Consensus Routing: The Internet as a Distributed System

John, J.P. *et al. Proc. of the 5th USENIX Conf. on NSDI '08*, pp. 301-364, 2008

Presented by Jack Kosaian

Traditional BGP: responsiveness over consistency

Receive update from neighbor

Determine new best route

Update forwarding table

Propagate updates to neighbors

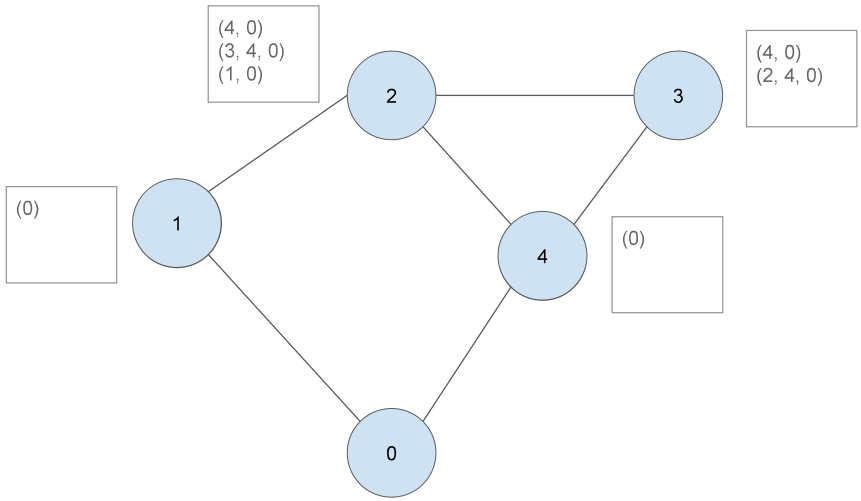
Issues with favoring responsiveness

Routing loops and blackholes

Loops account for 90% of all packet loss

What causes routing loops and blackholes?

1. Link failures



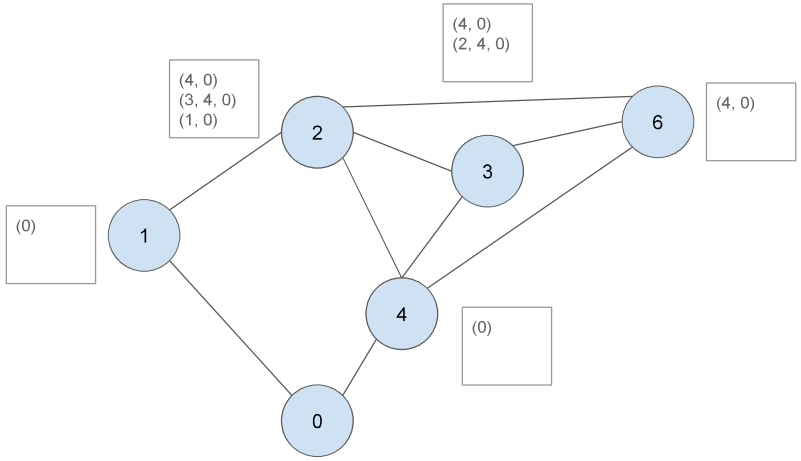
Issues with favoring responsiveness

Routing loops and blackholes

Loops account for 90% of all packet loss

What causes routing loops and blackholes?

1. Link failures
2. Policy changes



Issues with favoring responsiveness

Routing loops and blackholes

Loops account for 90% of all packet loss

What causes routing loops and blackholes?

1. Link failures
2. Policy changes

Leads to larger problems in Internet

1. inconsistency → unpredictability
2. unpredictability → vulnerability
3. vulnerability → distrust

Goal

“A simple, practical routing protocol that allows general routing policies and achieves high availability”

Consensus Routing

Insights:

- Consistency is a safety property
  - Forward packets along stable routes

- Responsiveness is a liveness property
  - React quickly to failures and policy changes

Separating safety and liveness improves overall availability

Consensus Routing

1. Stable mode
   - Routes adopted only after all dependent routers agree

2. Transient mode
   - Accommodate link failures and instability

Note: Following description assumes one router per AS for simplicity. This can be relaxed.

Stable Mode

Coordination proceeds in epochs

Guarantee that adopted routes are consistent
- If A routes (B, C, D), then B routes (C, D)

After coordination, ASes have set of Stable Forwarding Tables (SFT)

Forward along SFT's

If encounter a failed link or unavailable route, enter transient mode

Stable Mode

During each epoch
1. Routers log updates without changing SFT
2. Snapshot taken to determine completed updates
3. Consistent set of incomplete updates and valid ASes determined
4. Set of incomplete updates and valid ASes flooded to all ASes
5. Routers make updates to SFT
6. Routers forward using new SFT in next epoch
7. Routers discard outdated SFT's on epoch completion

Logging and Processing Updates

Router State: for each destination prefix

- Route from each neighbor
- Best route
- Route advertised to each neighbor
- History of received and selected updates (H_A)
- Set of incomplete updates (I_A)
- Stable next hop (part of SFT)

Triggers

Pair of <AS number, unique ID>

Correspond to a route that is to be withdrawn

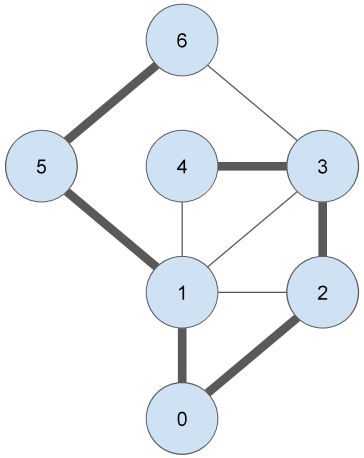
Propagated when a neighbor has heard route being withdrawn (route update)

Track whether withdrawal is complete

- Have all dependent ASes processed the withdrawal?

Processing an update (*B, r, t*)

1. Add *t* to I_A
2. Process update: determine the new best route (*new*) to prefix *p*
3. Add update (*t*, *r*) to head of History ($H_A[p]$.push_front*(<R, (t, r)>)*)
   a. *old.next_hop* != *B* and *new.next_hop* != *B*
      i. Do nothing; best route has not changed
   b. *old.next_hop* != *B* and *new.next_hop* == *B*
      i. Best route changes to go through *B* as next hop
      ii. Let neighbors know of our new route; Send *new* along with new trigger *t'*
      iii. $H_A[p]$.push_front(<*S*, *(t', new)*>)
   c. *old.next_hop* == *B*
      i. Best route will change
      ii. Let neighbors know of new route; Send *new* along with unchanged trigger *t*
      iii. $H_A[p]$.push_front(<*S*, *(t, new)*>)
4. Remove *t* from I_A

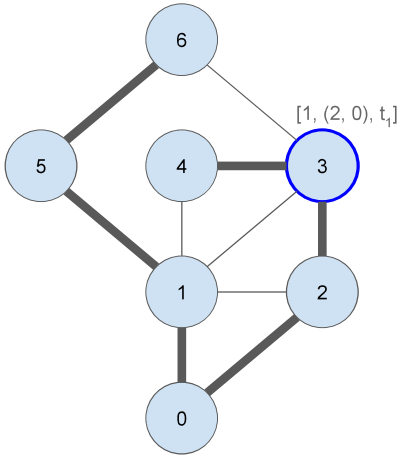


What should AS3 do?

I_3:

$H_3[0]$:

[1, (2, 0), t_1]
B r t



Processing an update (*B, r, t*)

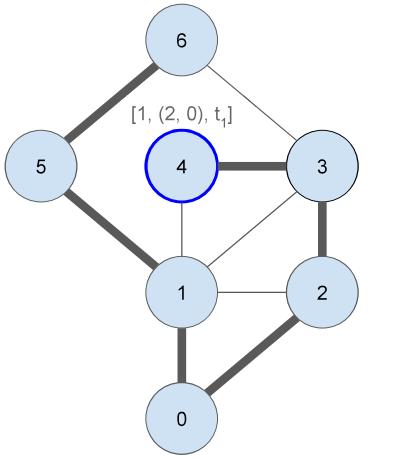
1. Add *t* to I_A
2. Process update: determine the new best route (*new*) to prefix *p*
3. Add update (*t*, *r*) to head of History ($H_A[p]$.push_front(<*R, (t, r)*>))
   a. *old.next_hop* != *B* and *new.next_hop* != *B*
      i. Do nothing; best route has not changed
   b. *old.next_hop* != *B* and *new.next_hop* == *B*
      i. Best route changes to go through *B* as next hop
      ii. Let neighbors know of our new route; Send *new* along with new trigger *t'*
      iii. $H_A[p]$.push_front(<*S, (t', new)*>)
   c. *old.next_hop* == *B*
      i. Best route will change
      ii. Let neighbors know of new route; Send *new* along with unchanged trigger *t*
      iii. $H_A[p]$.push_front(<*S, (t, new)*>)
4. Remove *t* from I_A

What should AS4 do?

I_4:

$H_4[0]$:

[1, (2, 0), t_1]
B r t



Processing an update (*B, r, t*)

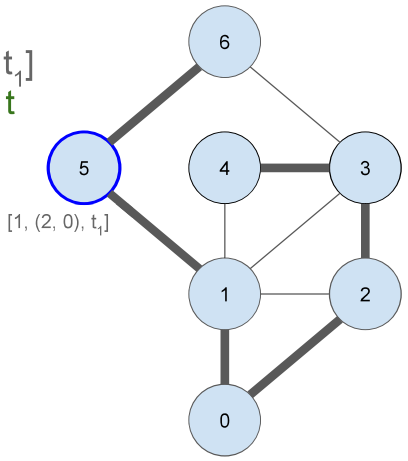
1. Add *t* to I_A
2. Process update: determine the new best route (*new*) to prefix *p*
3. Add update (*t*, *r*) to head of History ($H_A[p]$.push_front(<*R, (t, r)*>))
   a. *old.next_hop* != *B* and *new.next_hop* != *B*
      i. Do nothing; best route has not changed
   b. *old.next_hop* != *B* and *new.next_hop* == *B*
      i. Best route changes to go through *B* as next hop
      ii. Let neighbors know of our new route; Send *new* along with new trigger *t'*
      iii. $H_A[p]$.push_front(<*S, (t', new)*>)
   c. *old.next_hop* == *B*
      i. Best route will change
      ii. Let neighbors know of new route; Send *new* along with unchanged trigger *t*
      iii. $H_A[p]$.push_front(<*S, (t, new)*>)
4. Remove *t* from I_A

What should AS5 do?

I_5:

$H_5[0]$:

[1, (2, 0), t_1]
B r t



Processing an update (*B, r, t*)

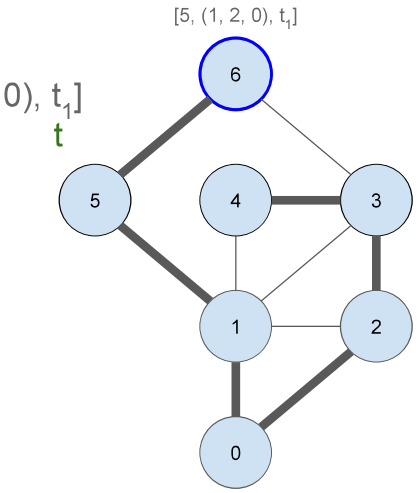
1. Add *t* to I_A
2. Process update: determine the new best route (*new*) to prefix *p*
3. Add update (*t*, *r*) to head of History ($H_A[p]$.push_front(<*R, (t, r)*>))
   a. *old.next_hop* != *B* and *new.next_hop* != *B*
      i. Do nothing; best route has not changed
   b. *old.next_hop* != *B* and *new.next_hop* == *B*
      i. Best route changes to go through *B* as next hop
      ii. Let neighbors know of our new route; Send *new* along with new trigger *t'*
      iii. $H_A[p]$.push_front(<*S, (t', new)*>)
   c. *old.next_hop == B*
      i. Best route will change
      ii. Let neighbors know of new route; Send *new* along with unchanged trigger *t*
      iii. $H_A[p]$.push_front(<*S, (t, new)*>)
4. Remove *t* from I_A

What should AS6 do?

I_6:

$H_6[0]$:

[5, (1, 2, 0), t_1]
B r t



Processing an update (*B, r, t*)

1. Add *t* to I_A
2. Process update: determine the new best route (*new*) to prefix *p*
3. Add update (*t*, *r*) to head of History ($H_A[p]$.push_front(<*R, (t, r)*>))
   a. *old.next_hop* != *B* and *new.next_hop* != *B*
      i. Do nothing; best route has not changed
   b. *old.next_hop* != *B* and *new.next_hop* == *B*
      i. Best route changes to go through *B* as next hop
      ii. Let neighbors know of our new route; Send *new* along with new trigger *t'*
      iii. $H_A[p]$.push_front(<*S, (t', new)*>)
   c. *old.next_hop == B*
      i. Best route will change
      ii. Let neighbors know of new route; Send *new* along with unchanged trigger *t*
      iii. $H_A[p]$.push_front(<*S, (t, new)*>)
4. Remove *t* from I_A

Distributed Snapshot

Goal: Get history (H^s) and incomplete updates (I^s) from each AS

What makes an update incomplete?
- Update is currently being processed (in I_A)
- Update is waiting on MRAI timer (in A's output queue)
- Update is in transit from neighbor

How do we check for updates in transit?
- Receive marker: start of snapshot
- Log updates received from neighbors (except one that sent marker)
- Send marker to all neighbors (except one that sent marker)
- Stop logging updates on interface when receive marker from that neighbor
- Add all logged triggers to I^s

Where are we now?

Each router has history (H^S_A) and incomplete triggers (I^S_A) from last epoch

Need to aggregate these and agree on consistent routes

Frontier Computation

Each router sends H^S_A and I^S_A to consolidators

Consolidators propagate snapshot reports

Run consensus to agree on S, set of ASes participating in snapshot

Determine global set of incomplete triggers I
- Key insights:
  - If a trigger *t* is incomplete, any updates following *t* could be unstable
  - If *t* is in I^S_A, add it to I all following it in H^S_A to I

Flood I and S to all ASes

SFT Updates

Use I and S to construct SFT for next epoch

Start with existing SFT

For each destination *p*
- Find latest "S" update *u* in $H_A[p]$ s.t. neither trigger or any before in I
- Drop all records before *u* from $H_A[p]$
- If route in *u* contains an AS not in S, mark *null* in SFT
- Else adopt route in *u* as new route to *p* in SFT

Switching Epochs

Keep SFT's from k^{th} and $(k + 1)^{th}$ epochs

If have completed $(k + 1)^{th}$ SFT, forward using new route

What if some upstream router hasn't finished $(k + 1)^{th}$ SFT?
- Set bit in packet
- Forward using k^{th} SFT from there on-out

Once $(k + 2)^{th}$ SFT completed, discard k^{th} SFT

Transient Mode

Used when a stable route is not available

When might this be the case?
- Link failure to next hop
- Best route involved AS that didn't make snapshot last epoch

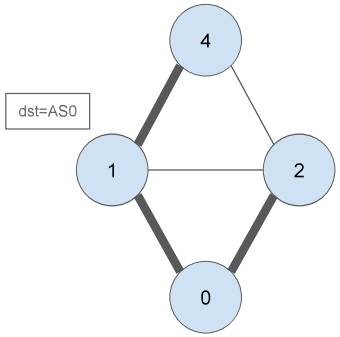
Utilize heuristics to keep traffic moving forward
- Deflection
- Detour routes
- Backup routes

Deflection

Route through a neighbor that has stable route

No neighbor has stable route? Backtrack
- Send packet to previous AS, have them deal with it

Deflection



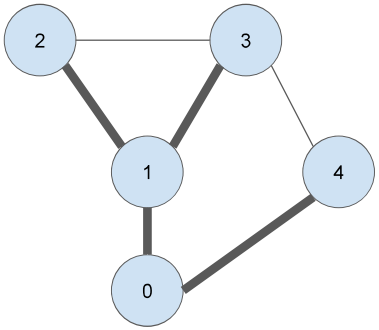
Detour Routes

No neighbors have available stable routes to destination

Hand off responsibility to another neighbor

Idea:
- Neighbor might have different view of available routes

Detour Routes



Backup Routes

Set up routes that are designated to be backups

Advertise backup routes to neighbors

If a neighbor encounters unavailability, no other neighbors have stable paths
- Route through advertised backup route

Evaluation

Questions to be answered:
- How effective is consensus routing at maintaining availability?
- What is the overhead of running consensus routing?

Evaluation techniques
- Simulation
- PlanetLab experiments
- XORP prototype development

Simulator

Use AS-level topology obtained from CAIDA
- Business relationships of links inferred

Match economic incentives for export and selection policies
- "Valley-free" export
- Favor customer routes

Compare consensus routing to BGP in presence of
- Link failures
- Traffic engineering

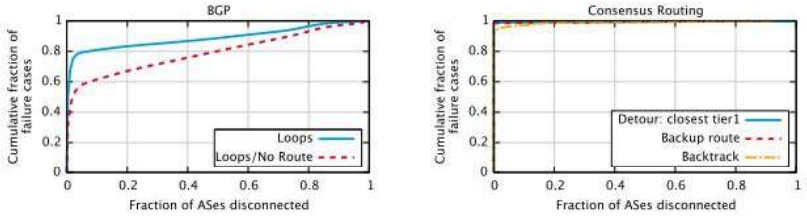
Link Failures

Many rounds each with one link failure

Fail routes at multi-homed stub ASes
- Ensures that route is always available to AS

Simulate both BGP and consensus routing

Link Failures



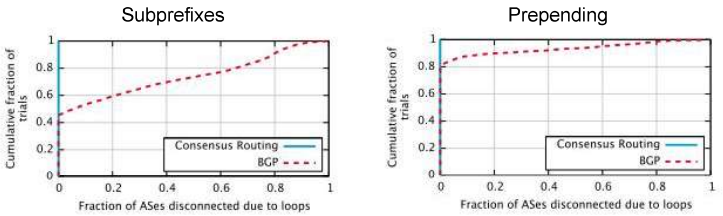
BGP: 13% of failures cause ~50% disconnectivity

Traffic Engineering

How do BGP and consensus routing respond to traffic engineering?

Two simulations:
- Advertise and withdraw subprefixes
  - Selectively withdraw subprefix advertisements from some providers
- Path prepending

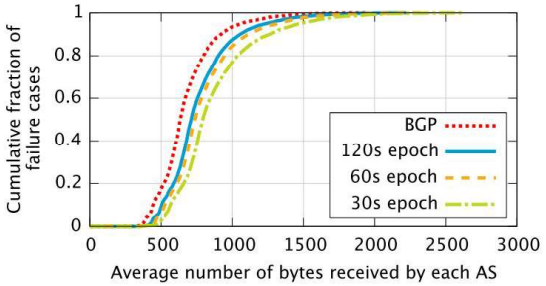
Traffic Engineering



BGP
- Subprefixes: > 55% of trials resulted in connection loss due to loops
- Prepending: ~20% of trials resulted in connection loss due to loops

Consensus Routing: no loops

Volume of Control Traffic



BGP has large messages to begin with

Overhead

Questions:
- How much additional traffic is involved in consensus routing?

Overhead

Questions:
- How much additional traffic is involved in consensus routing?
- How long does consensus take?

Cost of consensus

Number of nodes	Time when first node learns value	Time when last node learns value
9	434 ms	490 ms
18	485 ms	1355 ms
27	590 ms	1723 ms

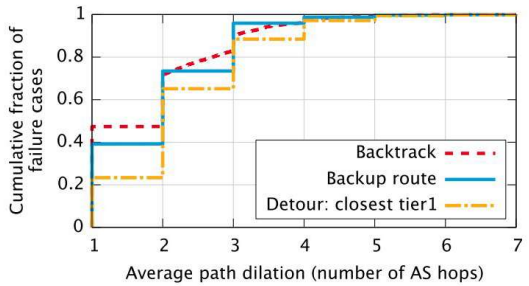
As number of nodes in consensus increases, total latency of consensus increases

Overhead

Questions:

- How much additional traffic is involved in consensus routing?
- How long does consensus take?
- How much longer are paths when avoiding failure?

Path dilation



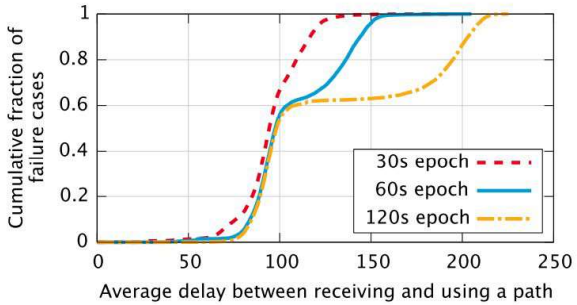
Dilation = len(route_with_failure) - len(intended_route)

Overhead

Questions:

- How much additional traffic is involved in consensus routing?
- How long does consensus take?
- How much longer are paths when avoiding failure?
- What is the response time to routing updates?

Response Time



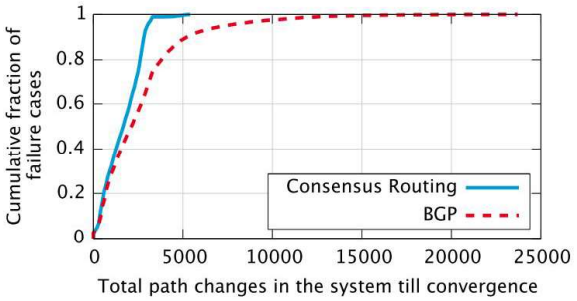
Longer epoch = longer response time

Overhead

Questions:

- How much additional traffic is involved in consensus routing?
- How long does consensus take?
- How much longer are paths when avoiding failure?
- What is the response time to routing updates?
- How many routing changes are made?

Route Flux



BGP requires many more path changes before convergence

Overhead

Questions:

- How much additional traffic is involved in consensus routing?
- How long does consensus take?
- How much longer are paths when avoiding failure?
- What is the response time to routing updates?
- How many routing changes are made?
- What is the overhead of implementation for routers?

Implementation Overhead

XORP
- Open source routing platform

Effects of adding consensus routing
- 8% overhead in update processing
- 11% more lines of code

Goal

“A simple, practical routing protocol that allows general routing policies and achieves high availability”

Discussion

How likely is adoption?
- How likely are ASes to be willing to participate?
- Need ~10 consolidators, are only 13 T1 ASes

What additional policy could be enabled using consensus routing?
- Selectively dropping a peer’s snapshot?

Is too much power given to consolidators?