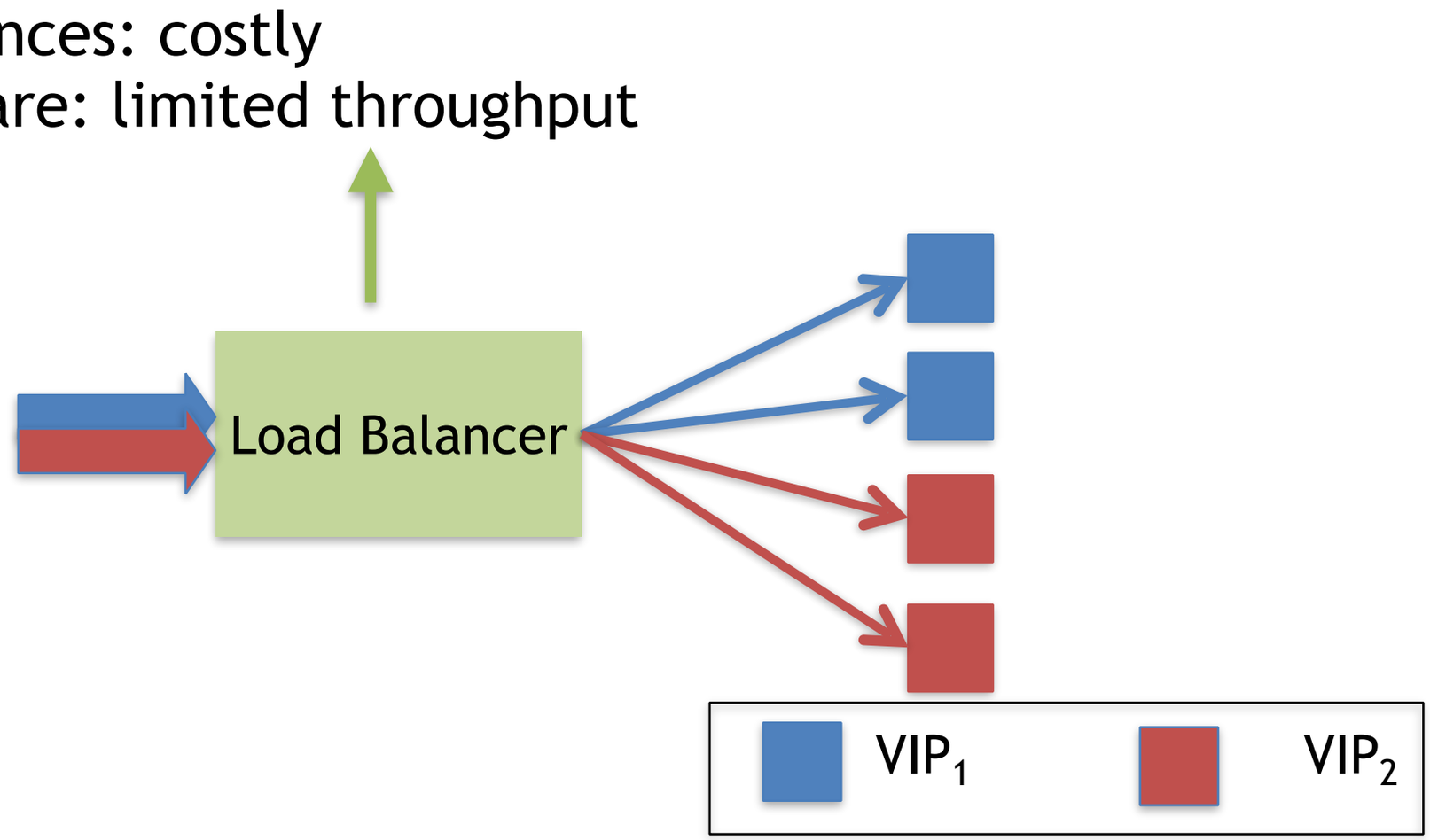
Niagara: Efficient Traffic Splitting on Commodity Switches

Nanxi Kang, Monia Ghobadi,
John Reumann, Alexander Shraer,
Jennifer Rexford

Service load balancing

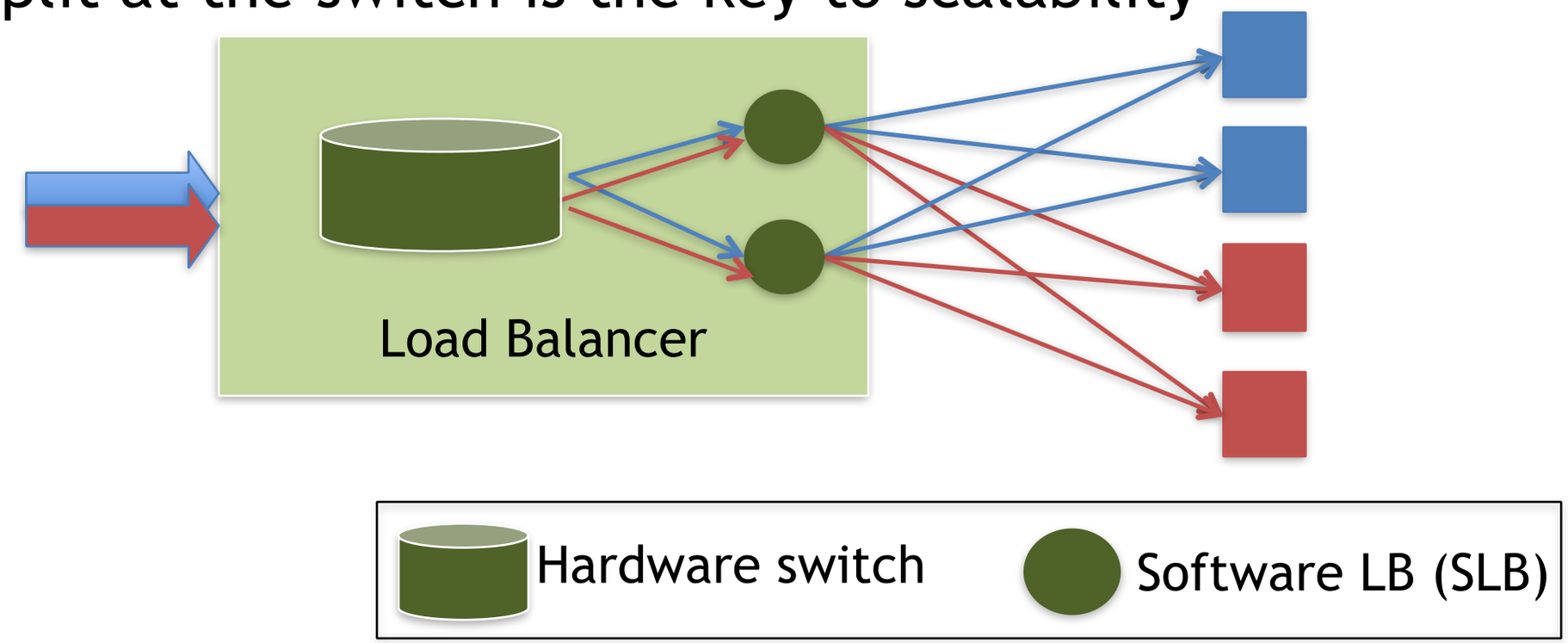
- A network hosts many services (Virtual-IPs)
- Each service is replicated for greater throughput
- A load balancer spreads traffic over service instances

X > Appliances: costly
> Software: limited throughput



Hierarchical Load Balancer

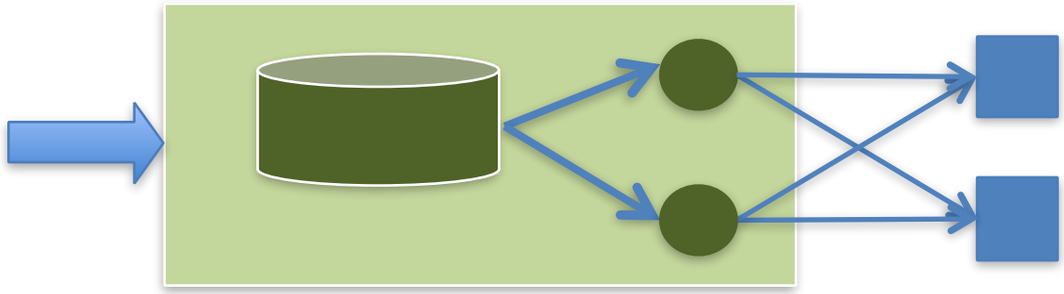
- Modern LB scales out with a hierarchy[1][2]
  - A hardware switch split traffic over SLBs
  - SLBs direct requests to servers
    - SLBs track connections and monitor health of servers
- Traffic split at the switch is the key to scalability



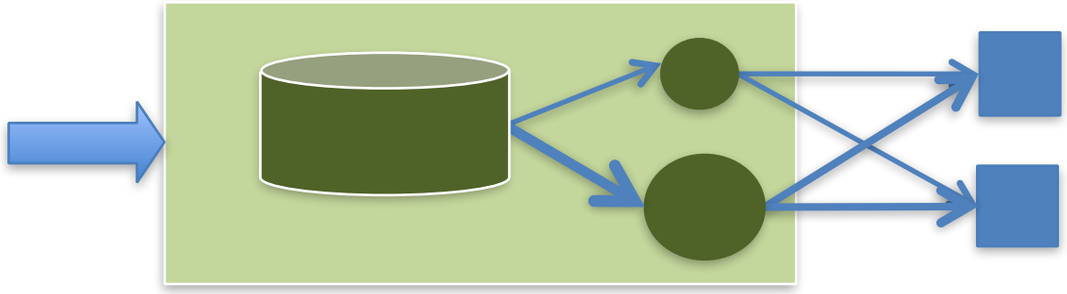
[1]: Duet (SIGCOMM'14)
[2]: Ananta (SIGCOMM'13)

Accurate Weighted Split

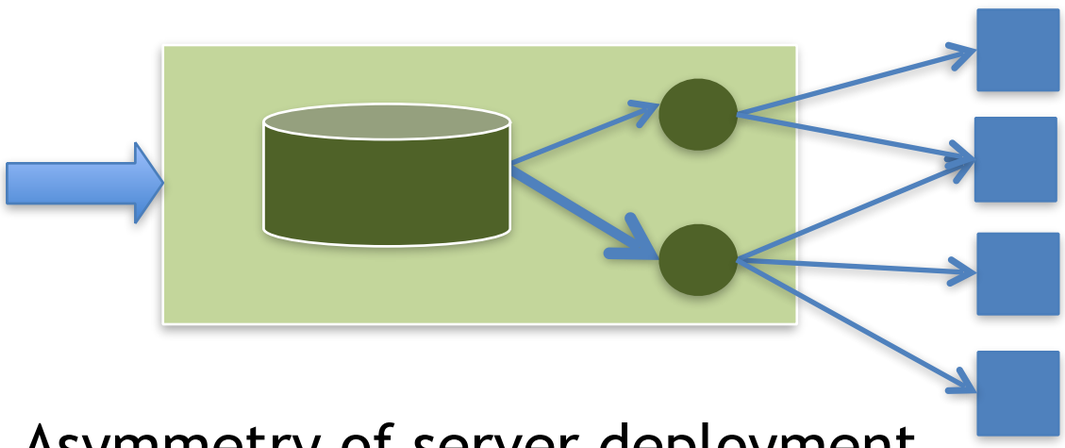
- SLBs are weighted in the traffic split
  - Throughput of SLB
  - Deployment of VIP
  - Failures, or recovery



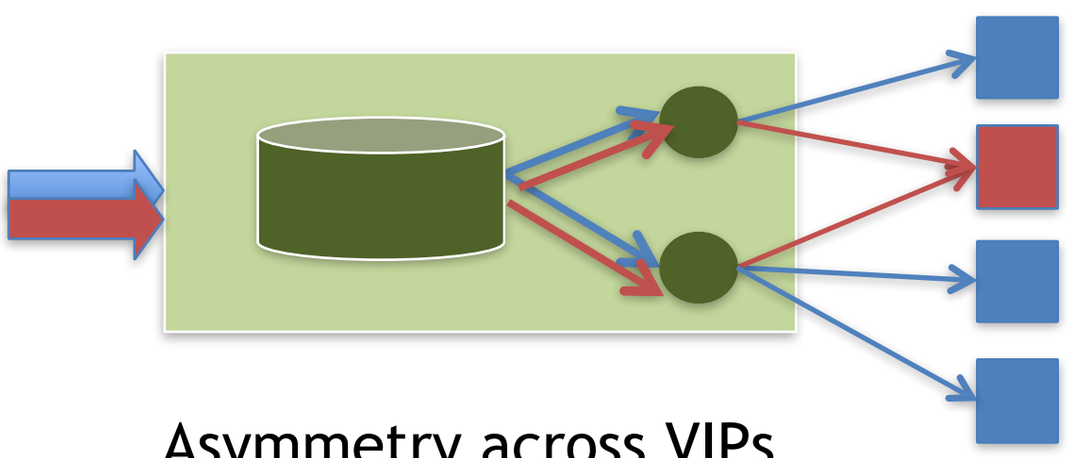
Symmetry



Asymmetry of LB



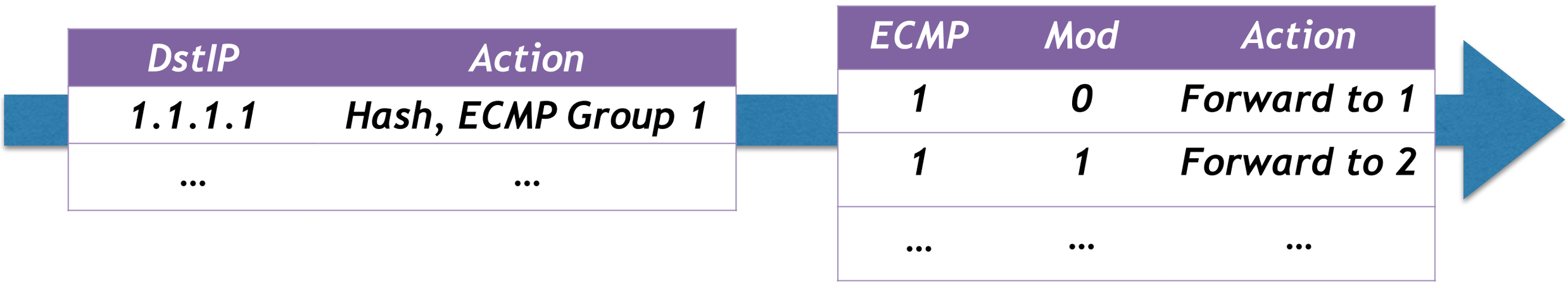
Asymmetry of server deployment



Asymmetry across VIPs

Existing hash-based split

- Hash-based ECMP
  - Hash 5-tuple header fields of packets
  - Dst_SLB = Hash_value mod #SLBs



Equal split over two SLBs

Existing hash-based split

- Hash-based ECMP
  - Hash 5-tuple header fields of packets
  - Dst_SLB = Hash_value mod #SLBs
- WCMP gives unequal split by repeating

DstIP	Action
1.1.1.1	Hash, ECMP Group 1
…	…

ECMP	Mod	Action
1	0	Forward to 1
1	1	Forward to 2
1	2	Forward to 2
…	…	…

(1/3, 2/3) is achieved by adding the second SLB twice

Existing hash-based split

- ECMP and WCMP only split the *flowspace* equally
  - WCMP cannot scale to many VIPs, due to the rule-table constraint
    - e.g., (1/8, 7/8) takes 8 rules

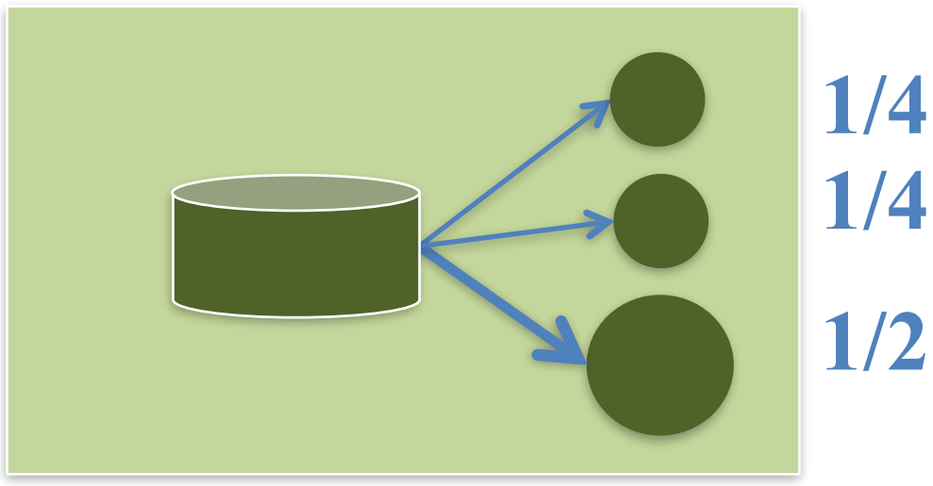
DstIP	Action
1.1.1.1	Hash, ECMP Group 1
…	…

ECMP	Mod	Action
1	0	Forward to 1
1	1	Forward to 2
1	2	Forward to 2
…	…	…

Limited rule-table size (TCAM)

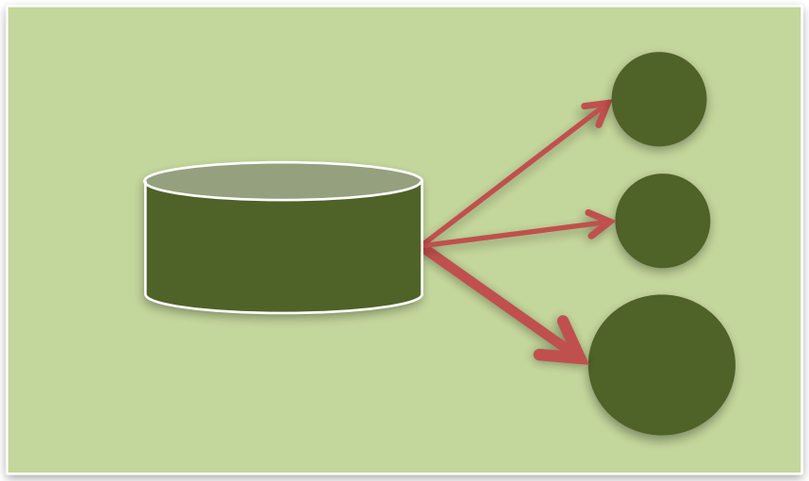
A wildcard-matching approach

- OpenFlow + TCAM
  - OpenFlow : program rules at switches
  - TCAM : support wildcard matching on packet headers
- A starting example
  - Single service : VIP = 1.1.1.1
  - Weight vector: (1/4, 1/4, 1/2)



Match (dst_ip, src_ip)		Action
1.1.1.1	*00	Forward to 1
1.1.1.1	*01	Forward to 2
1.1.1.1	*	Forward to 3

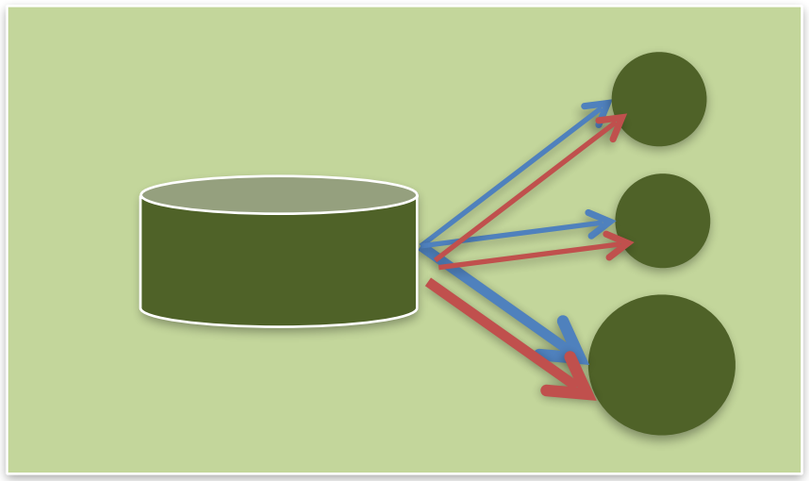
Challenges: Accuracy



- How rules achieve the weight vector of a VIP?
  - Arbitrary weights
  - Non-uniform traffic distribution over flowspace

#bytes or #connections

Challenges: Accuracy



1/6 1/4
1/3 1/4
1/2 1/2

?

- How rules achieve the weight vector of a VIP?
  - Arbitrary weights
  - Non-uniform traffic distribution over flowspace

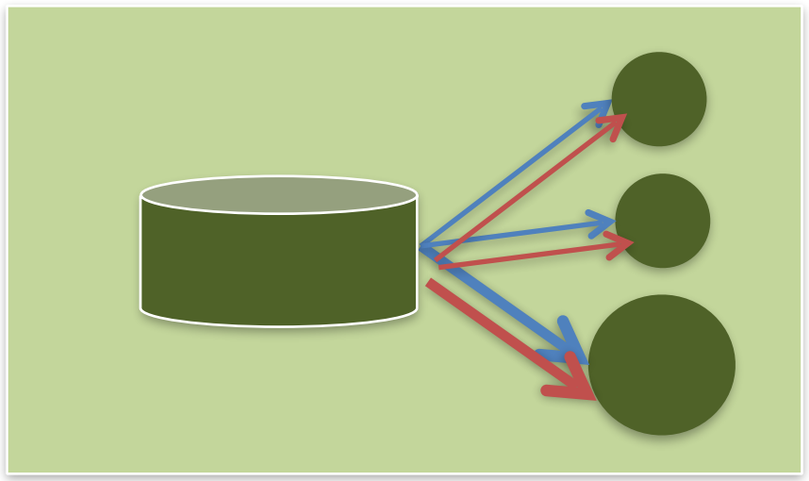
1. Approximate weights with rules

- How VIPs (100 -10k) share a rule table (~4,000)?

2. Packing rules for multiple VIPs
3. Sharing default rules
4. Grouping similar VIPs

Niagara: rule generation algorithms!

Challenges: Accuracy



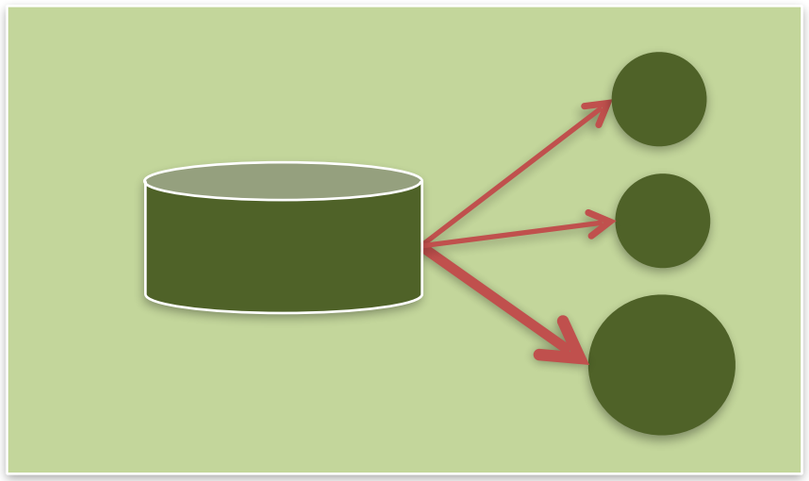
1/6 1/4
1/3 1/4
1/2 1/2

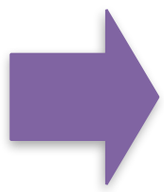
?

- How rules achieve the weight vector of a VIP?
  - Arbitrary weights — **1. Approximate weights with rules**
  - Non-uniform traffic distribution over flowspace
- How VIPs (100 -10k) share a rule table (~4,000)?
  - **2. Packing rules for multiple VIPs**
  - **3. Sharing default rules**
  - **4. Grouping similar VIPs**

Niagara: rule generation algorithms!

Basic ideas



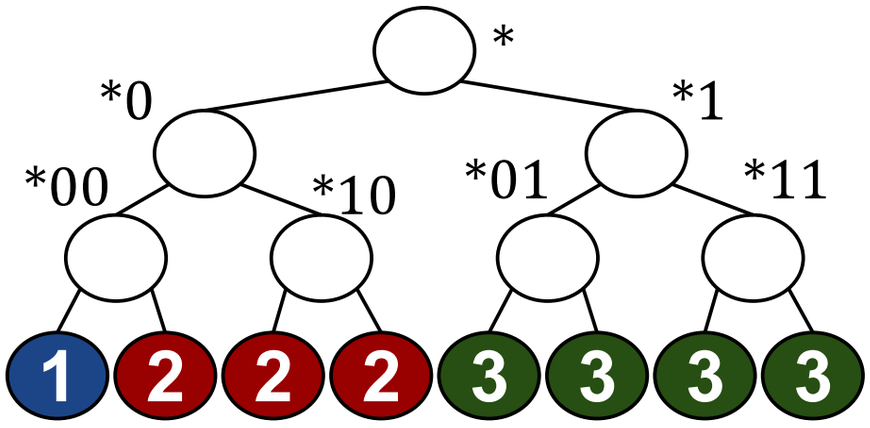


- Uniform traffic distribution
  - e.g., *000 represents 1/8 traffic
- "*Approximation*" of the weight vector?
  - Header matching discretizes portions of traffic
  - Use error bound to quantify approximations
- $1/3 \approx 1/8 + 1/4$

Match	*Action*
***100**	***Forward to 1***
***10**	***Forward to 1***

Naïve solution

- Bin pack suffixes
  - Round weights to multiples of $1/2^k$
  - When k = 3, (1/6, 1/3, 1/2) ≈ (1/8, 3/8, 4/8)



**000*	*Fwd to 1*
**100*	*Fwd to 2*
**10*	*Fwd to 2*
**1*	*Fwd to 3*

- Observation
  - 1/3 ≈ 3/8 = **1/2 - 1/8** saves one rule
  - Use ***subtraction*** and ***rule priority***

**000*	*Fwd to 1*
**0*	*Fwd to 2*
***	*Fwd to 3*

Approximation with $1/2^k$

- Approximate a weight with powers-of-two terms – 1/2, 1/4, 1/8, …
- Start with

#	Weight w	Approx v	Error v - w
1	1/6	0	-1/6
2	1/3	0	-1/3
3	1/2	1	1/2

Under-approximated

Over-approximated

Approximation with $1/2^k$

- Reduce errors i*teratively*
- In each round, move $1/2^k$ from an over-approximated weight to an under-approximation weight

#	Weight w	Approx v	Error v - w
1	1/6	0	-1/6
2	1/3	0	-1/3
3	1/2	1	1/2

Under-approximated

Over-approximated

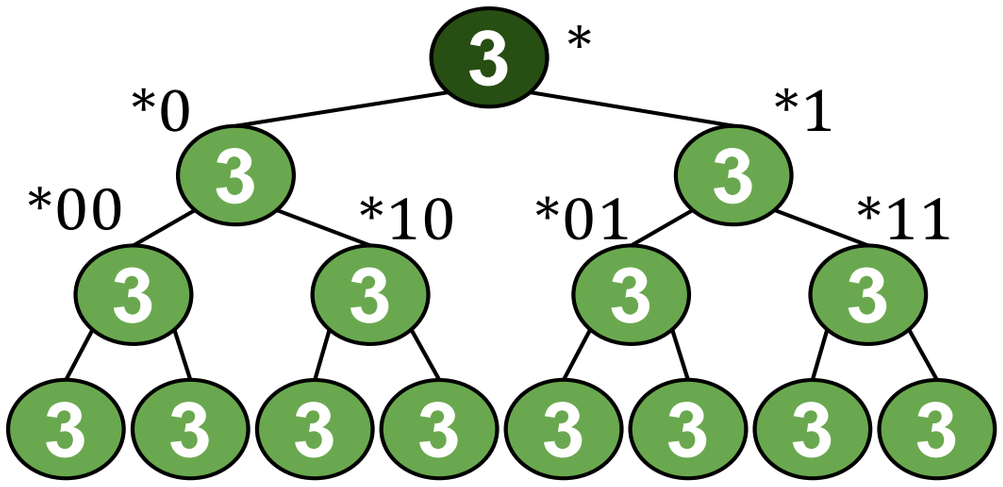
move 1/2

1	1/6	0	-1/6
2	1/3	1/2	-1/3 + 1/2 = 1/6
3	1/2	1 - 1/2	1/2 - 1/2 = 0

Initial approximation

#	Weight	Approx	Error
1	1/6	0	-1/6
2	1/3	0	-1/3
3	1/2	1	1/2

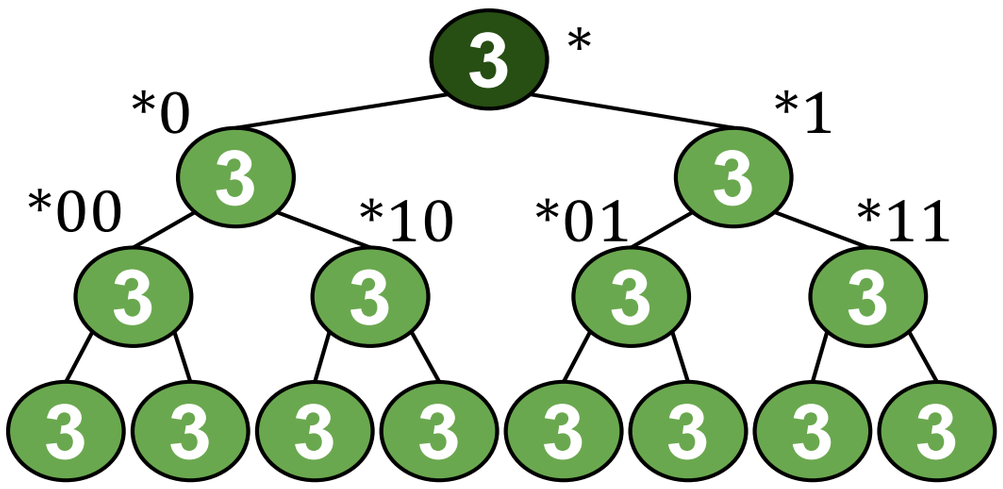
*	Fwd to 3

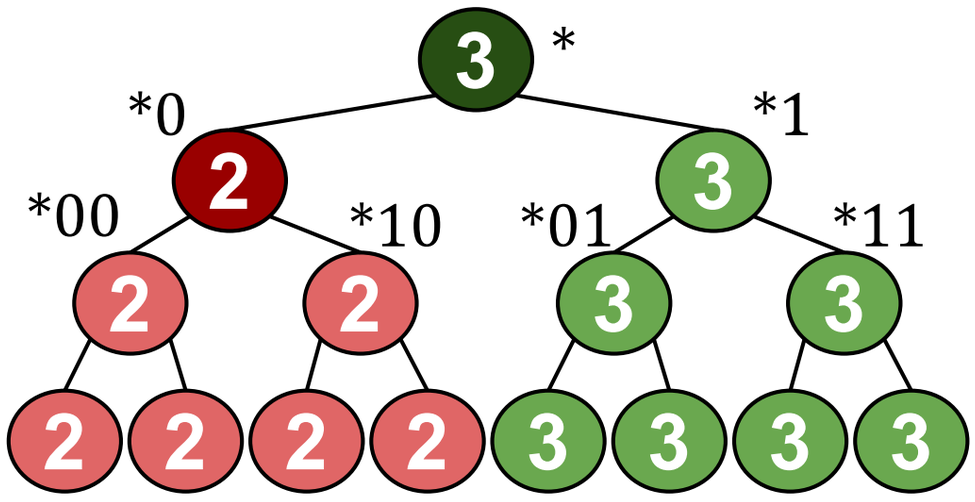


Move 1/2 from W_3 to W_2

#	Weight	Approx	Error
1	1/6	0	-1/6
2	1/3	1/2	1/6
3	1/2	1 -1/2	0

*0	Fwd to 2
*	Fwd to 3

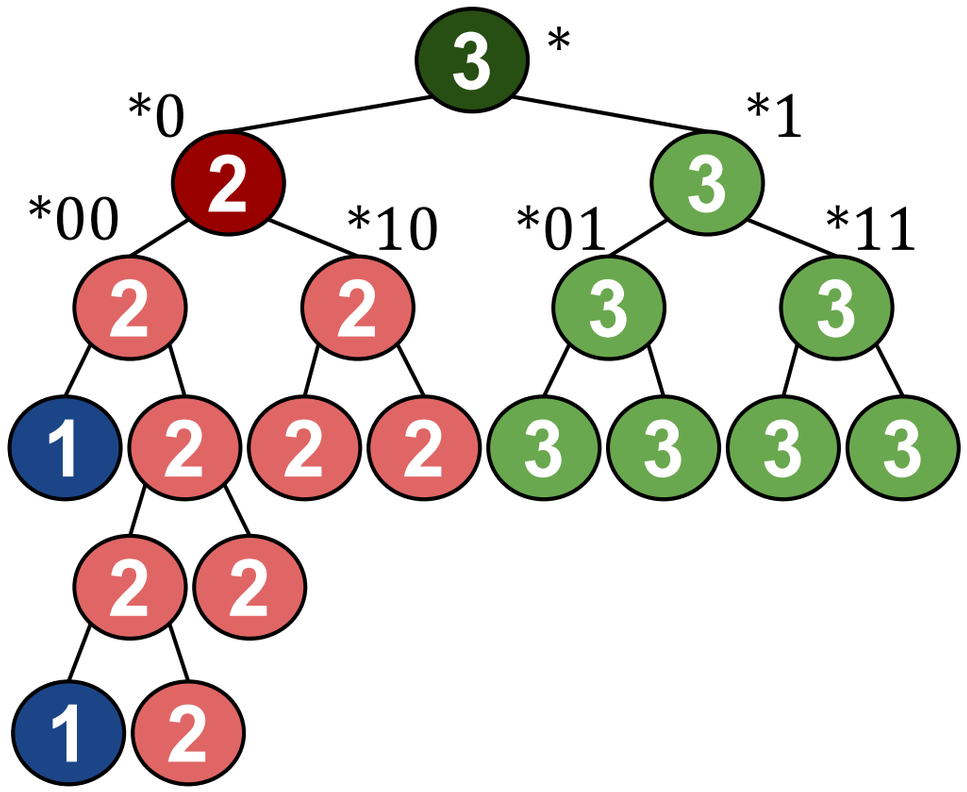




Final result

#	Weight	Approx
1	1/6	1/8 +1/32
2	1/3	1/2 -1/8 -1/32
3	1/2	1 -1/2

*00100	Fwd to 1
*000	Fwd to 1
*0	Fwd to 2
*	Fwd to 3



Reduce errors exponentially!

Truncation for less rules

- Limited rule-table size?
  - Truncation, i.e., stop iterations earlier
- Imbalance: $\Sigma\ |error_i| / 2$
  - Total over-approximation

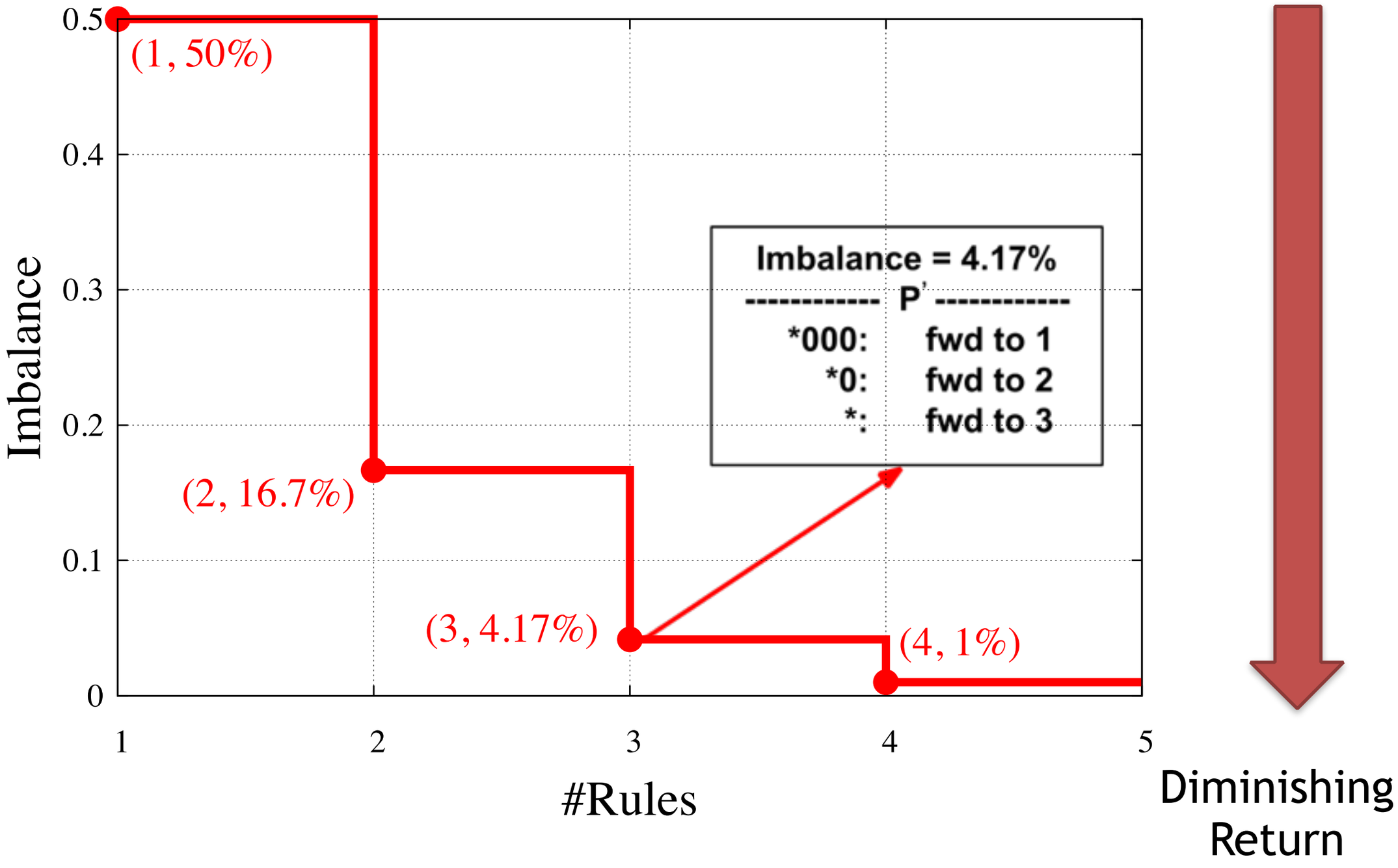
**00100*	*Fwd to 1*
**000*	*Fwd to 1*
**0*	*Fwd to 2*
***	*Fwd to 3*

Full rules
Imbalance = 1%

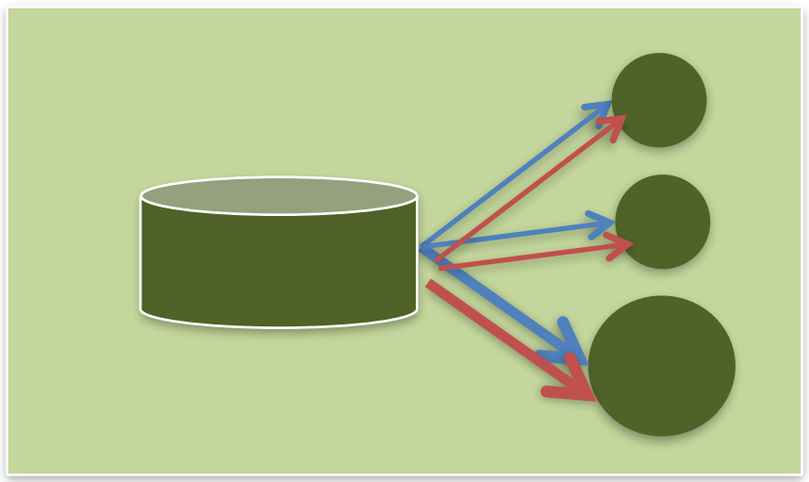
**000*	*Fwd to 1*
**0*	*Fwd to 2*
***	*Fwd to 3*

Rules after truncation
Imbalance = 4%

Stairstep: #Rules v.s. Imbalance



Multiple VIPs



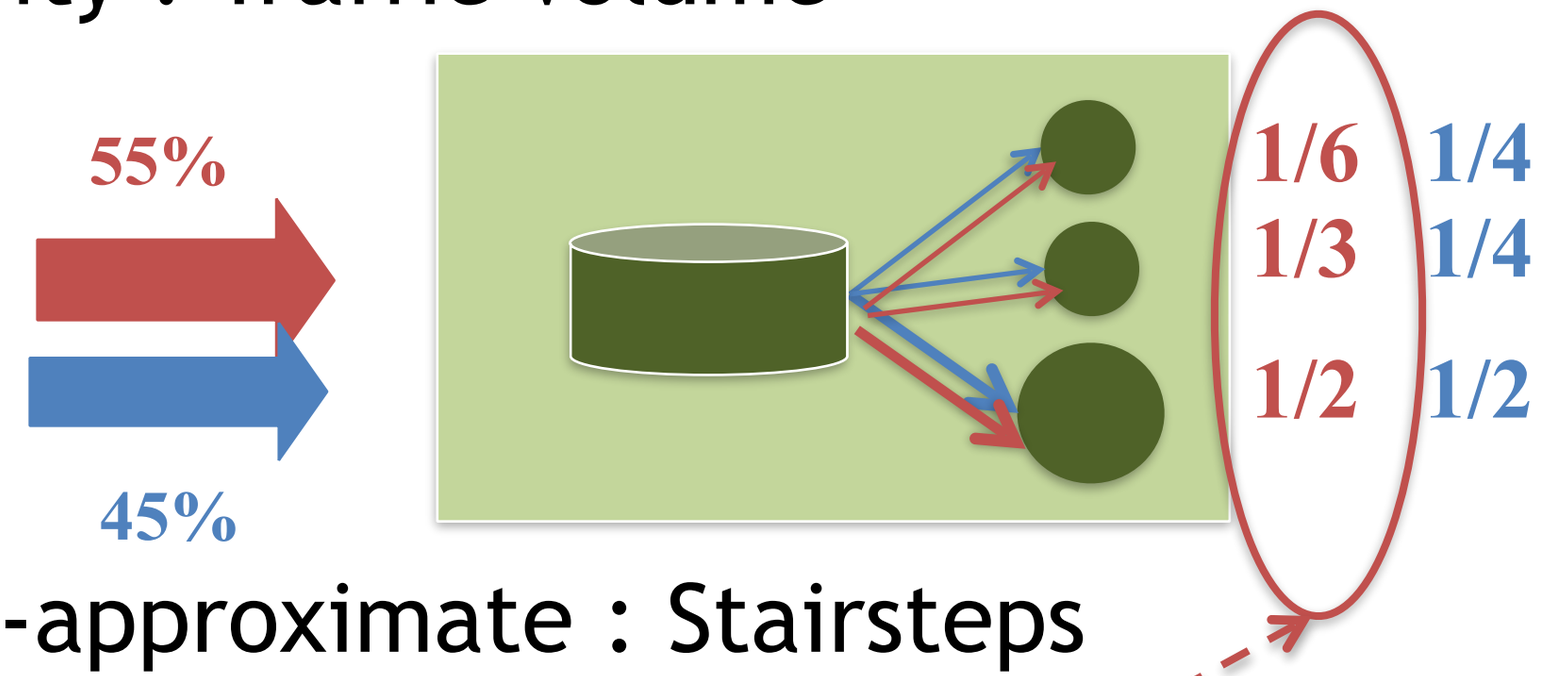
1/6 1/4
1/3 1/4
1/2 1/2

- How rules achieve the weight vector of a VIP?
  - Arbitrary weights
  - Non-uniform traffic distribution over flowspace
- How VIPs (100-10k) share a rule table (~4,000)?

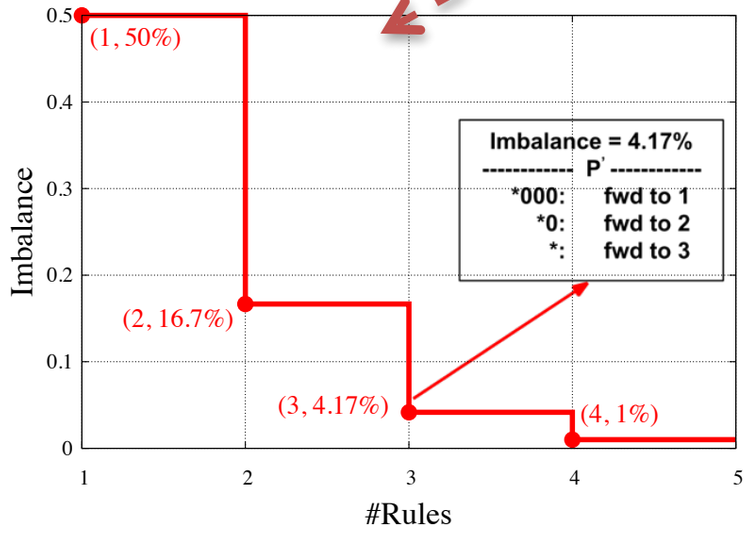
Minimize $\Sigma\ traffic\_volume_j \times \Sigma\ |error_{ij}| / 2$

Characteristics of VIPs

- Popularity : Traffic Volume



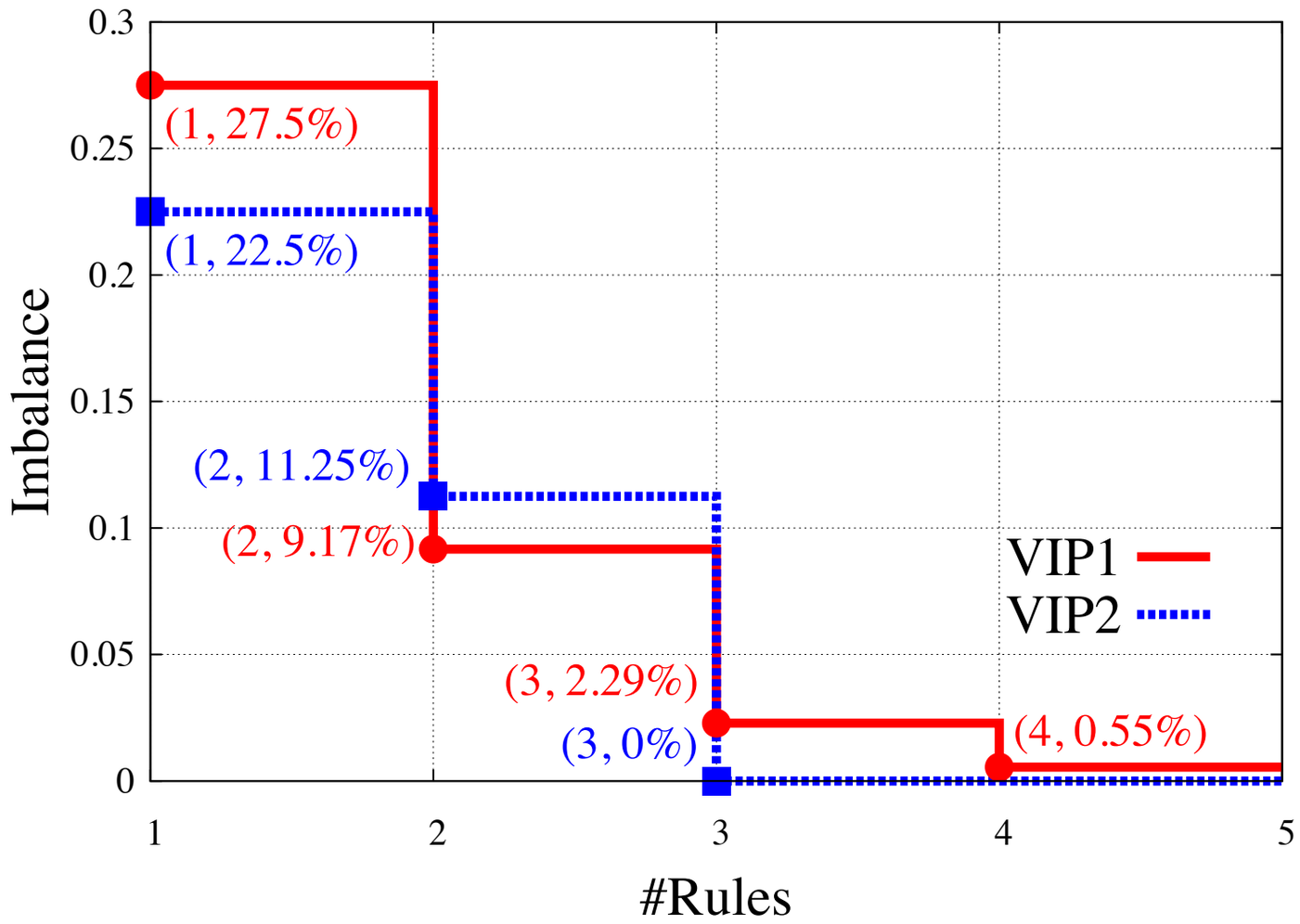
- Easy-to-approximate : Stairsteps



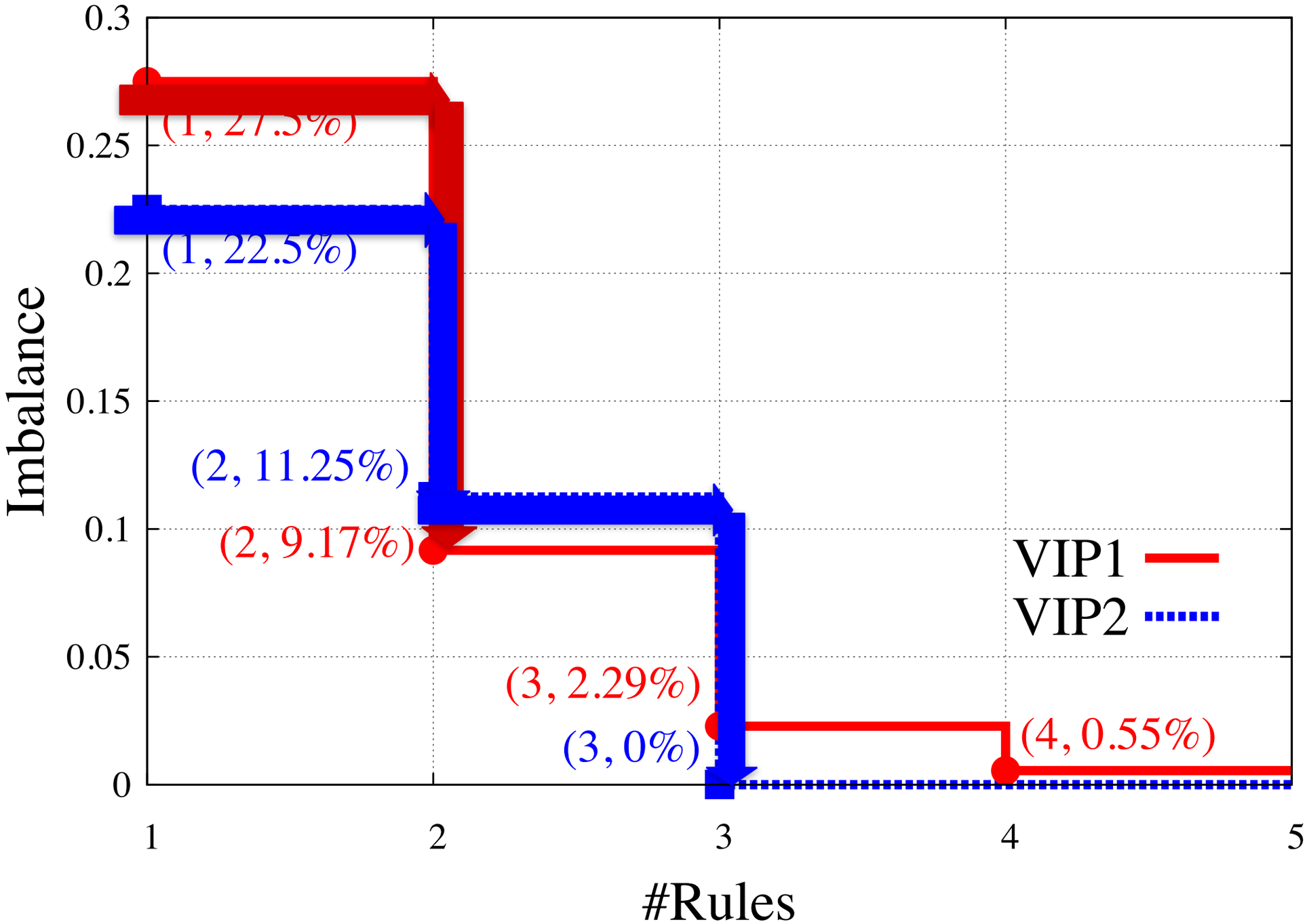
Stairsteps

- Each stairstep is scaled by its traffic volume

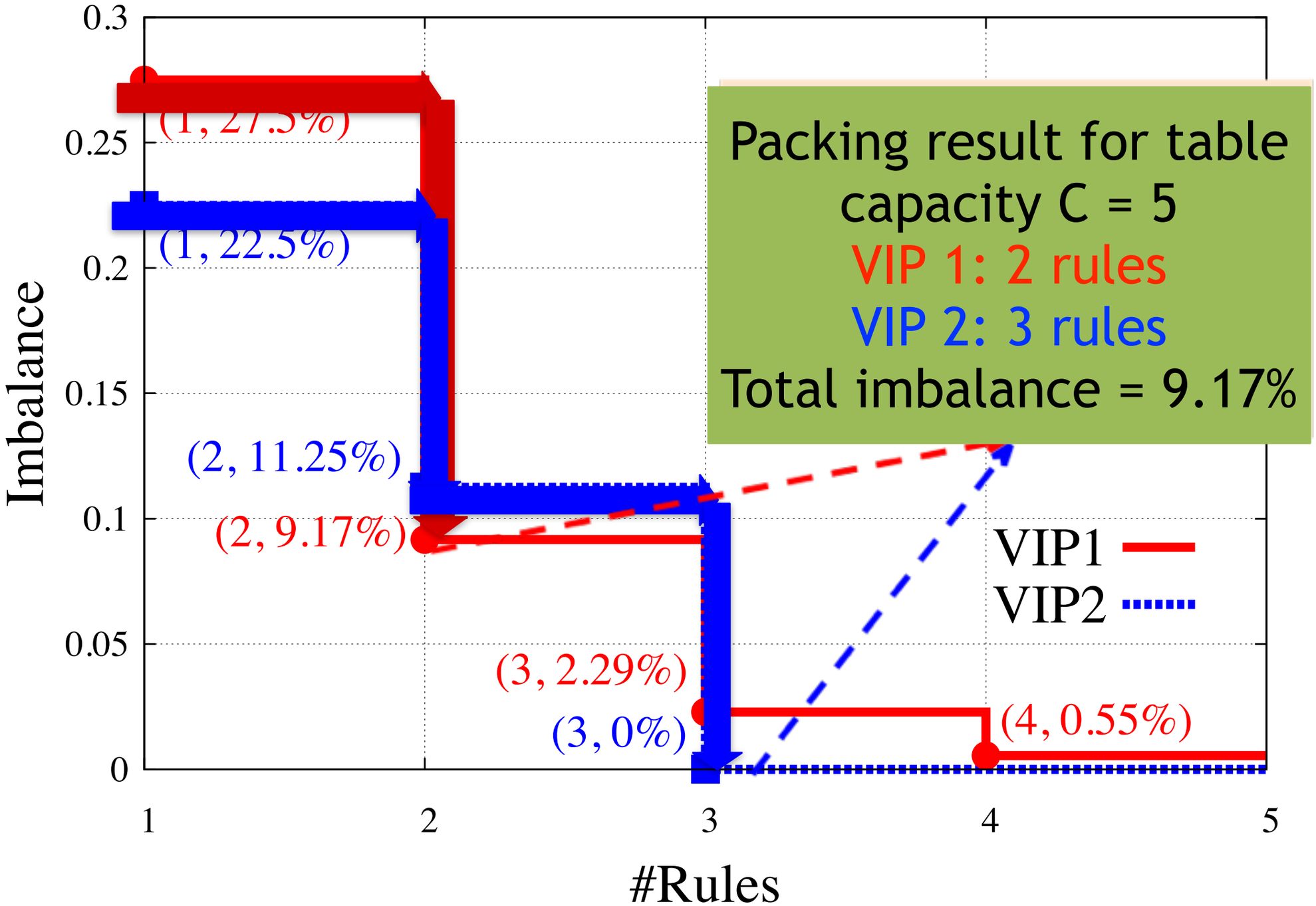
VIP_1	55%	(1/6, 1/3, 1/2)
VIP_2	45%	(1/4, 1/4, 1/2)



Rule allocation



Rule allocation



Pack Result

Packing result for table
capacity C = 5
VIP 1: 2 rules
VIP 2: 3 rules
Total imbalance = 9.17%

Match (dst, src)	*Action*
*VIP 1, *0*	*Fwd to 2*
*VIP 1, * *	*Fwd to 3*
*VIP 2, *00*	*Fwd to 1*
*VIP 2, *01*	*Fwd to 2*
*VIP 2, * *	*Fwd to 3*

Sharing default rules

- Build default split for ALL VIPs

Weights		=	Default		+	Delta	
1/6	1/4		0			1/6	1/4
1/3	1/4		1/2			-1/6	-1/4
1/2	1/2		1/2			0	0

VIP 1, *0	Fwd to 2
VIP 1, *	Fwd to 3
VIP 2, *00	Fwd to 1
VIP 2, *01	Fwd to 2
VIP 2, *	Fwd to 3

Imbalance = 9.17%

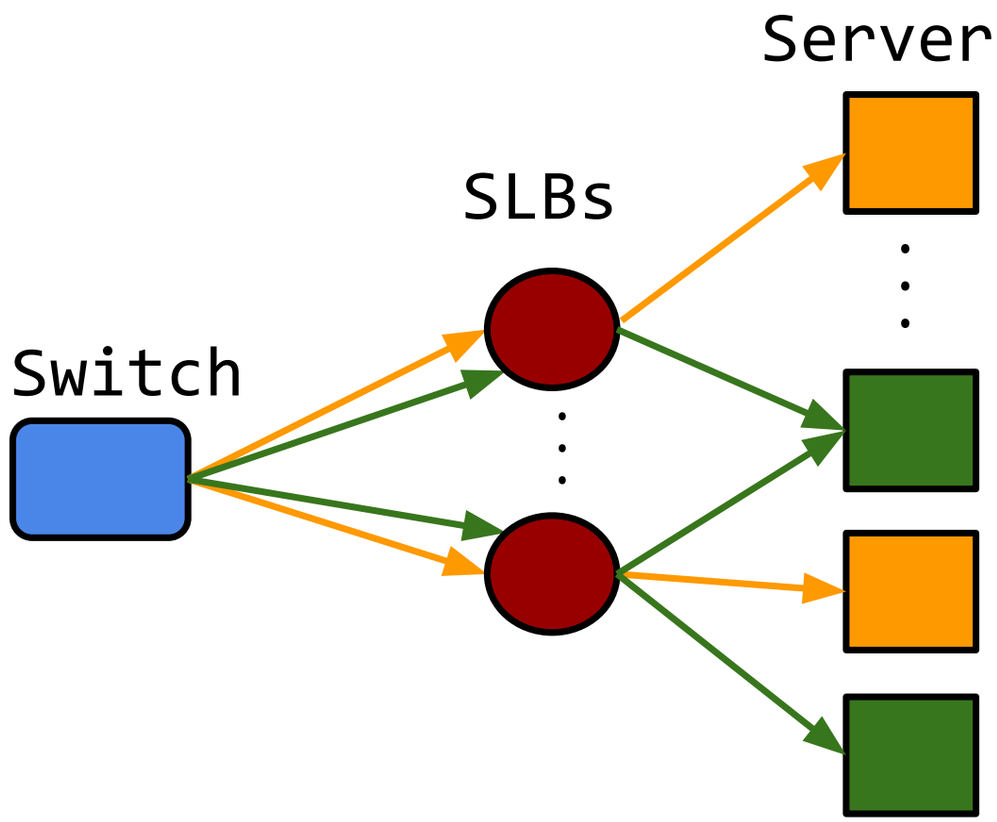
V.S.

VIP 1, *00100	Fwd to 1
VIP 1, *000	Fwd to 1
VIP 2, *00	Fwd to 1
*0	Fwd to 2
*	Fwd to 3

Imbalance = 0.55%

Evaluation : datacenter network

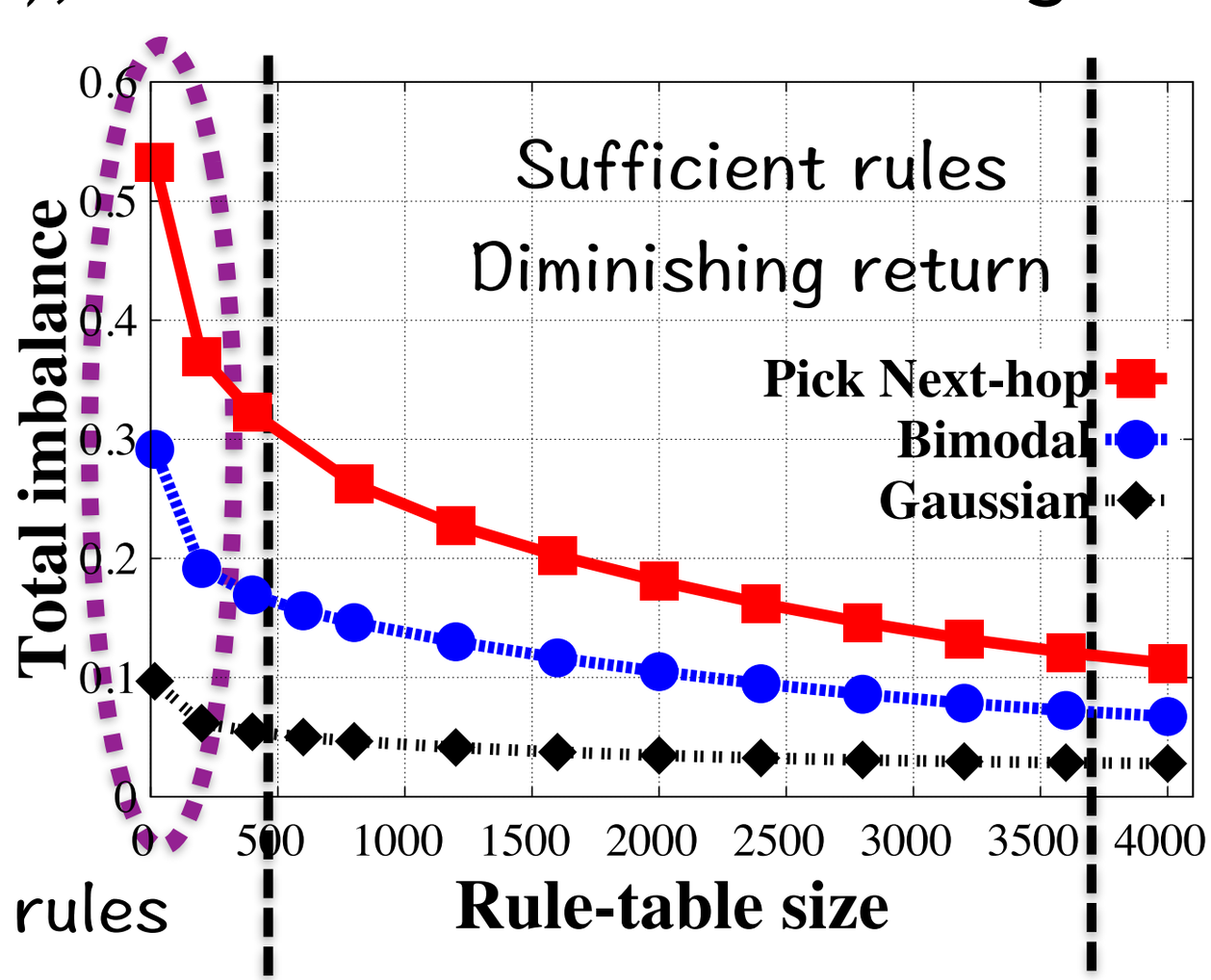
- Synthetic VIP distributions
- LBer switch connects to SLBs



Load Balance 10,000 VIPs

- Weights
  - Gaussian: equal weights
  - Bimodal: big (4x) and small weights
  - Pick_Next-hop: big(4x), small and zero-value weights
  - 16 weights per VIP

Imbalance of ECMP



Insufficient rules

Take-aways

- Wildcard matches approximate weights well
  - Exponential drop in errors
- Prioritized packing reduces imbalance sharply
- Default rules serve as a good starting point

- Refer to our full paper for
  - Multiple VIP Grouping
  - Incremental update to reduce “churn”
  - Niagara for multi-pathing

Thanks!