Engineering Egress with Edge Fabric

Shlinker et al.
Facebook and USC

Presented by: Alex Guerra, Henri Schmidt, Varun Rao

Autonomous Systems (ASes)

- Large network or group of networks that has a unified routing policy.
- Every computer or device that connects to the Internet is connected to an AS.
- Every AS controls a specific set of IP addresses

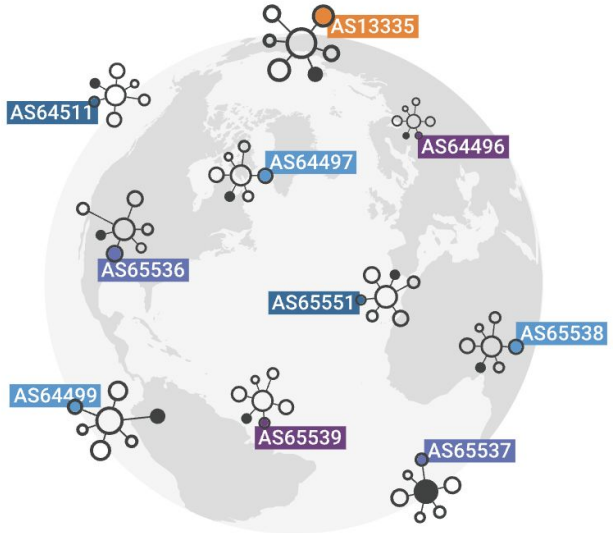


Image Credit:
https://www.cloudflare.com/learning/network-layer/what-is-an-autonomous-system/

BGP Overview

- 2 types: eBGP and iBGP
- eBGP sessions are established between border routers at edge of ASes
- Exchange routes between neighboring ASes
- Distribute routes to internal routers within an AS through iBGP
- This is then combined with IGP (interior gateway protocol) to learn the internal network topology and construct a forwarding table

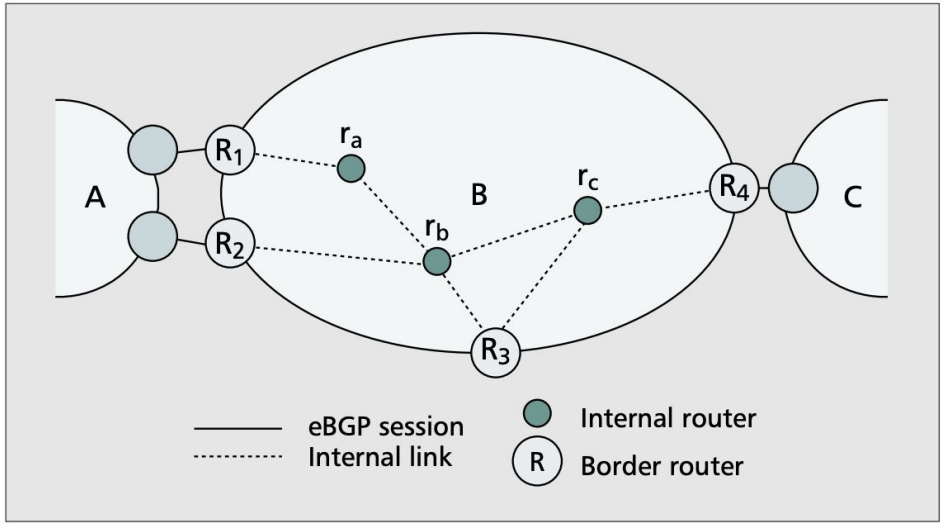


Image Credit: Caesar, Matthew, and Jennifer Rexford. "BGP routing policies in ISP networks." *IEEE network* 2005

BGP's Selection of Routes

- Makes best-path decisions based on current reachability, hop counts and other attributes through hierarchical rules.
- BGP routers are made aware of new routes between ASes or within an AS through broadcasts from neighboring routers
- Steps in the BGP decision process

Step	Attribute	Controlled by local or neighbor AS?
1.	Highest LocalPref	Local
2.	Lowest AS path length	Neighbor
3.	Lowest origin type	Neither
4.	Lowest MED	Neighbor
5.	eBGP-learned over iBGP-learned	Neither
6.	Lowest IGP cost to border router	Local
7.	Lowest router ID (to break ties)	Neither

Image Credit: Caesar, Matthew, and Jennifer Rexford. "BGP routing policies in ISP networks." *IEEE network* 2005

Concept-Check Questions on BGP

A BGP router always prefers routes with the minimum path length.

A) True **B) False**

The decision process to select routes is NOT part of the BGP Protocol Specification

A) True **B) False**

Facebook’s Global Network

- Facebook’s built PoPs around the world
- Interconnect with thousands of Autonomous Systems (ASes)

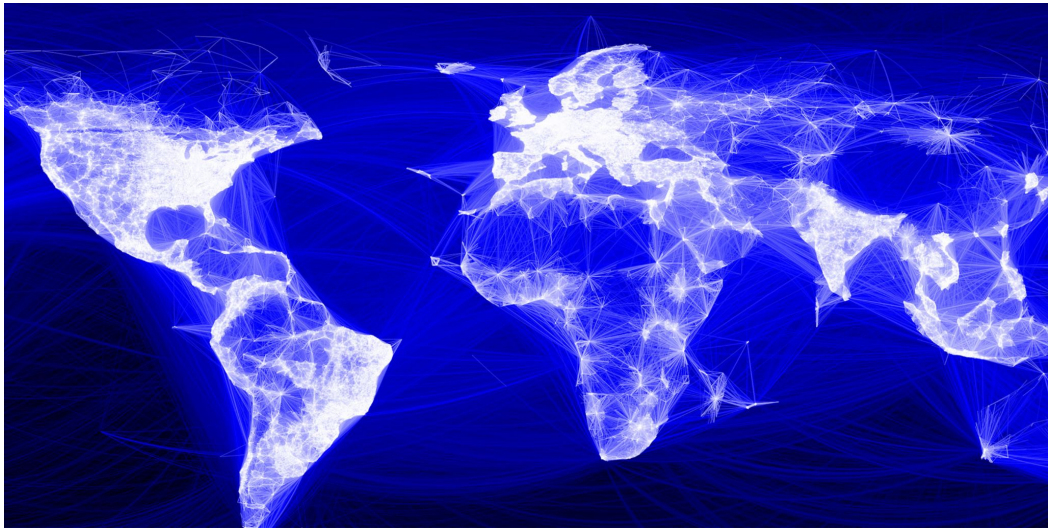
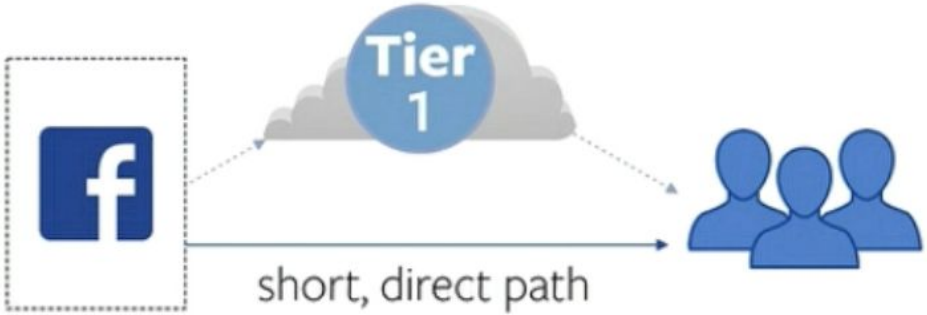


Image Credit: https://engineering.fb.com/wp-content/themes/code-fb-com/img/default_feature.jpg

Benefits of Rich Interconnections

Short, direct paths which can bypass congestion in transit networks

In contrast, the traditional Internet provider hierarchy can struggle to provide the capacity needed to deliver the rapidly growing demand for content



Substantial path diversity since there are multiple diverse paths to route content to end users

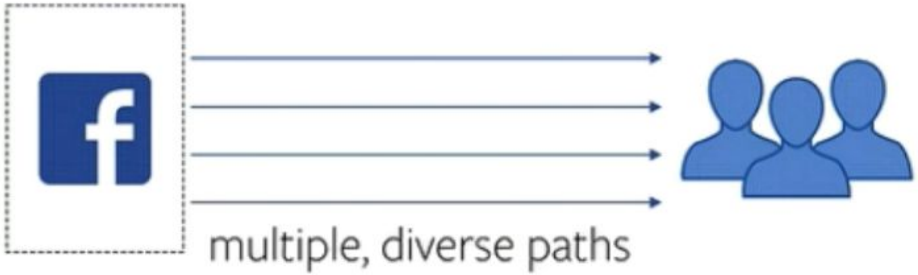


Image credit: Edge Fabric Presentation at SIGCOMM 2017

Where does BGP Fit In?

- At every PoP, FB deploys multiple edge routers
- Physical circuits connect these to external networks
- BGP is used to exchange reachability information, and determine which users can be reached.
- BGP at router selects which route to use

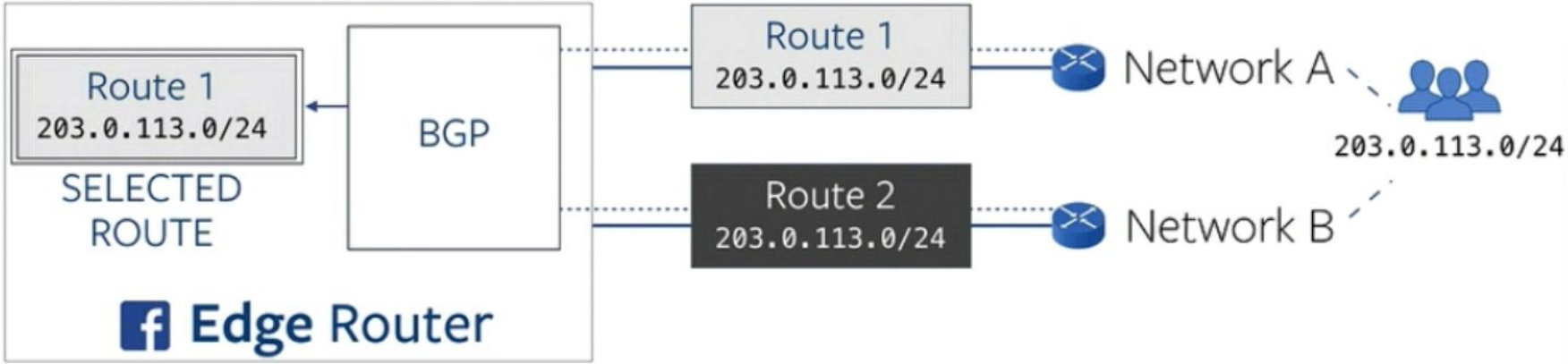


Image credit: Edge Fabric Presentation at SIGCOMM 2017

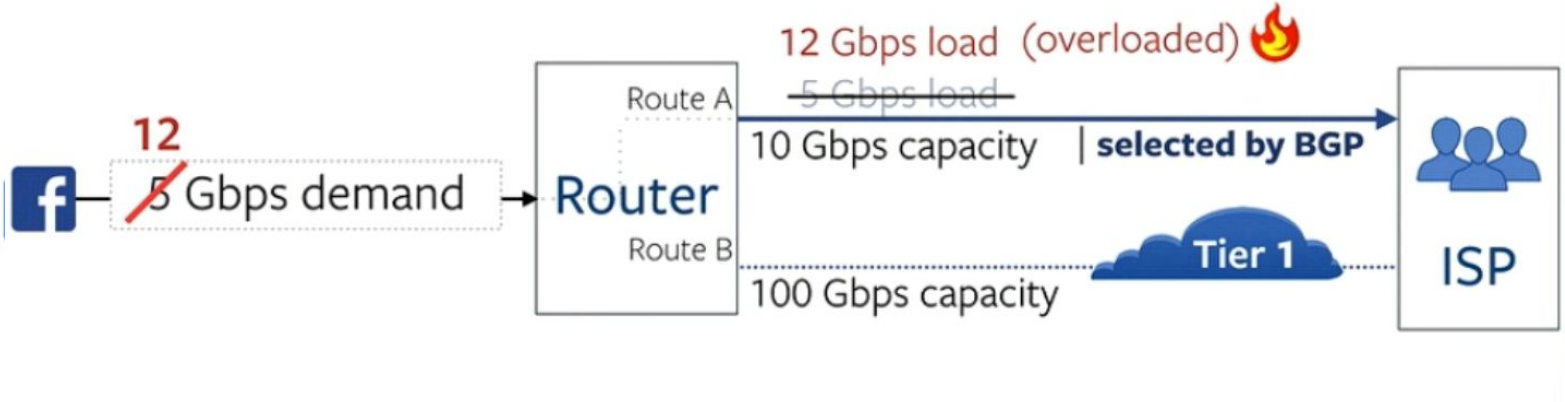
Challenges of BGP

Main Objective: Use rich connectivity provided by BGP and PoPs to deliver traffic to end users with best performance.

Key Challenge: BGP is **NOT CAPACITY** or **PERFORMANCE aware**

Example - BGP is NOT Capacity Aware

Suppose BGP's policy was configured to prefer short, direct paths to end users.

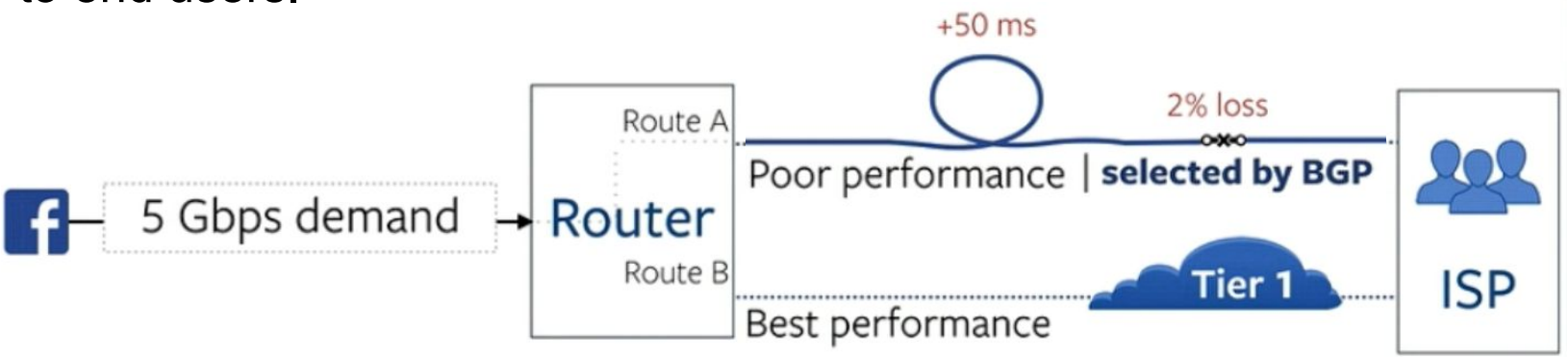


Cannot dynamically configure BGP to adapt to capacity and demand

Image credit: Edge Fabric Presentation at SIGCOMM 2017

Example - BGP is NOT Performance Aware

Suppose BGP's policy was configured to prefer short, direct paths to end users.



Cannot configure BGP to adapt its routing decisions based on performance in real time

Image credit: Edge Fabric Presentation at SIGCOMM 2017

Edge Fabric’s Solution

- BGP is fundamental to interconnection
- All the networks Facebook connects with, expect the use of BGP

Solution

- Shift control from BGP at routers in the edge of the network to a software controller

Interdomain Connectivity at a PoP

Transit Providers

- Deliver traffic to any user on the internet
- 2 or more in each PoP
- Connect with a private circuit

Peers (end-user ISPs, mobile providers)

- Private Peers : 10s per PoP, connect with private circuit
- IXP (Internet Exchange Point Peers): 100s, connect with a shared fabric instead of dedicated private interconnection

BGP Route Preferences

Private Peers > IXP Peers > Transit Providers

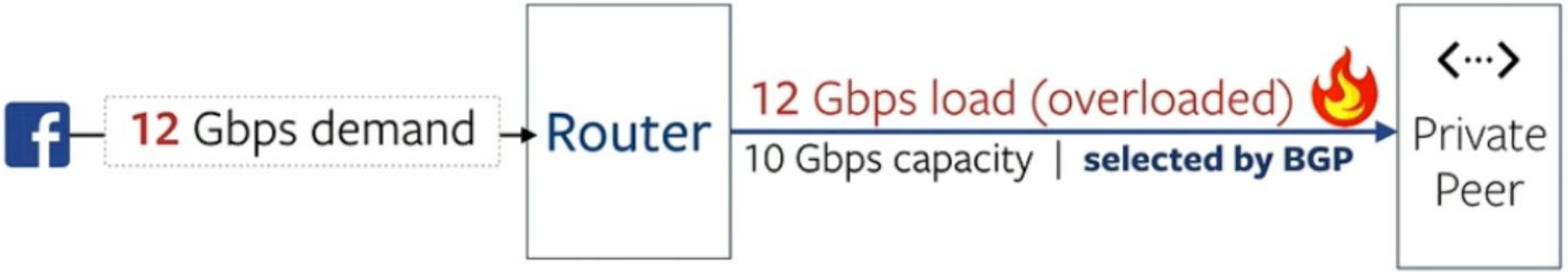
Reasons:

- Peers > Transits
  - Peers provide short and direct paths to end users
- Private peers > IXP peers
  - Prefer routing traffic through a circuit whose capacity is dedicated to traffic between Facebook and other networks

Vast majority of traffic from a Facebook's PoP egresses through **Private Peers**

Key Challenge

Cannot acquire sufficient capacity with private peers to satisfy demand



We have 12 Gbps demand, but a capacity with a peer of only 10 Gbps

Image credit: Edge Fabric Presentation at SIGCOMM 2017

Capacity Constraints in Production

- 2 days study of 20 PoPs
- Subset selected for geographic and connectivity diversity that combined serve most Facebook traffic
- Identified circuits predicted to have demand > capacity

Occurred in 17 out of 20 PoPs

18% of all circuits across all these PoPs had at least one instance where demand exceeded capacity

Capacity Constraints in Production Plot

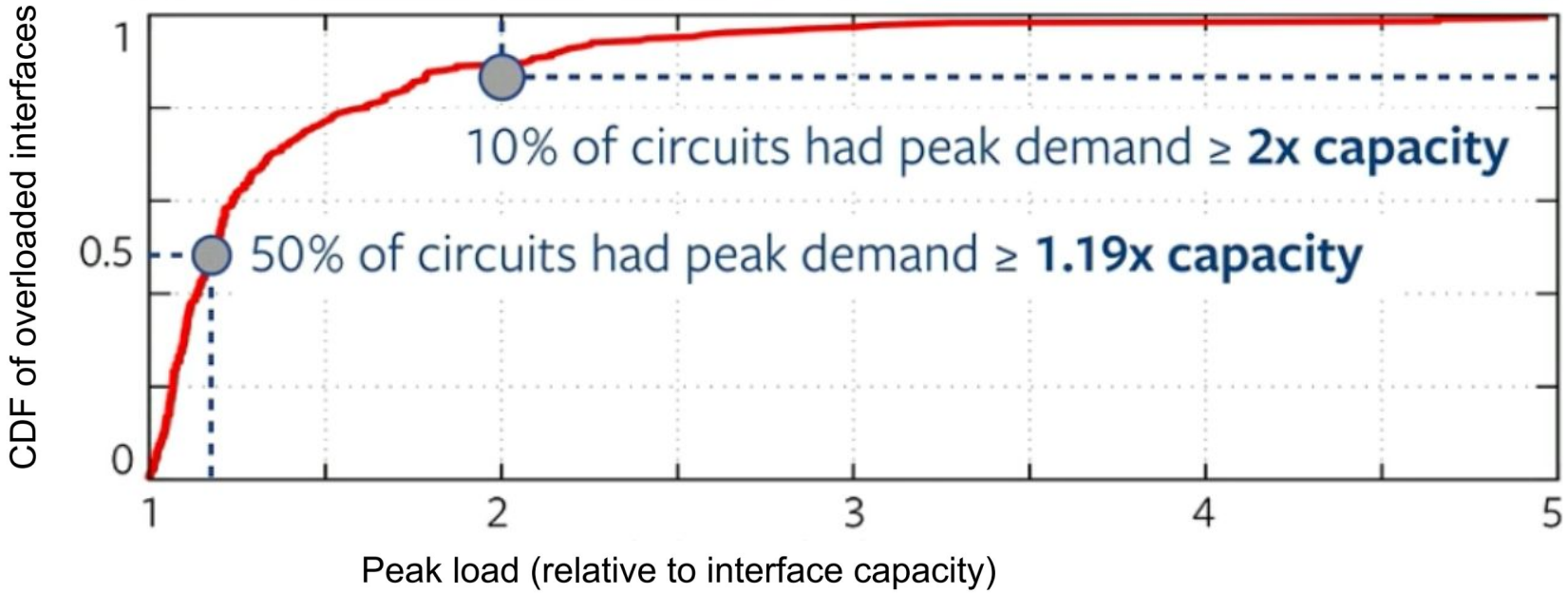


Image credit: Edge Fabric Presentation at SIGCOMM 2017

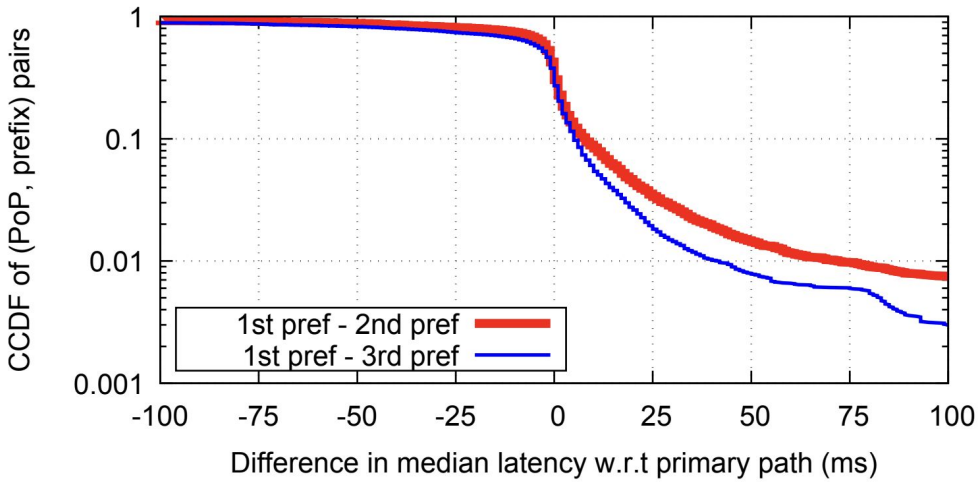
BGP Decisions Can Hurt Production Performance

- To understand the severity of this problem, they compare the performance of alternative paths at four PoPs
- Shift a fraction of actual user traffic for all IP prefixes onto the second and third best BGP path
- Compare the difference in median round-trip latency between the preferred and less preferred paths

BGP Decisions Can Hurt Production Performance Plot

- 5% of <PoP,prefix> pairs could see an improvement of 20+ms if switched from BGP's preferred path to its second preference

- 3% could see such an improvement if switched to BGP's third preference.



Key takeaway: May be able to avoid overload by detouring traffic to less-preferred routes, without resorting to paths that have much worse baseline performance

Sidestepping BGP’s Limitations

Main Objective: Use rich connectivity provided by BGP and PoPs to deliver traffic to end users with best performance.

Key Challenge: BGP is **NOT CAPACITY** or **PERFORMANCE aware**

Approach: Shift Control from BGP at routers to a software controller

High Level Design Priorities

Operation Simplicity

- Minimize change and system complexity for better synchronization with other system components

Ease of deployment

- Want the solution to interoperate with existing infrastructure and tooling
- Deploy to all PoPs around the world and benefit from it immediately

Avoiding A Congested Edge

Avoiding A Congested Edge: Overview

- **Overview**
  - Design goals of *EdgeFabric*
  - High level overview of the design of *EdgeFabric*
  - Measuring network state
    - Implementation
    - Design goals
  - Deciding traffic diversions
    - Policy
  - Diverting traffic
    - A BGP injection trick
    - Design goals
  - Towards performance aware routing

How do we avoid overloading peers?

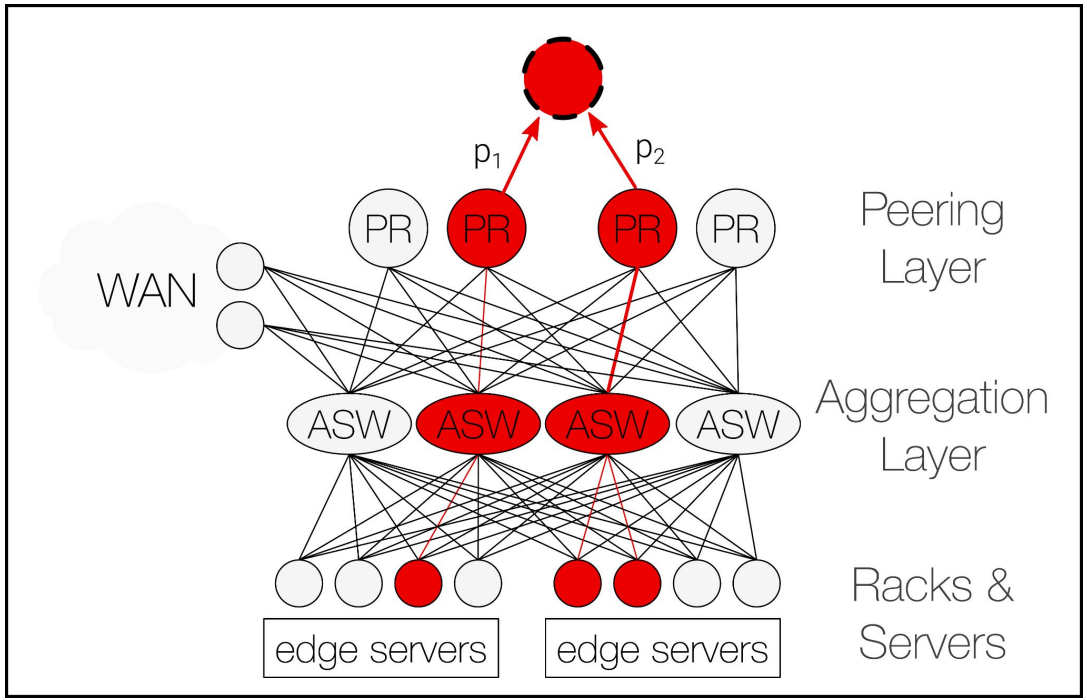


Figure 1: Egress traffic can overload a peer if it **serves** too many prefixes (i.e. p_1, p_2 in this example).

Divert egress traffic through different peers

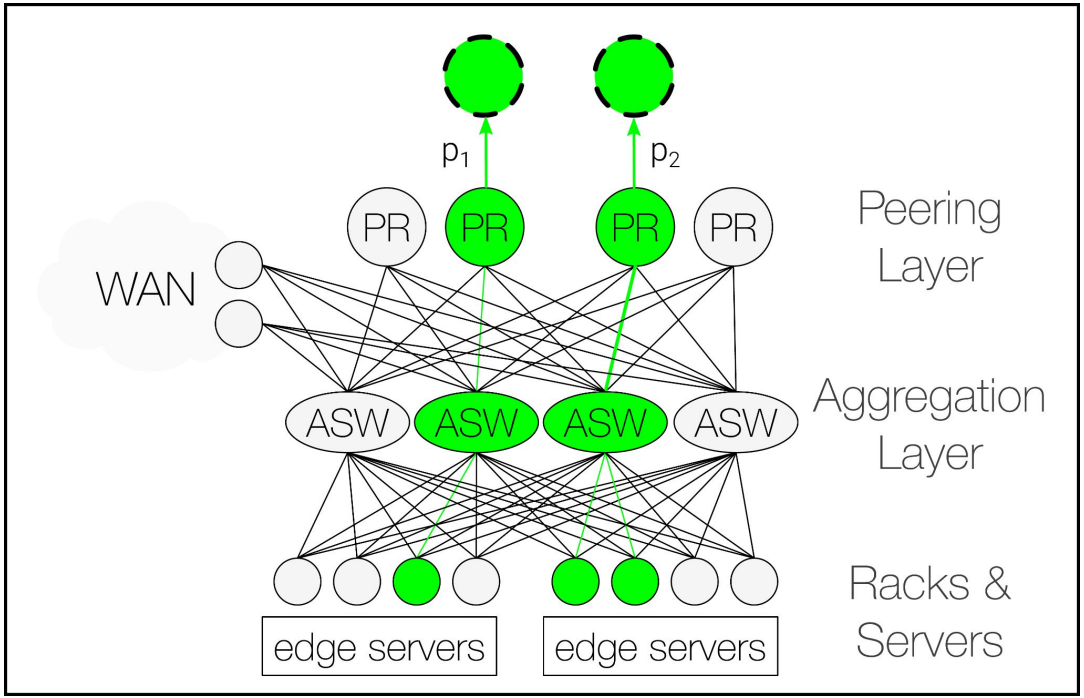


Figure 2: Egress traffic can be diverted by serving distinct prefixes by different peers (e.g. p_2 is now served by a different peer).

Design Goals

EdgeFabric aims to satisfy the following criterion:

I. *Operate on a per-PoP basis*
II. *Centralize control with SDN*
III. *Incorporate real-time traffic and performance measurements into decisions*
IV. *Use BGP for both routing and control*
V. *Leverage existing software and hardware*

High level overview of *EdgeFabric*

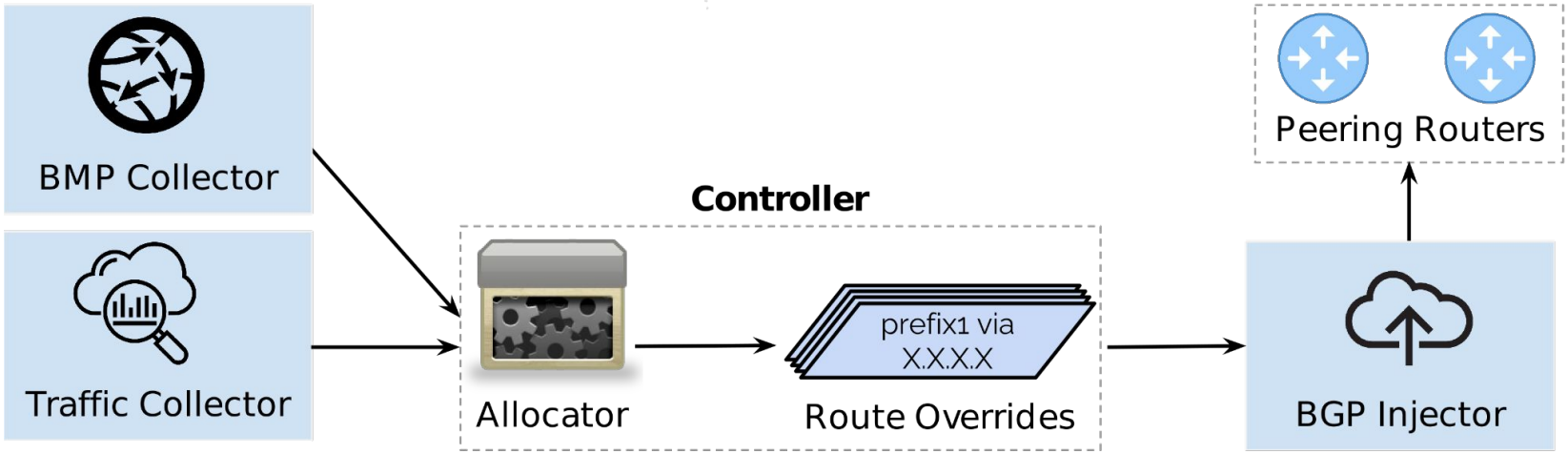


Figure 3: A PoP-level controller watches traffic and gathers routes and overrides routes in peering routers to avoid overloading peers.

How do we know which traffic to divert?

To avoid overloading peers, we first need a mechanism **measure** the amount of egress traffic at the peers and determine the set of **available** paths. However,

I. BGP only yields one available path per prefix
II. BGP does not yield set of available paths
III. Traffic information is not collected

BMP: BGP Monitoring Protocol

- *EdgeFabric* utilizes the BGP Monitoring Protocol (BMP) to enumerate *all available routes per prefix*, and solve *(i)*
- However, BMP does not report the route that would have been used prior to performing any overrides
- Controller then emulates BGP best path selection to determine best route and solve *(ii)*

Collecting Traffic Information

- *EdgeFabric* calculates average egress traffic for each prefix over a two minute window, solving (iii)
- Recursively splits prefixes if average traffic exceeds 250MB/s threshold for finer-grained control

Discussion Question: How does knowing per-prefix traffic information and BGP best path selection enable us to detect congestion?

Monitoring: Design Goals

Discussion Question: How does this set up meet our design goals?

I. *Operate on a per-PoP basis*
II. *Centralize control with SDN*
III. *Incorporate real-time traffic and performance measurements into decisions*
IV. *Use BGP for both routing and control*
V. *Leverage existing software and hardware*

Monitoring: Design Goals

Discussion Question: How does this set up meet our design goals?

I. *Operate on a per-PoP basis:* Controller is per-PoP
II. *Centralize control with SDN:* Controller is software defined
III. *Incorporate real-time traffic and performance measurements into decisions:* Controller measures traffic and path selection to determine congestion
IV. *Use BGP for both routing and control:* Controller uses BGP to determine path selection
V. *Leverage existing software and hardware:* Does not require changes to existing peering routers

Deciding traffic diversions

Though we now know which peers (if any) will be overloaded by default, we need some policy to determine alternative routes

- For each overloaded interface, the controller determines the set of prefixes that will traverse the interface
- It then iteratively selects <prefix, alternative route> pairs to shift away from interface until congestion is resolved

Deciding traffic diversions

Though we now know which peers (if any) will be overloaded by default, we need some policy to determine alternative routes

- For each overloaded interface, the controller determines the set of prefixes that will traverse the interface
- It then iteratively selects <prefix, alternative route> pairs to shift away from interface until congestion is resolved

Natural Question: Which <prefix, alternative route> pairs do we shift away from the interface first?

Policy: Deciding traffic diversions

Facebook uses the following policy to select **alternative routes***:*

1. Prefer IPv4 over IPv6 prefixes
2. Prefer prefixes that a peer prefers Facebook detour
3. Among multiple alternate routes for a given prefix, prefer routes with the longest prefix
4. Prefer paths based on BGP's best path selection process
5. Prefer paths based on an arbitrary but deterministic tiebreaker

Policy: Deciding traffic diversions

Facebook uses the following policy to select **alternative routes***:*

1. Prefer IPv4 over IPv6 prefixes
2. Prefer prefixes that a peer prefers Facebook detour ????
3. Among multiple alternate routes for a given prefix, prefer routes with the longest prefix
4. Prefer paths based on BGP's best path selection process
5. Prefer paths based on an arbitrary but deterministic tiebreaker

Incorporating traffic decisions

Question: Assuming we have made decisions on which <prefix, alternate path> pairs to select, how do we ensure a prefix is handled by its selected path?

- We *inject* BGP overrides using a BGP Injector Service
- New allocation and injection is done on 30 second intervals
- Old overrides are removed
- Ensure that it does not conflict with load balancer

A trick: modifying the `local_pref` attribute

BGP supports the ability to modify the local preference of routes by setting the `local_pref` attribute to prefer an alternative route.

Network	Next Hop	Metric	LocPrf	Weight	Path
192.0.2.0/24	1.1.1.1	20		0	1010 1011 286 4040 i
	2.2.2.2	20		0	2020 1011 4040 i
	3.3.3.3	10	100	0	2020 702 4040 i
	4.4.4.4	0	90	0	4040 i
	5.5.5.5	265		0	5050 1011 4040 i
	6.6.6.6		80	0	6060 4040 i
	7.7.7.7	10	100	0	7070 3356 3356 4040 i

Incorporating Traffic Decisions: Design Goals

Discussion Question: How does this set up meet our design goals?

I. *Operate on a per-PoP basis:* Controller is per-PoP
II. *Centralize control with SDN:* Controller is software defined
III. *Incorporate real-time traffic and performance measurements into decisions:* Controller measures traffic and path selection to determine congestion
IV. *Use BGP for both routing and control:* Controller uses BGP `local_pref` attribute override to determine path selection
V. *Leverage existing software and hardware:* Does not require changes to existing peering routers since `local_pref` can be modified via iBGP broadcast

Towards Performance Aware Routing

EdgeFabric avoids congestion (i.e. capacity aware), but it is not necessarily performance aware. For example:

- The selected BGP path may not be the best performing path
- Alternative paths may be worse performing than the selected BGP path
- BGP only supports destination based routing
- No content prioritization

Towards Performance Aware Routing

To tackle these challenges, *Facebook* develops a mechanism to direct specific flows along certain paths

- They use this mechanism to *measure* the performance of various paths
- Then inject new, “better performing” routes into the routing tables of the peering routers
- Future Work:
  - Override default BGP paths
  - Optimize use of limited capacity for high priority applications

Evaluating *EdgeFabric*

Results on Production Traffic

EdgeFabric was deployed for all production, creating detours whenever necessary to optimize traffic flow. Two main studies of this deployment were documented:

Evaluating Capacity Aware Routing: Two day study done in January 2017, *predating* the stateless controller. This stateful controller did *not* automatically split large-volume prefixes. It is believed (but not formally evaluated) that the stateless controller is better than what is shown in the study.

Evaluating Performance Aware Routing: From trial deployment of AltPath in late 2016 at four PoPs. Traffic here accounted for about 18% of all PoPs.

Evaluating Capacity Aware Routing

Does EdgeFabric prevent congestion at edge interfaces while enabling efficient utilization?

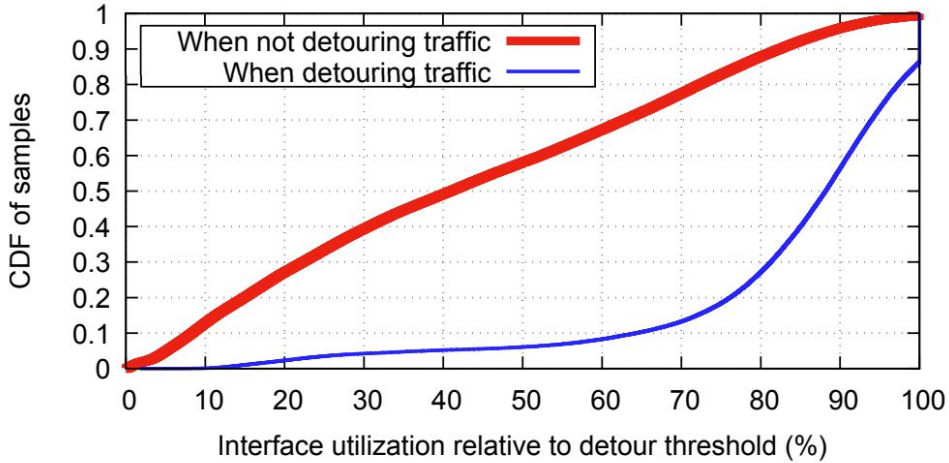


Figure 9: Utilization of interfaces relative to detour thresholds.

- Highest utilization is around 86%
- Less than 30% of of samples have >80% utilization

Evaluating Capacity Aware Routing

How much traffic does EdgeFabric detour?

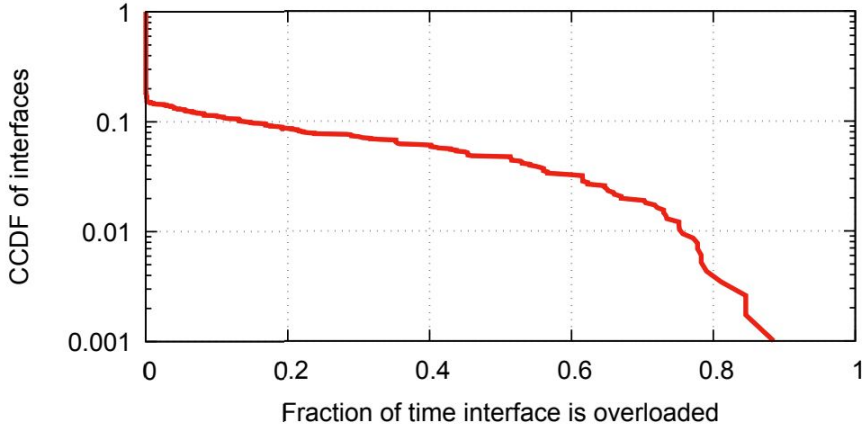


Figure 10: Fraction of time EDGE FABRIC detours from interfaces.

- Detoured traffic from 18% of interfaces at least once
- 5% of interfaces were detoured for the majority of the period.

Evaluating Capacity Aware Routing

How long do these detours last?

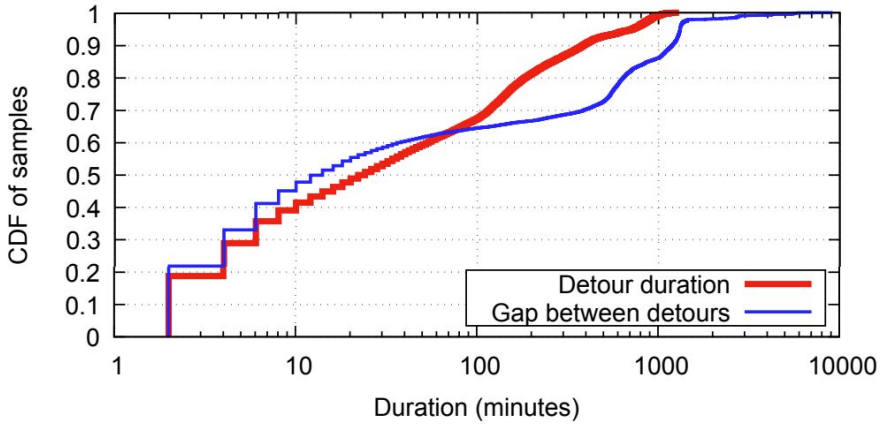


Figure 11: Distributions of EDGE FABRIC detour period lengths across (PoP, prefix) pairs and of time between detours.

- Median detour last 20 minutes, while median gap between detour is ~14 minutes.
- 10% of detours last >6 hours, but gaps between have 36% lasting >3 hours

Evaluating Capacity Aware Routing

How much traffic is being detoured at each PoP, and is this sustainable for the rest of the network?

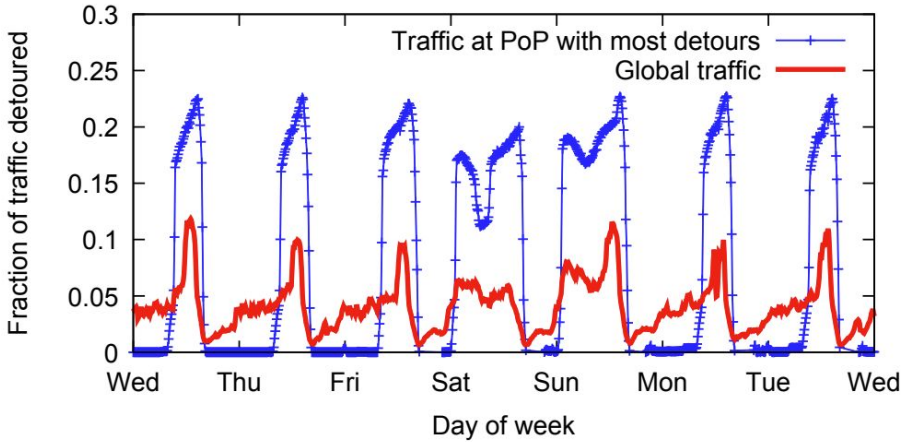


Figure 12: Fraction of traffic detoured by Edge Fabric across 20 PoPs and at the PoP with the largest fraction detoured.

- Plenty of spare capacity to absorb detours; PoP's *always* had at least 45% of their transit capacity free.

Evaluating Performance Aware Routing

What is the performance impact of using measurements of alternate paths to inform better decisions?

Comparing the AltPath designated path vs the default BGP path, the following results were obtained:

- 45% of prefixes had their median improve by at least 20 ms
- 28% of prefixes had their median latency improve by at least 100ms
- 17% of prefixes had a median latency got worse by at least 20ms
- 1% of prefixes had a median latency got worse by at least 100ms

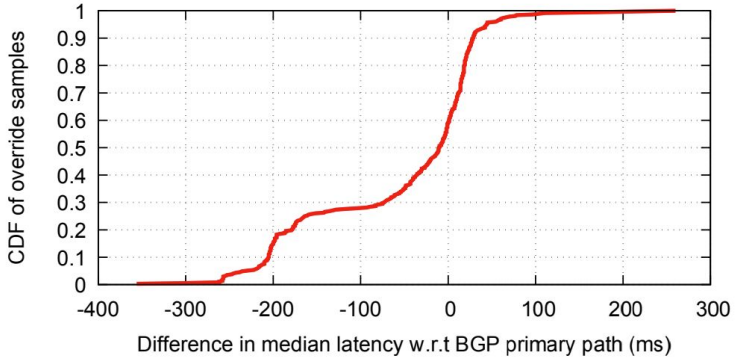


Figure 13: Performance difference between best paths chosen based on BGP policy and chosen based on alternate path measurements. Graph depicts latency-selected path minus policy-selected path, and so a negative value indicates that EDGE FABRIC's override achieved better performance (lower latency).

Discussion: What might cause the median latency to drop when using AltPath?

Evaluating Performance Aware Routing

Does AltPath accurately capture end to end performance?

- Conducted controlled experiment with the PEERING testbed. Used Linux traffic control framework to induce 40ms of increased latency on incoming traffic.

- Used AltPath measurements to estimate the difference in induced latency on the direct and transit paths over 5 minute intervals in 18 hour experiment.

- AltPath identified the difference in induced latency within 2.2ms of the induced difference at all times, with an average error of 0.6ms.

Operational Experience: Evolution of Edge Control

- Stateful => Stateless control
  - Computes projected utilization in 30 sec cycles without knowledge of previous detouring. Makes design simpler, easier to deploy/upgrade, and easy to test.
- Host based => edge based routing
  - Current implementation uses BGP to enact overrides, which only requires hosts to signal which flows require special treatment. Does not require hosts to be aware of network state and reduces synchronization.
- Global => per-PoP edge egress options
  - Previously, Facebook routed traffic on iBGP mesh, such that user's traffic could ingress at one PoP and egress at another. This could lead to unstable behavior (oscillations), so the design was simplified by having a global load balancer control where traffic ingresses, and having it egress from the same PoP as this ingress.
- Balanced => imbalance capacity
  - As size and scale of PoP's grew, there was more imbalanced capacity. Instead of solving with WCMP routing, they modified EdgeFabric to handle this automatically.

Operational Experience: The Challenge of IXPs

- Problem: At Internet Exchange Points (IXPs), a provider cannot know how much capacity is at a peer's port, since other entities may also simultaneously sending to that port.

- Some exchanges report *total* capacity per port, but this is not enough information since we still don't know the rest of the traffic at the exchange.

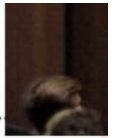
- Solution: Set capacity constraints by getting estimates of the maximum usable capacity from large public exchange peers, as they have more insight into their interface capacity and utilization.

Related Work

- Footprint*
  - focuses on shifting load of long-lived stateful client sessions between PoPs to avoid congestion.
- PECAN*
  - focuses on measuring performance and choosing ingress routes.
- Entact
  - overrides BGP's default routing decisions through a well-designed approach that balances performance, load, and cost, evaluating the approach via emulation.
- Espresso
  - Basically, Google's version of EdgeFabric. Many different design decisions based on varying priorities.
- B4, SWAN
  - Both systems centralize control of inter-datacenter networks to maximize utilization without hurting performance, but the main difference is that these systems operate in closed networks under unified administration.

* : complementary to EdgeFabric

EdgeFabric vs. Espresso



	Facebook's **Edge Fabric**	Google's **Espresso**
	use BGP to exchange routes with peers centralize control and incorporate additional inputs	
edge device	router	MPLS switch
enacts decisions via	BGP injections to routers	host-based overrides
role of hosts	mark packet's priority	select packet's route
decision granularity	<destination, priority/class>	packet
routing options	**per-PoP**	**global**
design priorities	Operational simplicity Ease of deployment	Maximum flexibility Cost savings

Conclusion

- Internet traffic dominated by a small number of content providers. Getting interdomain routing is critical for high performance/low congestion.

- EdgeFabric was introduced to augment BGP with measurement and control mechanisms to overcome BGP’s lack of congestion and performance awareness.

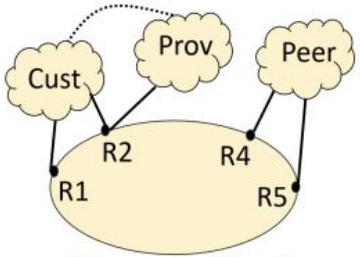
- EdgeFabric successfully reduces congestion of capacity constrained interconnections, provides great insight as to how future interdomain routing systems might be designed.

Optional Reading: Don’t Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations

Beckett et al.
Princeton, UCLA, and Microsoft

Main Problem/Motivation

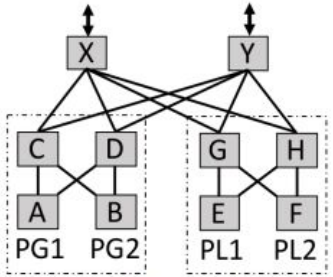
When generating BGP configurations, operators face the challenge of decomposing network level policies into correct device-level policies.



Policy

P1. Prefer Cust > Peer > Prov
P2. Disallow traffic between Prov and Peer
P3. For Cust, prefer R1 > R2
P4. Cust must be on path for its prefixes
P5. Cust must not be a transit to Prov

Figure 1: Creating router-level policies is difficult.



Policy

P1. Left cluster has global services with PG* prefixes, which should be announced externally as an aggregate PG

P2. Right cluster has local services with PL* prefixes, which should not be announced externally

Figure 2: Policy-compliance under failures is difficult.

The Solution: Propane

- Propane is a Domain Specific language (DSL) for specifying network policies.
- Behavior in the case of failures is specified by *path preferences* in the policy (can be underspecified).
- Once the policy has been programmed, there are many safety checks done to assert that the policy is compliant under failures.
- Final output is abstract (vendor neutral) BGP routing configuration.

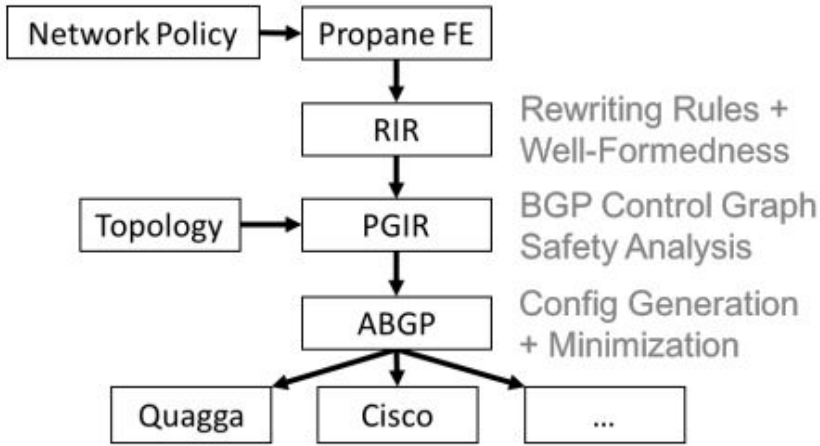


Figure 3: Compilation pipeline stages for Propane.

The Compilation Process: Propane FE => RIR

Syntax

pol	::=	$p_1, \ldots, p_n$	*policies*
p	::=	t => r_1»…»r_m | cc	*constraints*
x	::=	$d.d.d.d/d$	*prefix*
t	::=	`true`	*true*
	|	$!t$	*negation*
	|	$t_1 \mid t_2$	*disjunction*
	|	$t_1 \& t_2$	*conjunction*
	|	$prefix = x$	*prefix test*
	|	$comm = d$	*community test*
r	::=	l	*location*
	|	$\emptyset$	*empty set*
	|	**in**	*internal loc*
	|	**out**	*external loc*
	|	$r_1 \cup r_2$	*union*
	|	$r_1 \cap r_2$	*intersection*
	|	$r_1 \cdot r_2$	*concatenation*
	|	$!r$	*path negation*
	|	r^*	*iteration*
ln	::=	$r_1 \rightarrow r_2$	*links*
cc	::=	$agg(x, ln)$ | $tag(d, t, ln)$	*control constraints*

Propane Expansions

any	=	$\mathbf{out}^* \cdot \mathbf{in}^+ \cdot \mathbf{out}^*$
drop	=	$\emptyset$
internal	=	$\mathbf{in}^+$
only(X)	=	$\mathbf{any} \cap X^*$
never(X)	=	$\mathbf{any} \cap (!X)^*$
through(X)	=	$\mathbf{out}^* \cdot \mathbf{in}^* \cdot X \cdot \mathbf{in}^* \cdot \mathbf{out}^*$
later(X)	=	$\mathbf{out}^* \cdot (X \cap \mathbf{out}) \cdot \mathbf{out}^* \cdot \mathbf{in}^+ \cdot \mathbf{out}^*$
before(X)	=	$\mathbf{out}^* \cdot \mathbf{in}^+ \cdot \mathbf{out}^* \cdot (X \cap \mathbf{out}) \cdot \mathbf{out}^*$
end(X)	=	$\mathbf{any} \cap (\Sigma^* \cdot X)$
start(X)	=	$\mathbf{any} \cap (X \cdot \Sigma^*)$
exit(X)	=	$(\mathbf{out}^* \cdot \mathbf{in}^* \cdot (X \cap \mathbf{in}) \cdot \mathbf{out} \cdot \mathbf{out}^*) \cup$ $(\mathbf{out}^* \cdot \mathbf{in}^+ \cdot (X \cap \mathbf{out}) \cdot \mathbf{out}^*)$
enter(X)	=	$(\mathbf{out}^* \cdot \mathbf{out} \cdot (X \cap \mathbf{in}) \cdot \mathbf{in}^* \cdot \mathbf{out}^*) \cup$ $(\mathbf{out}^* \cdot (X \cap \mathbf{out}) \cdot \mathbf{in}^+ \cdot \mathbf{out}^*)$
link(X, Y)	=	$\mathbf{any} \cap (\Sigma^* \cdot X \cdot Y \cdot \Sigma^*)$
path$(\vec{X})$	=	$\mathbf{any} \cap (\Sigma^* \cdot X_1 \ldots X_n \cdot \Sigma^*)$
novalley$(\vec{X})$	=	$\mathbf{any} \cap$ $!\mathbf{path}(X_2, X_1, X_2) \cap \cdots \cap$ $!\mathbf{path}(X_n, X_{n-1}, X_n)$

Figure 4: Regular Intermediate Representation (RIR) syntax (left), and Propane language expansions (right).

The Compilation Process: Propane RIR => PGIR

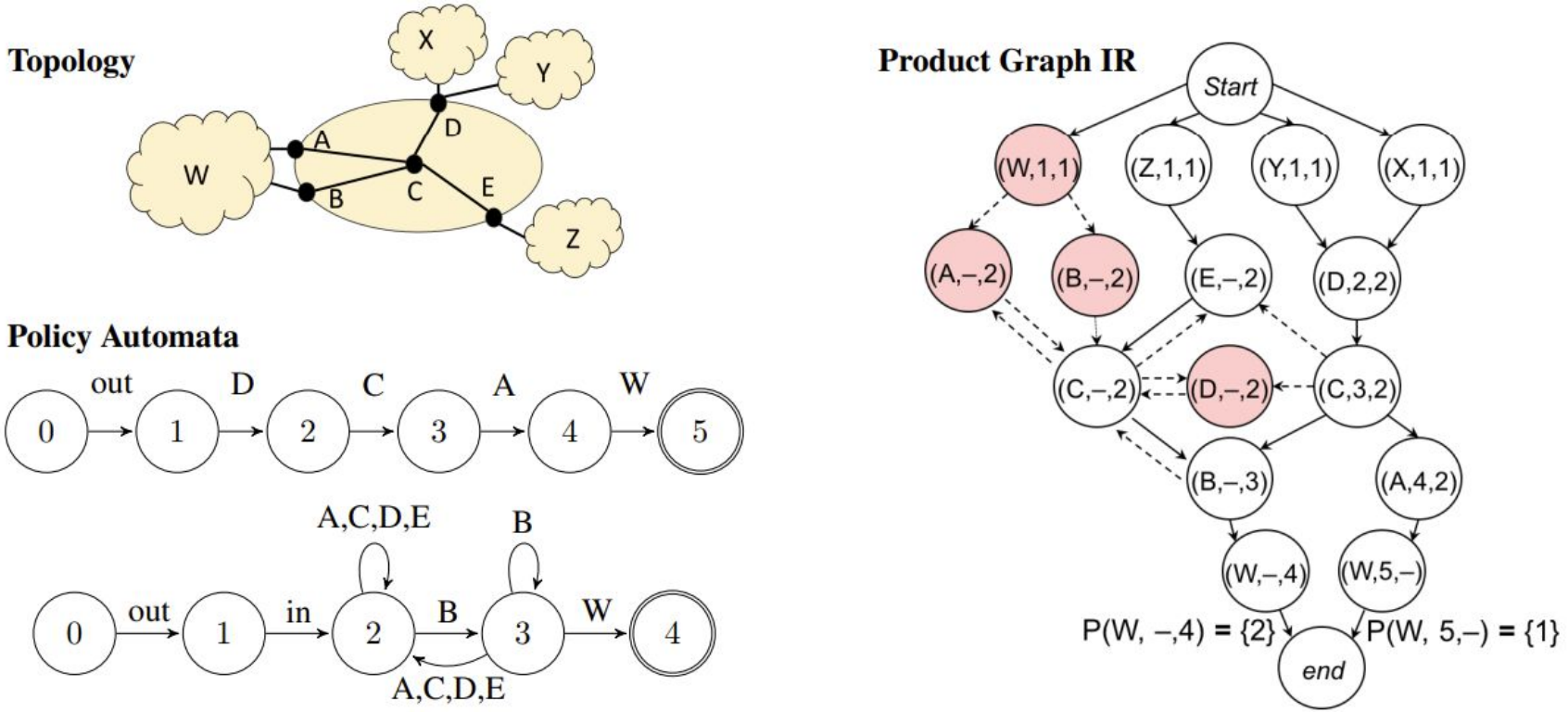


Figure 5: Product graph construction for policy $(\texttt{W} \cdot \texttt{A} \cdot \texttt{C} \cdot \texttt{D} \cdot \textbf{out}) \gg (\texttt{W} \cdot \texttt{B} \cdot \textbf{in}^{+} \cdot \textbf{out})$.

Safety Analysis: Imposing order on the PGIR

- BGP uses local preferences on per device basis.
- Our compiler must be able to choose which shadow to prefer in the PGIR. This requires a *total ordering* on the shadow nodes for each router.
- Goal: Order shadow nodes so as to be policy compliant in the event of failures.
- Solution: Regret free preferences

Algorithm 1 Inferring regret-free preferences

```
1:  procedure REGRET-FREE(G, N_1, N_2)
2:      if N_1 ≉ N_2 then return false
3:      q ← Queue()
4:      q.Enqueue(N_1, N_2)
5:      while !q.Empty() do
6:          (m, n) ← q.Dequeue()
7:          if m ≰_rank n then return false
8:          for n' in adj(G, n) do
9:              if (∃m' ∈ adj(G, m), m' ≈ n') or
10:                 (∃m' ∈ G, dominates m, m' ≈ n') then
11:                 if (m', n') not marked then
12:                     mark (m', n') as seen
13:                     q.Enqueue(m', n')
14:             else return false
15:     return true
```

The Compilation Process: PGIR => ABGP

- Given a total ordering on the PGIR, can represent a state of the automata as a ‘community value’.

- Routers can now match based on their peers and the community value corresponding to the state of the PGIR

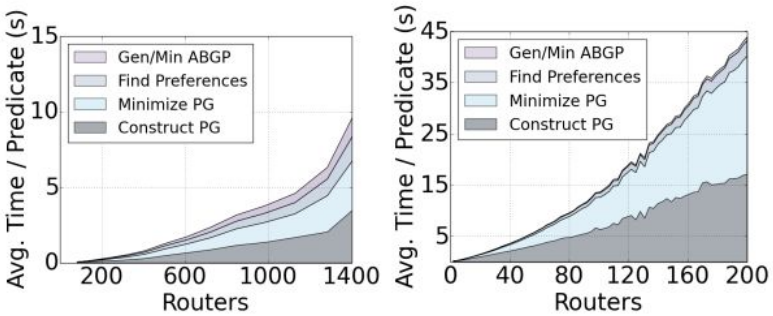


Figure 10: Compilation time.

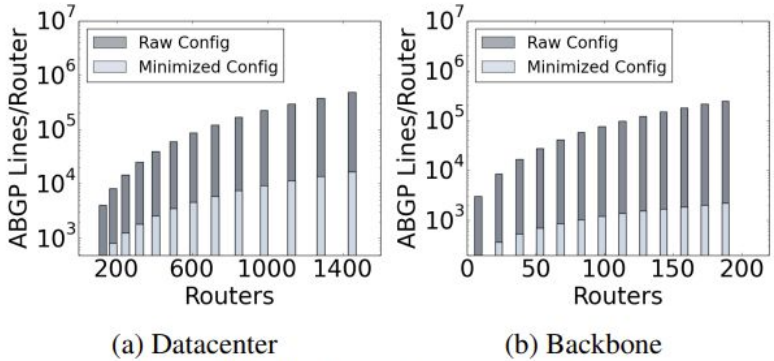


Figure 11: Configuration minimization.

Final Discussion:

How are EdgeFabric and Propane related, i.e., why are we reading these papers on the same week?