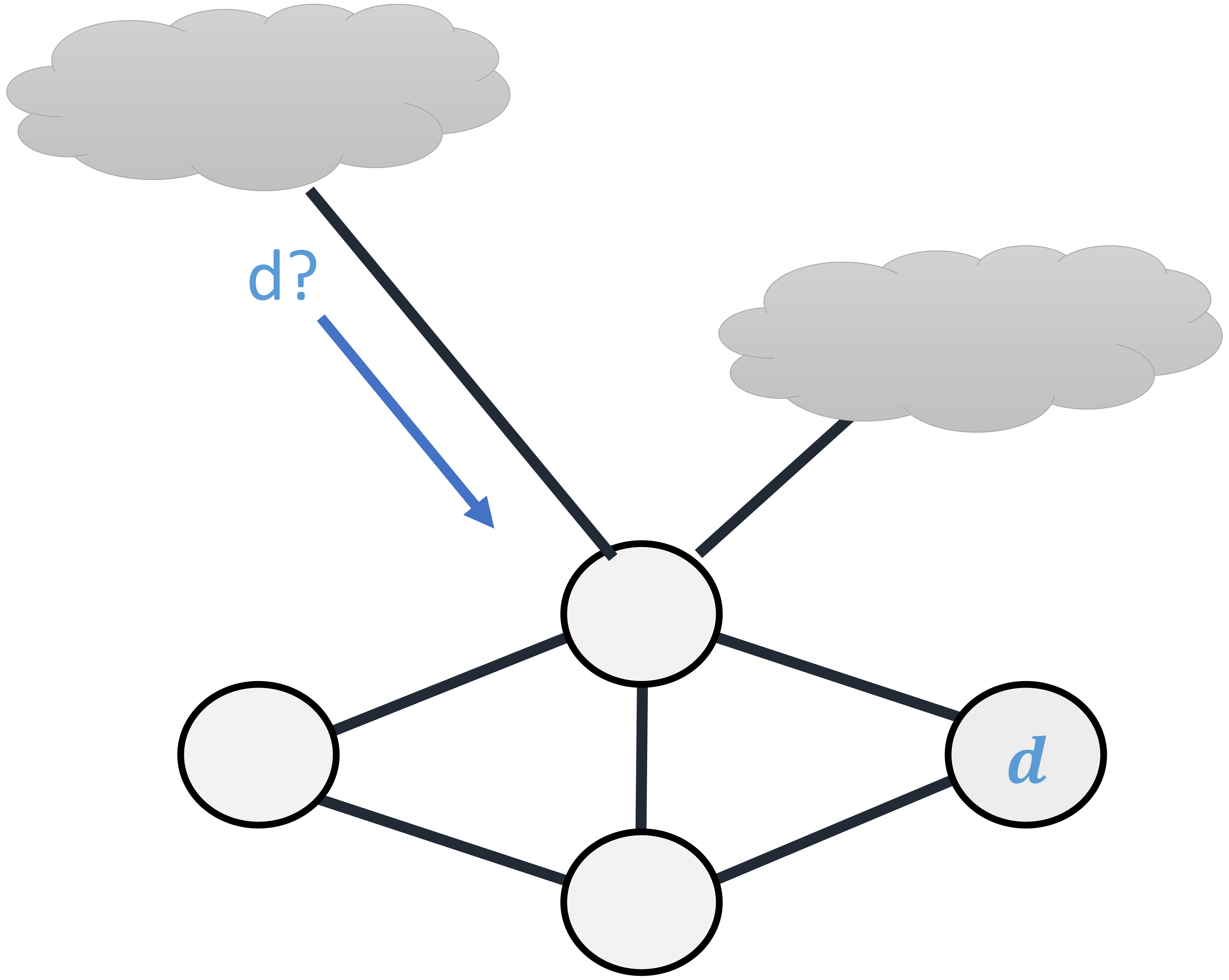
- **Concrete cost: O(tde)**
  - **t = time for 1 prefix**
  - **d = # of prefixes**
  - **e = # of edges**
  - **empirically: tde = n^2 * root(n)**
- **Abstract cost: O(te)**
  - **many messages now have the same value and can be processed at the same time**
  - **processing no longer depends on the # prefixes**
  - **empirically: te = n * root(n)**

New Analyses via Abstraction: BGP Hijacking Attacks



- **Can my peer networks hijack traffic destined for IP addresses that I own?**

New Analyses via Abstraction: BGP Hijacking Attacks



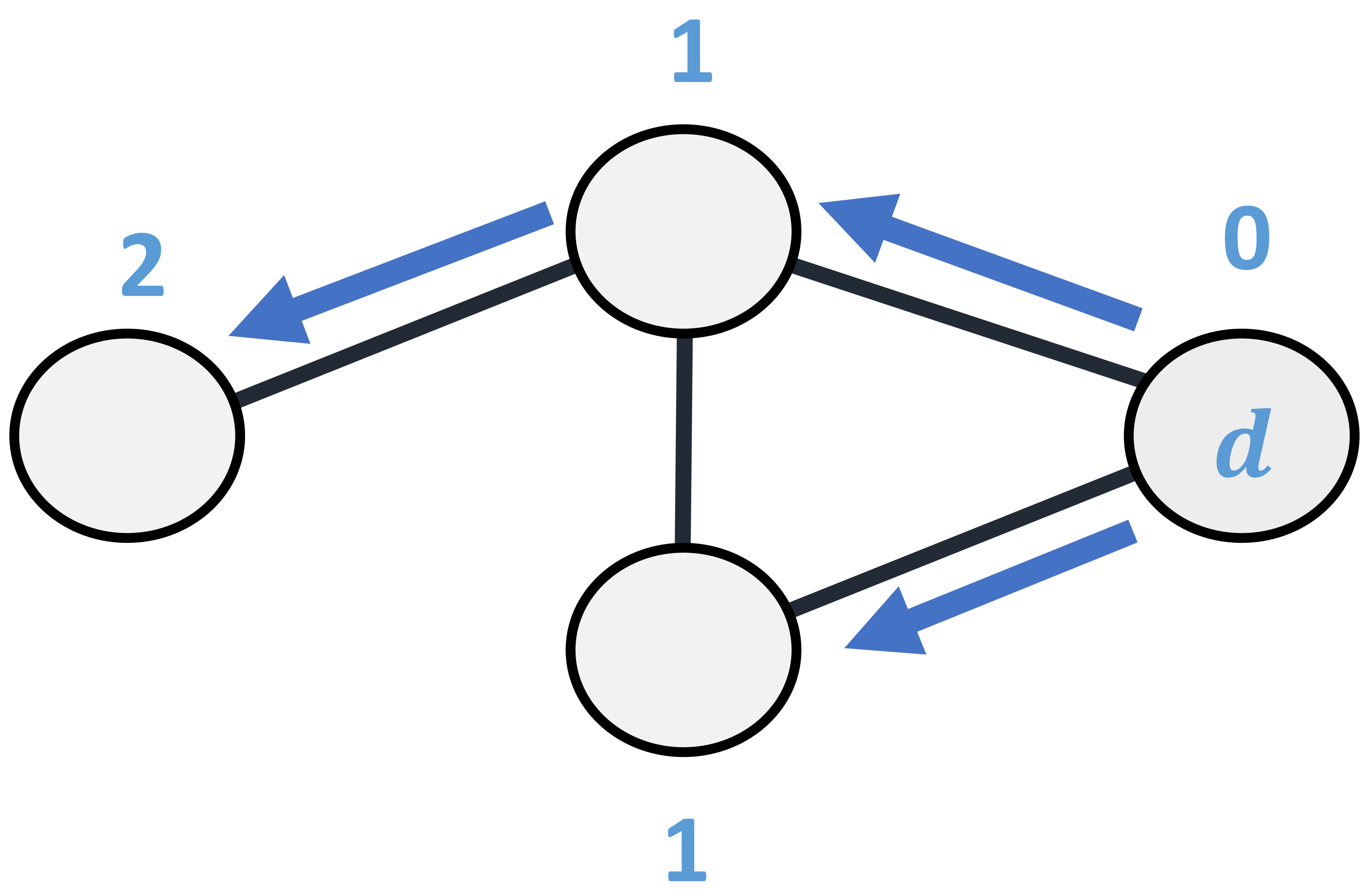
```
type origin =
   Internal | External

type abs_bgp = {
  comm : set[int32];
  origin : set[origin];
]
```

abstract
message
origins

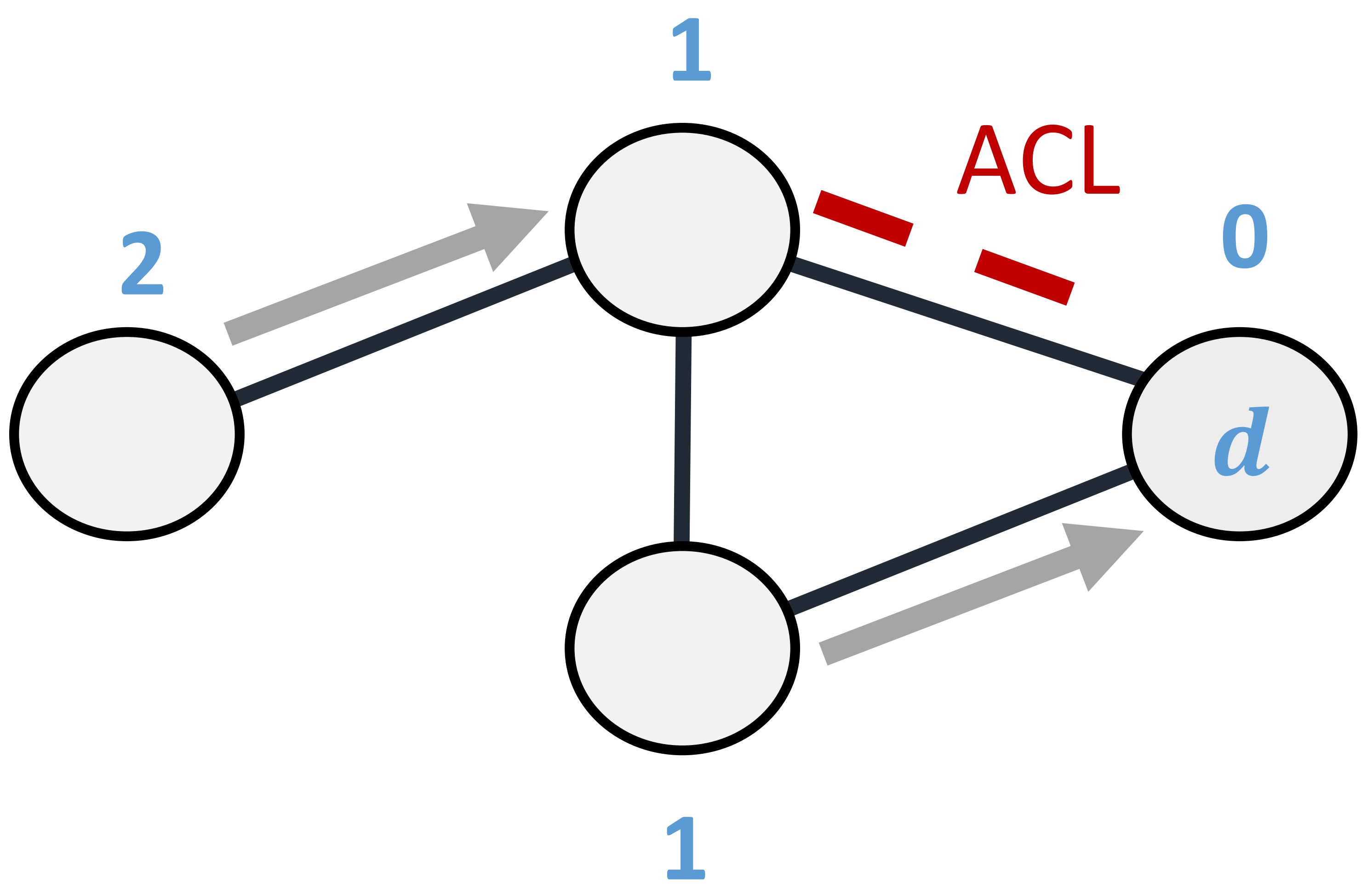
Questions/Problems/ToDos

(a subset!)

Adding Dataplane Facts

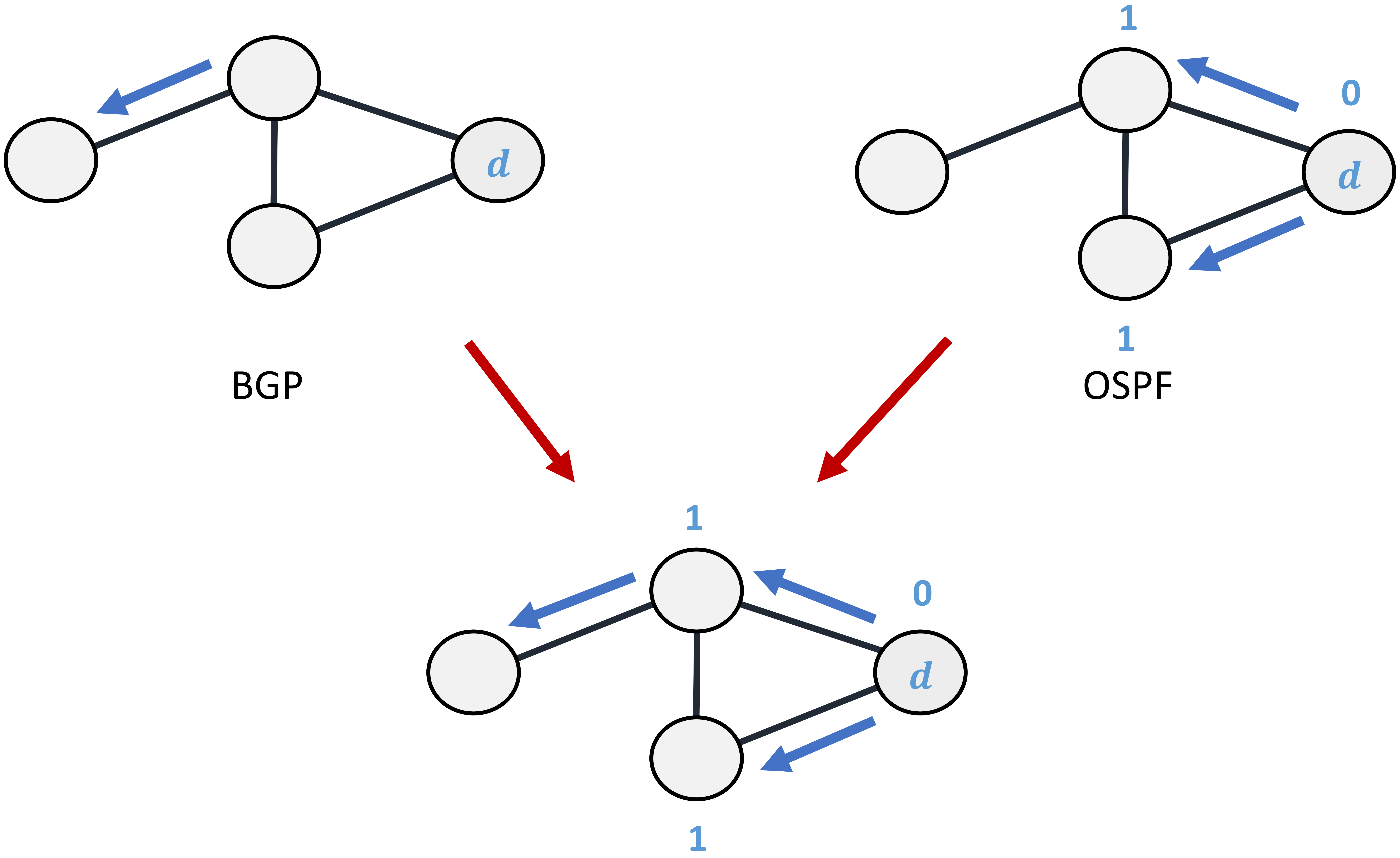


Control plane propagation of routes

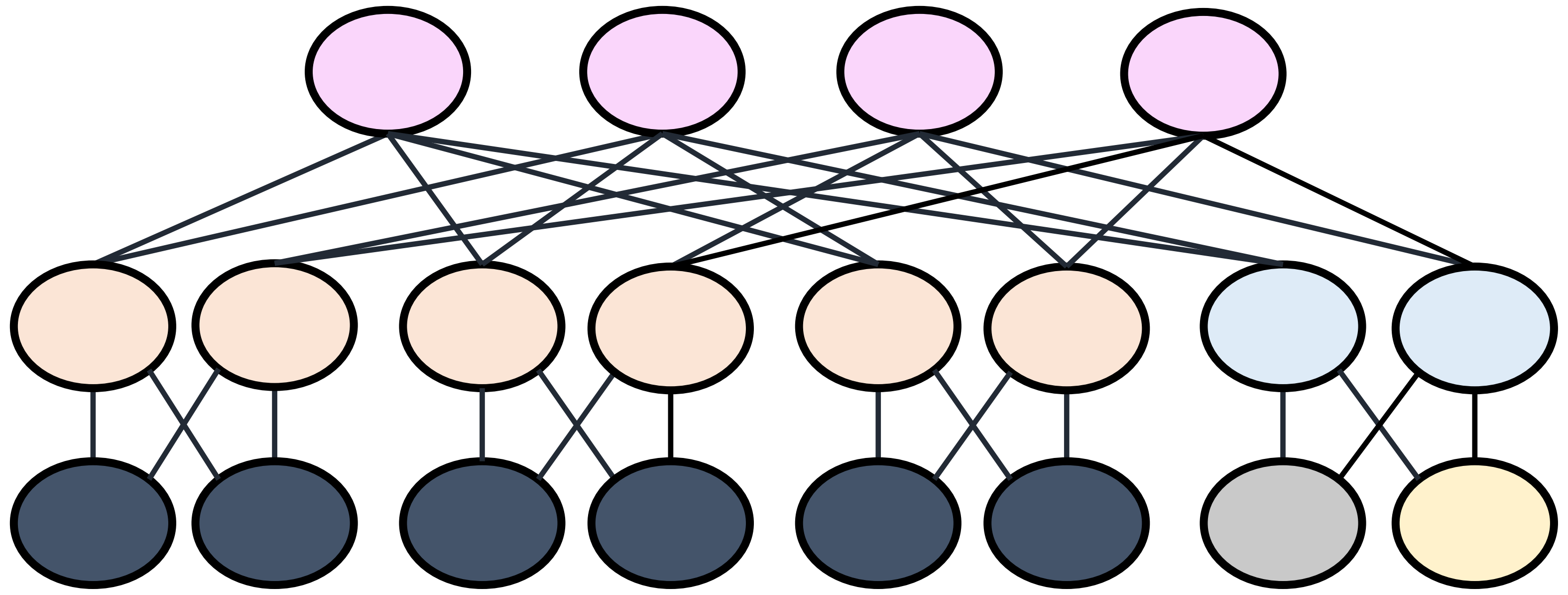


Data plane propagation of traffic

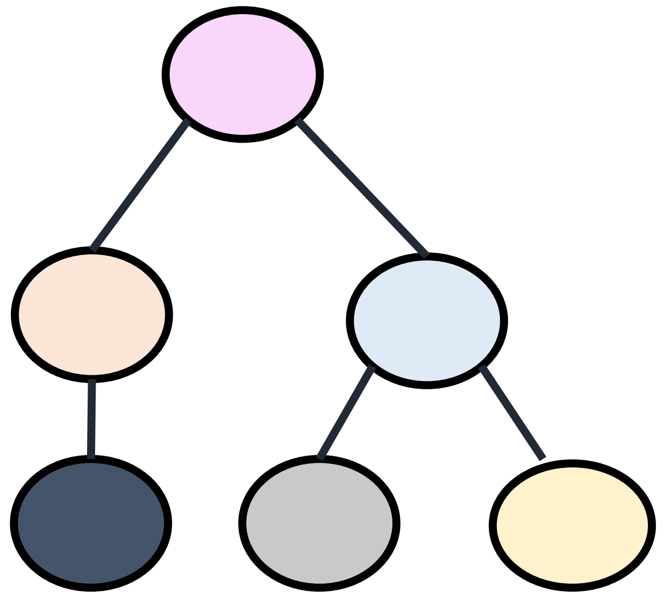
Composing Protocols



A meta-language for routing transformations?



program 1



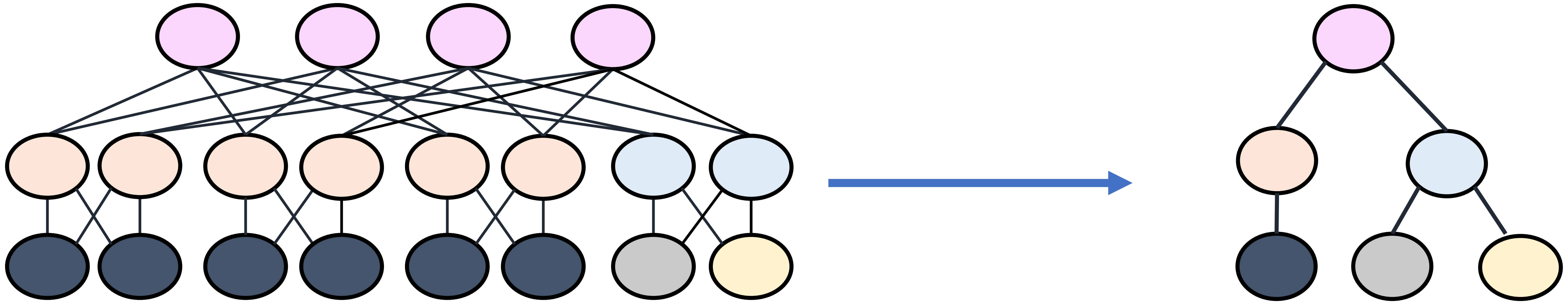
```
type bgp = {
  lp : int32;
  comm : set[int32];
  med : int32;
  rid : int32;
  as_len : int32;
  as_origin : int32;
}
```

program 2

```
type bgp = {
  comm : set[int32];
  as_origin : int32;
]
```

The lower level, compact calculus isn’t always a win

- **One transformation requires identifying transfer functions that are “the same”**



- **But they aren’t actually *ever* exactly the same in BGP because it adds the current node identity to the AS path**
- **But we can show they are “close enough” in this special case**
- **In Batfish, AS path extension is implicit; in NV, explicit and gets in the way of identifying “similar” transfer functions**

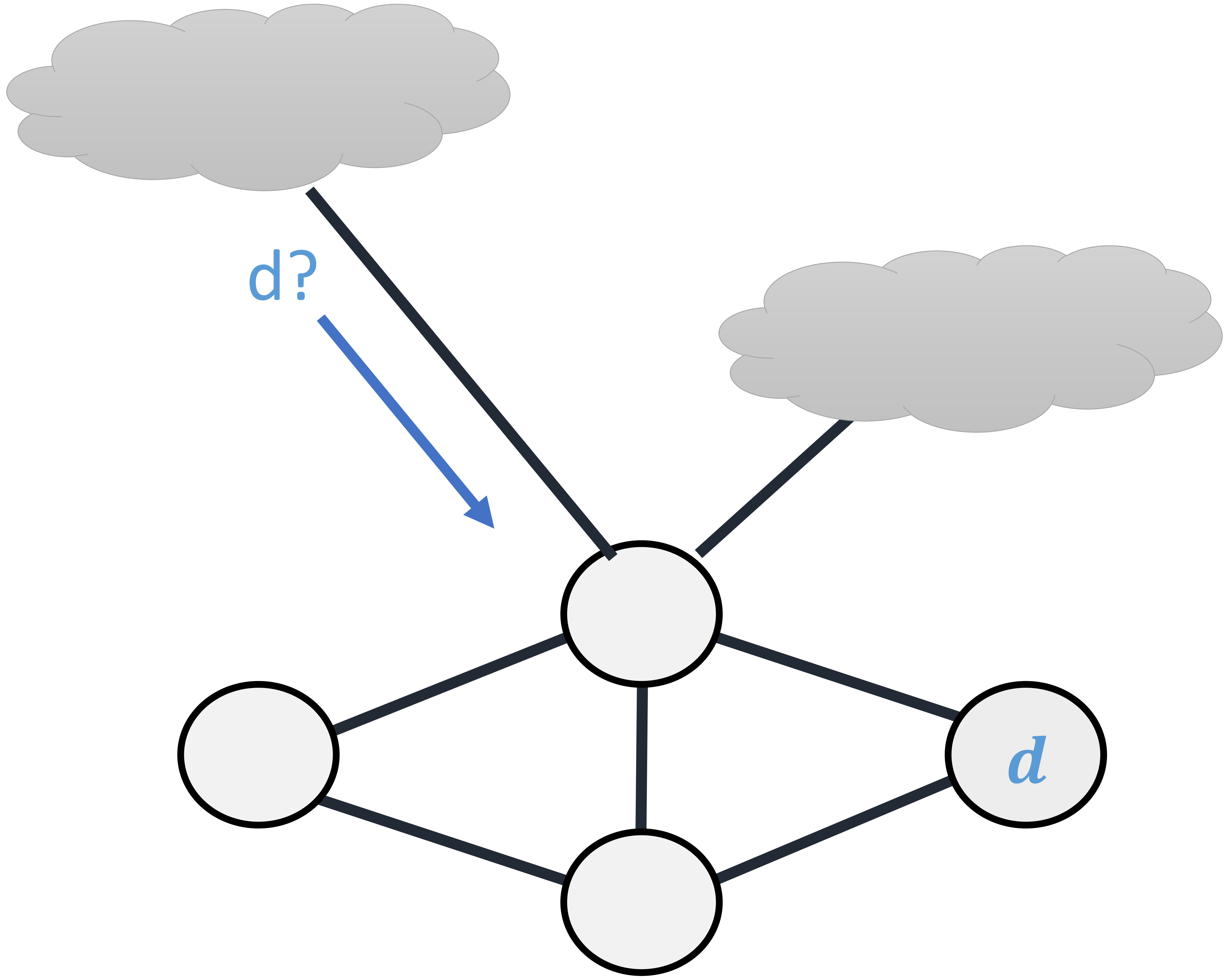
Solutions?

- Solution 1 (the hack): Add annotations during translation that says “this adding to the AS-path; ignore me”
- Solution 2 (better?): Add modules to the NV language so you can encapsulate the AS-path operations in a module. The module encapsulates the differences.
- Other thoughts?

Solutions?

- **Solution 1 (the hack): Add annotations during translation that says “this adding to the AS-path; ignore me”**
- **Solution 2 (better?): Add modules to the NV language so you can encapsulate the AS-path operations in a module. The module encapsulates the differences.**
- **Other thoughts?**

Final Goal



- I hope functional programming can actually help us understand complex network protocols
- I'm looking at you, iBGP!