eXplicit Path Control in Commodity Data Centers: Design and Applications

[1]Shuihai Hu, [1]Kai Chen, [2]Haitao Wu, [1]Wei Bai, [3]Chang Lan, [1]Hao Wang, [4]Hongze Zhao, [2]Chuanxiong Guo

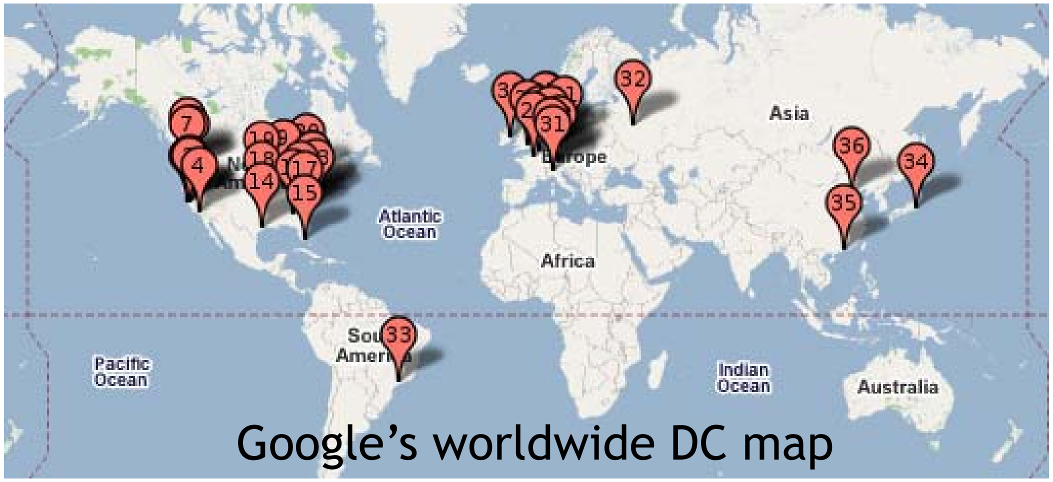
[1]Sing Group @ Hong Kong University of Science and Technology, [2]Microsoft, [3]UC Berkeley, [4]Duke University

USENIX NSDI 2015
Oakland, CA

Data centers around the world

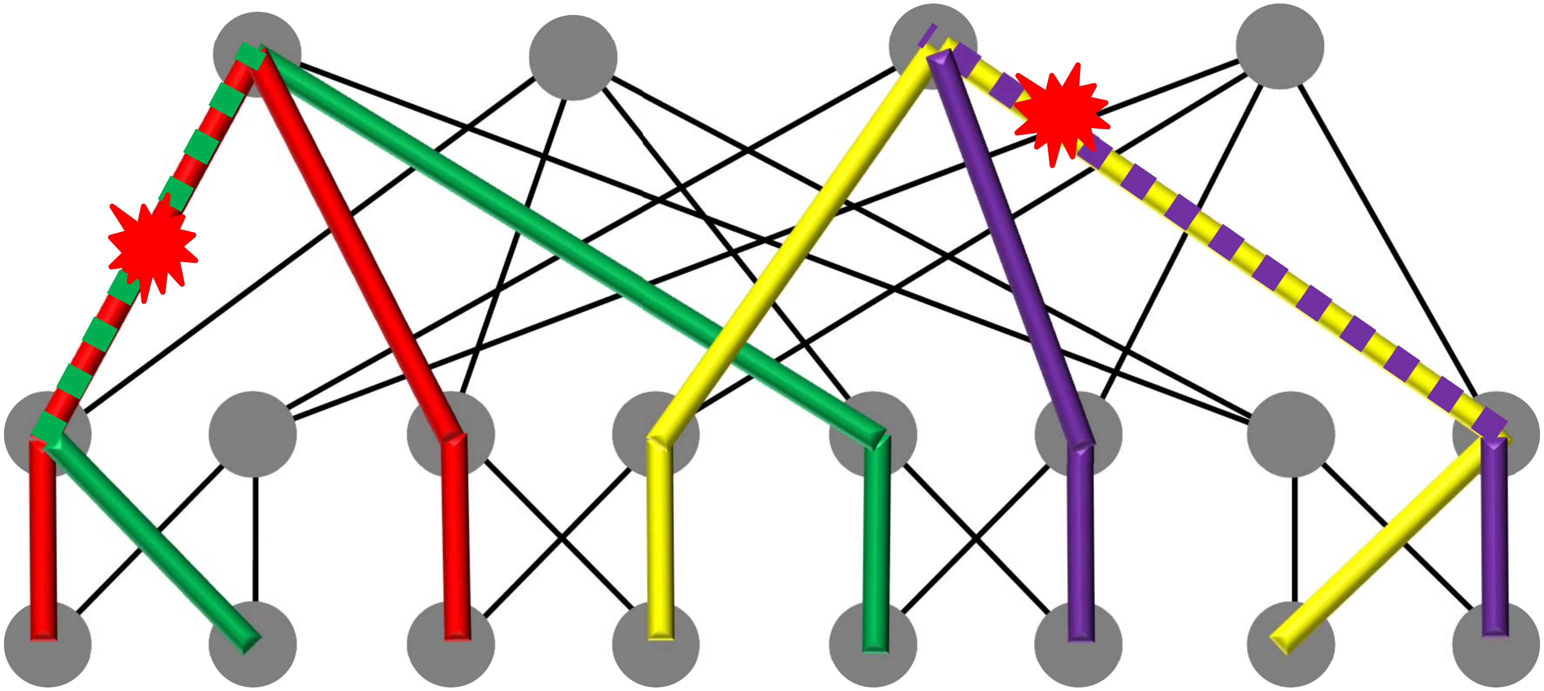
Facebook DC interior

Microsoft’s Chicago DC



Google’s worldwide DC map

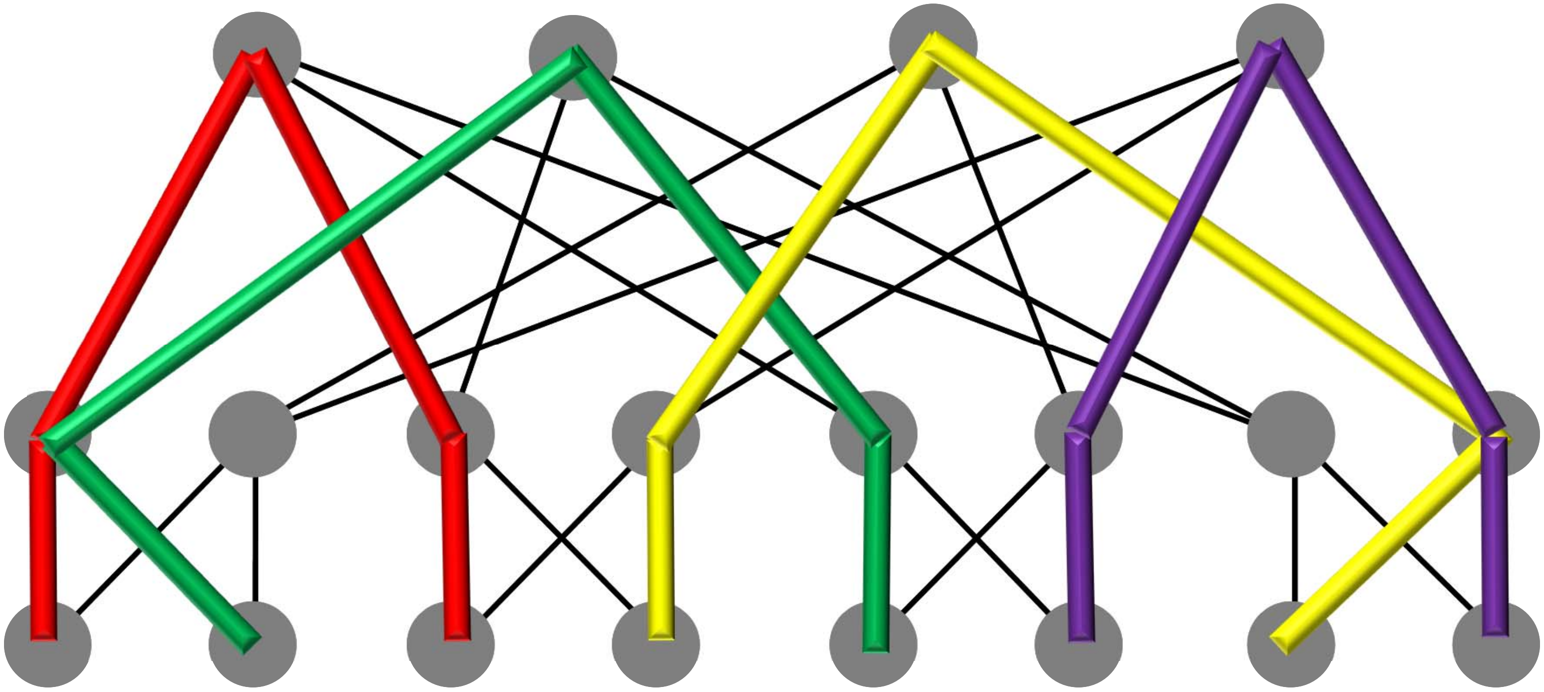
Multi-path and ECMP



- State-of-the-art ECMP
  - Forward packets based on hash of headers
  - Flows take randomized, implicit paths
  - On average, over 60% bandwidth waste due to path collision (Hedera [NSDI'10])

Flow 1 🟥
Flow 2 🟩
Flow 3 🟨
Flow 4 🟪

eXplicit Path Control

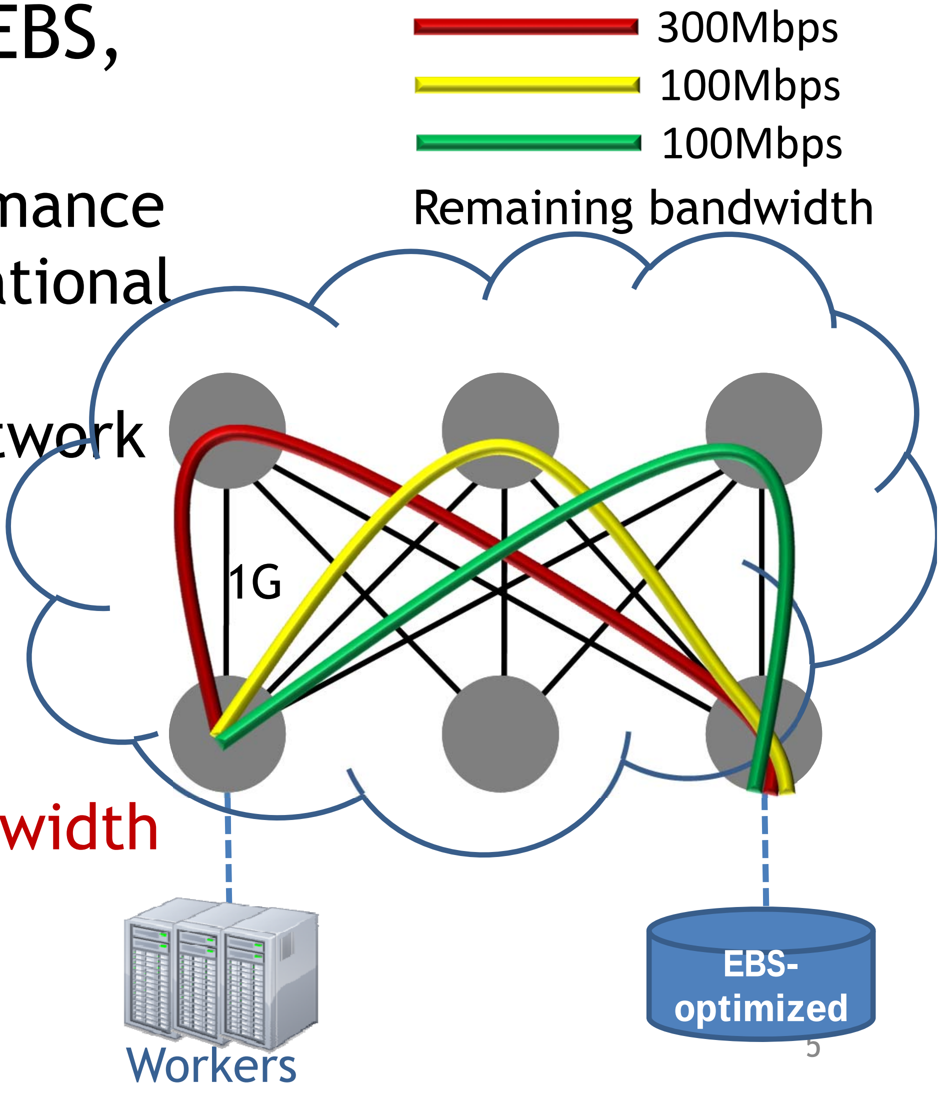


To fully utilize network, we must explicitly control paths for flows

Flow 1 🟥
Flow 2 🟩
Flow 3 🟨
Flow 4 🟪

The case for explicit path control (#1)

- Provisioned IOPS (Amazon EBS, Azure Premium Storage)
  - Deliver predictable performance for I/O intensive apps, relational DBs
  - Must provide necessary network bandwidth guarantee

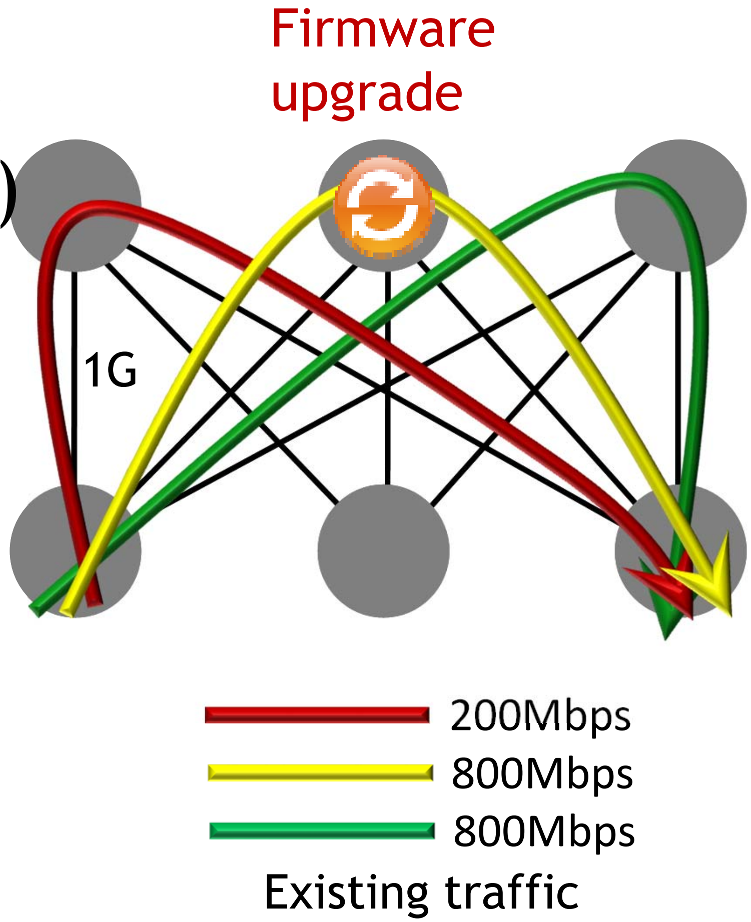


Explicit path control makes bandwidth guarantee easier to implement

The case for explicit path control (#2)

- DC network updates (zUpdate [Sigcomm'13], Dionysus [Sigcomm'14])
  - Congestion-free
  - Loop-free
  - …

Explicit path control makes DC network updates easier to conduct

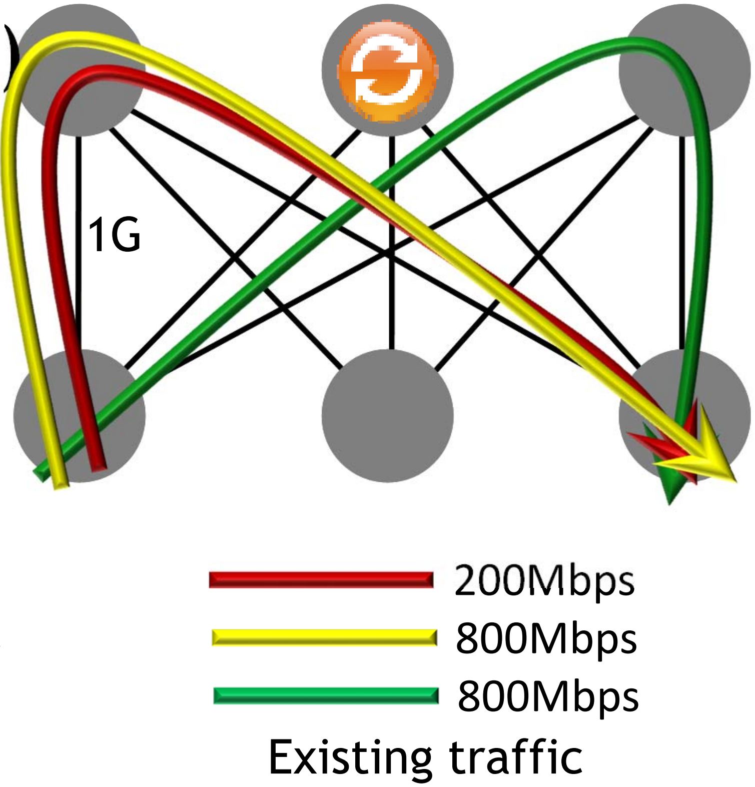


The case for explicit path control (#2)

- DC network updates (zUpdate [Sigcomm’13], Dionysus [Sigcomm’14])
  - Congestion-free
  - Loop-free
  - …

Explicit path control makes DC network updates easier to conduct

Upgrade → Reboot



🟥 200Mbps

🟨 800Mbps

🟩 800Mbps

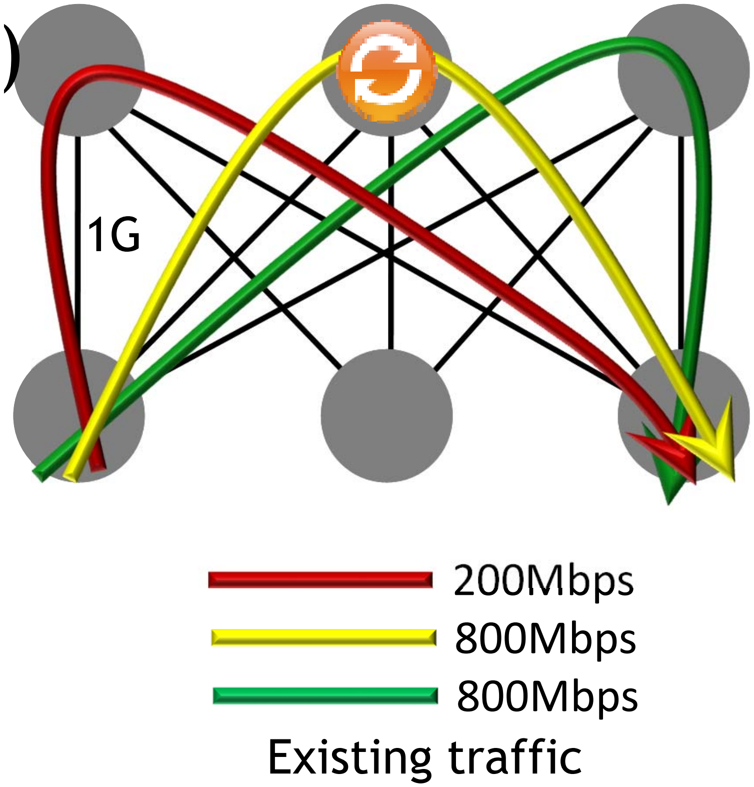
Existing traffic

The case for explicit path control (#2)

- DC network updates (zUpdate [Sigcomm’13], Dionysus [Sigcomm’14])
  - Congestion-free
  - Loop-free
  - …

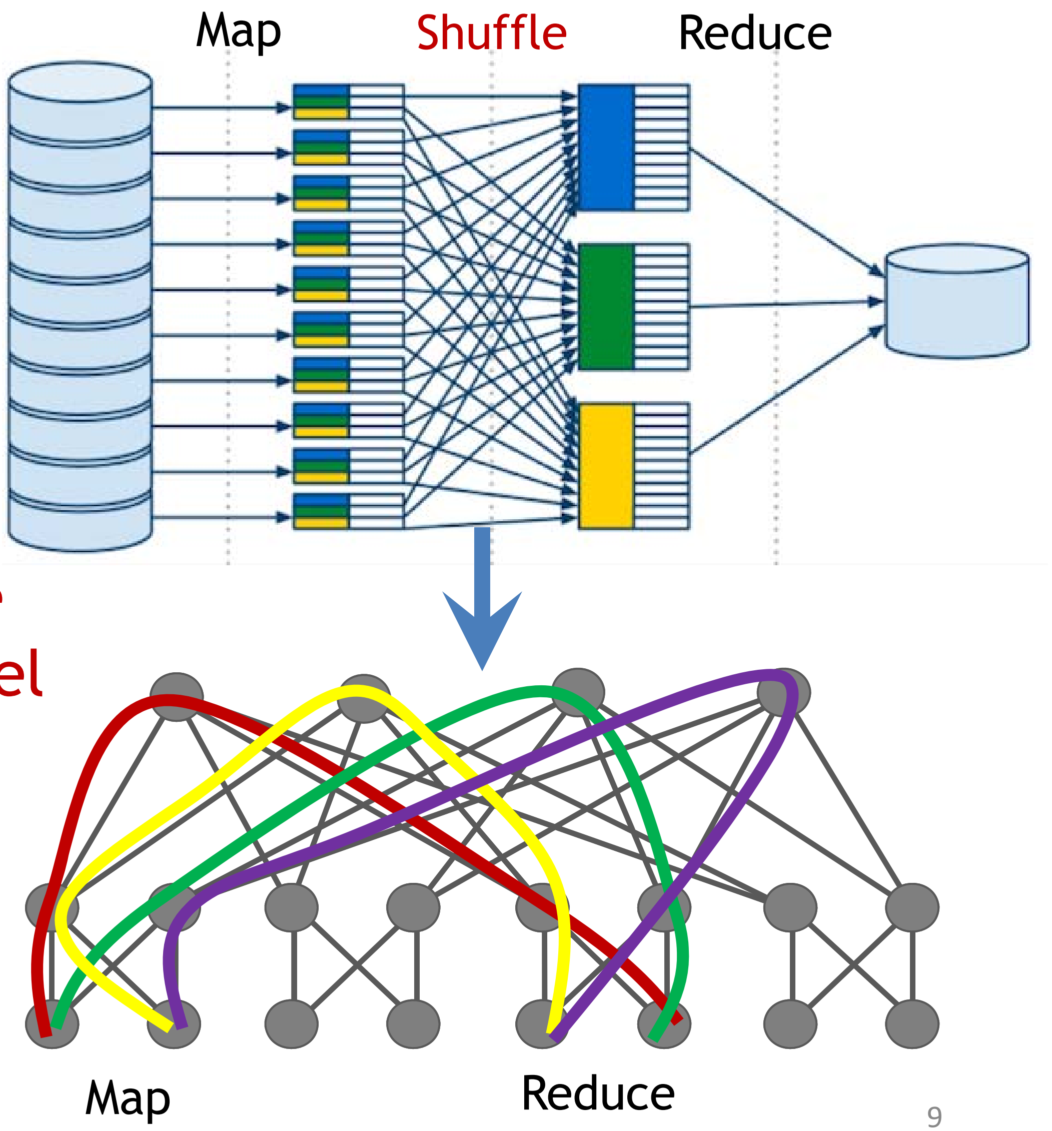
Explicit path control makes DC network updates easier to conduct

Upgrade completed



The case for explicit path control (#3)

- Map-reduce/Hadoop applications
  - Shuffle stage stresses network, requires full bisection bandwidth



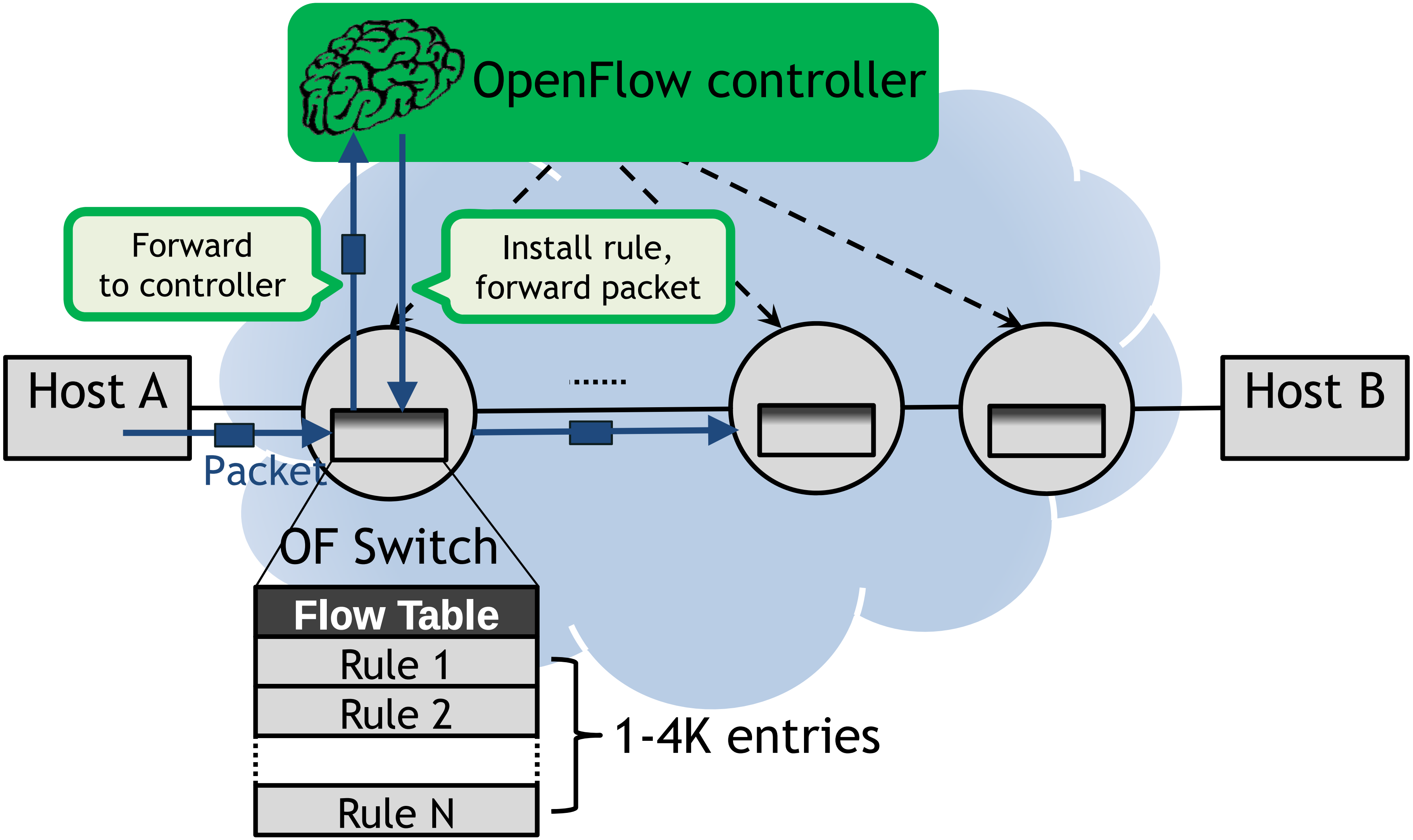
Explicit path control can be leveraged to arrange parallel paths for shuffling

Still many other cases …

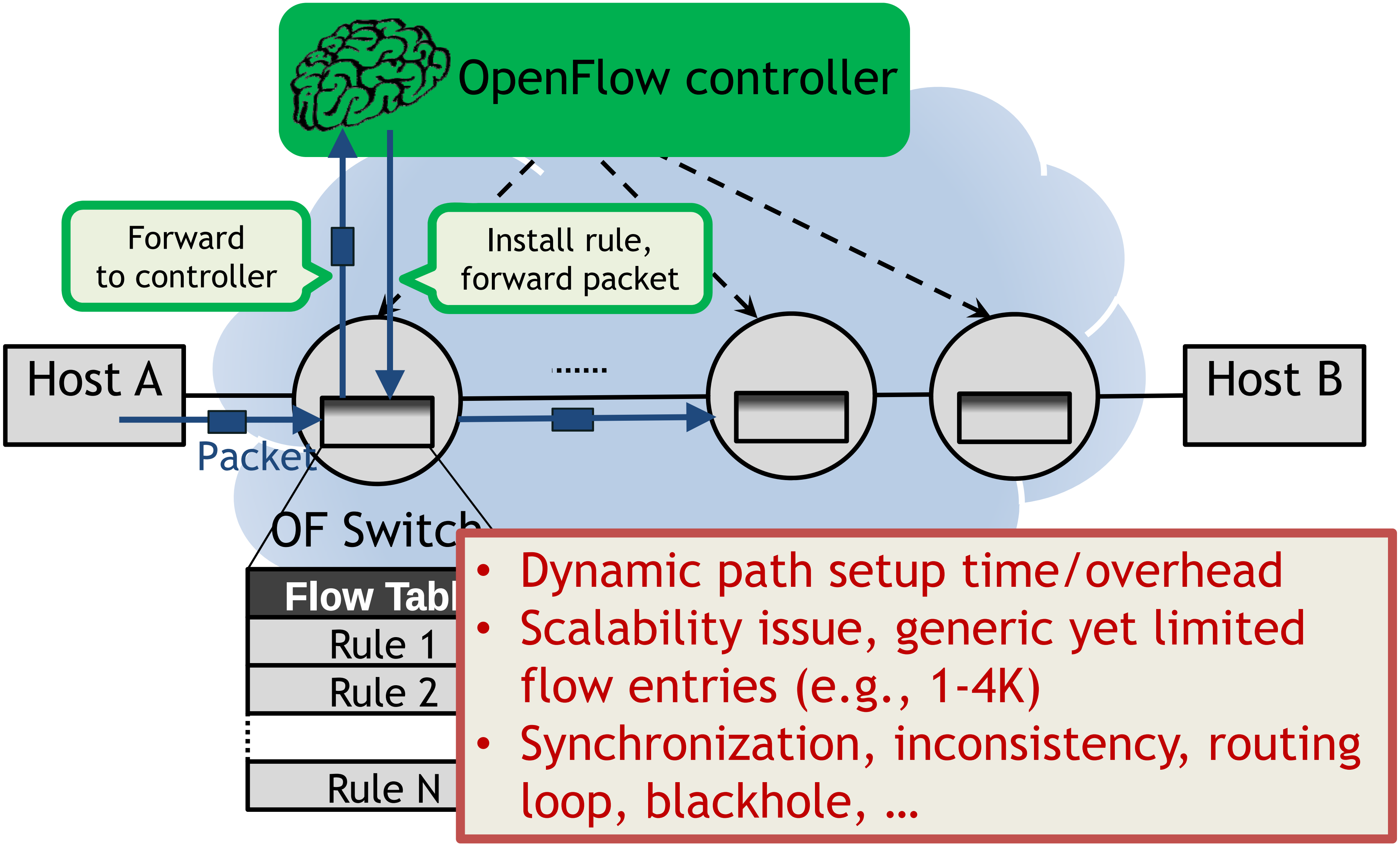
- Traffic engineering
  - e.g., MicroTE [CoNEXT'11], B4/SWAN [Sigcomm'13]
- Flow scheduling or packet scheduling
  - e.g., Hedera [NSDI'10], Fastpass [Sigcomm'14]
- Multiple path congestion control
  - e.g., MPTCP [Sigcomm'11], XMP [CoNEXT'13]
- Network virtualization and bandwidth guarantees
  - e.g., SecondNet [CoNEXT'10], Oktopus [Sigcomm'11], TIVC [Sigcomm'12], CloudMirror [Sigcomm'14]
- Power saving
  - e.g., ElasticTree [NSDI'10]
- Network diagnosis and failure handling
  - e.g., NetPilot [Sigcomm'12]
- …

All require or benefit from explicit path control

OpenFlow-enabled (dynamic) implementation

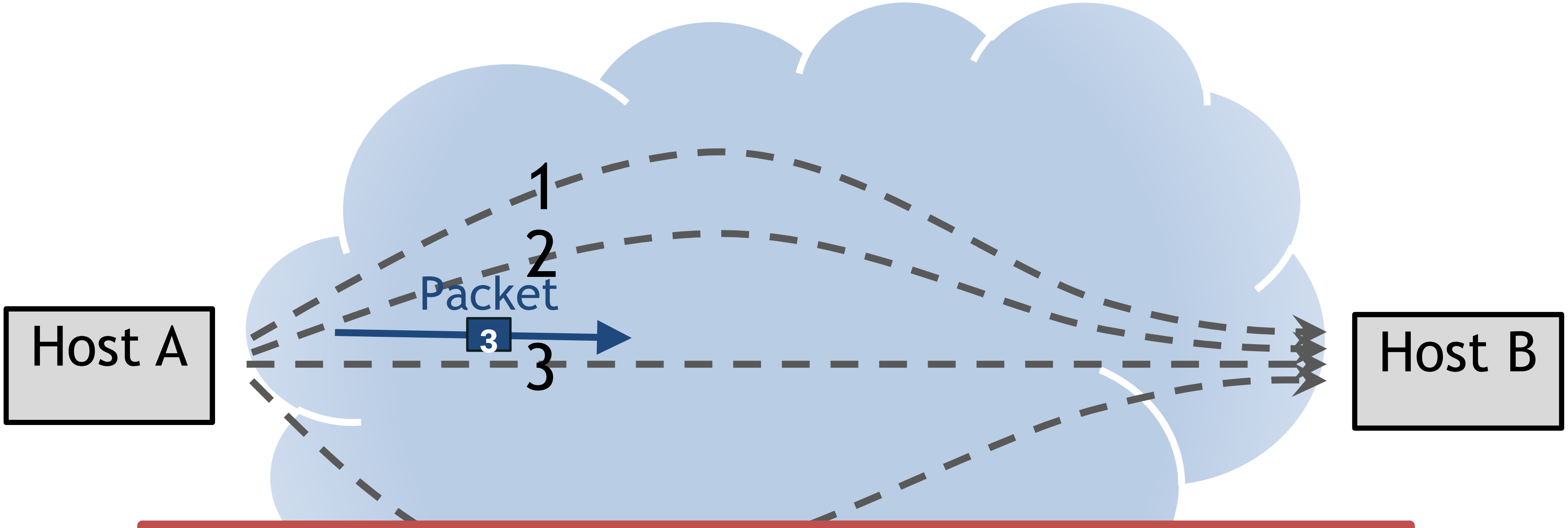


OpenFlow-enabled (dynamic) implementation



- Dynamic path setup time/overhead
- Scalability issue, generic yet limited flow entries (e.g., 1-4K)
- Synchronization, inconsistency, routing loop, blackhole, …

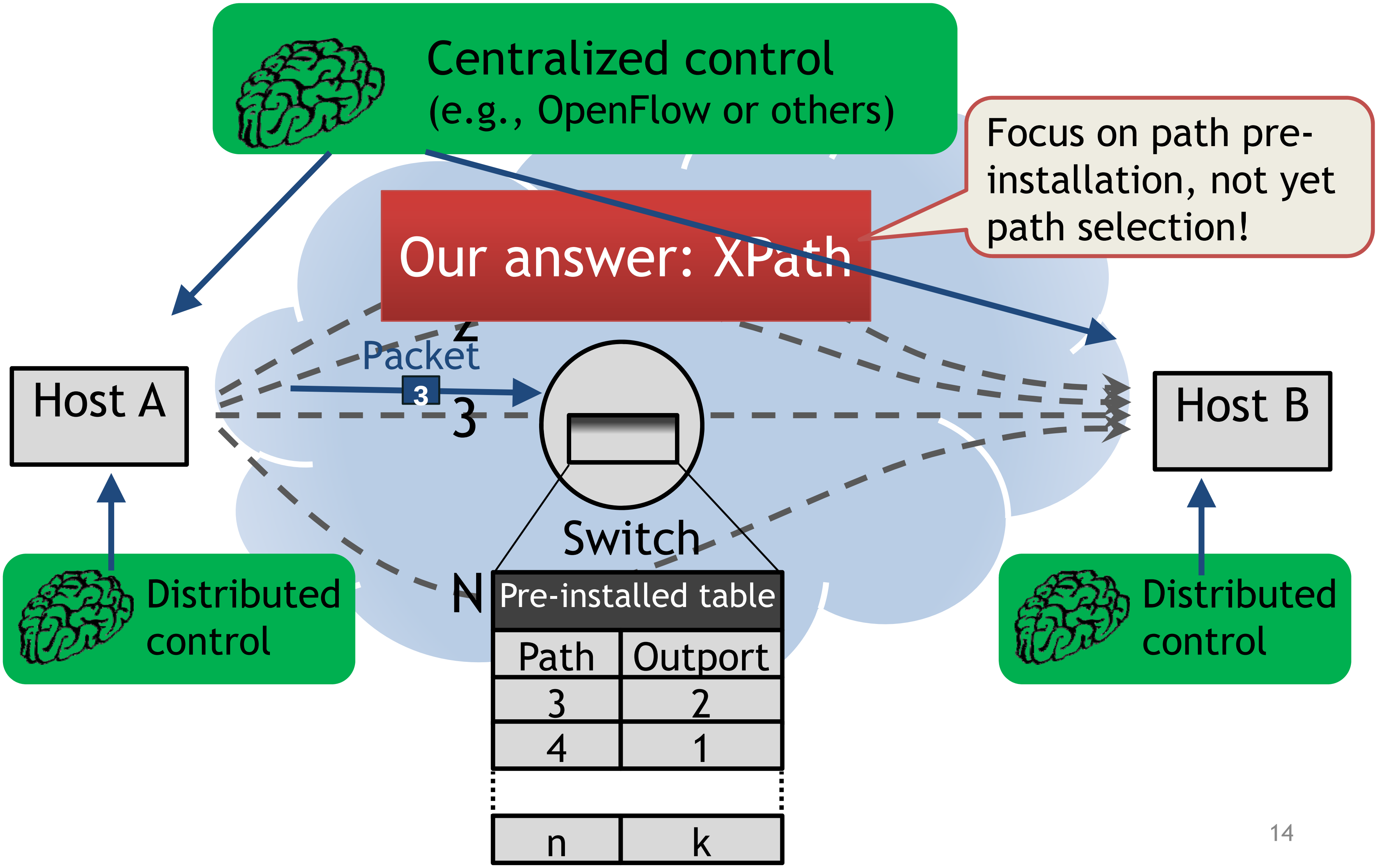
Can we pre-install all desired paths?



If yes,

- Eliminate dynamic path setup time/overhead
- Avoid synchronization/inconsistency, loop-free forwarding, no routing blackhole …
- Enable new services/applications

Can we pre-install all desired paths?



Centralized control
(e.g., OpenFlow or others)

Focus on path pre-installation, not yet path selection!

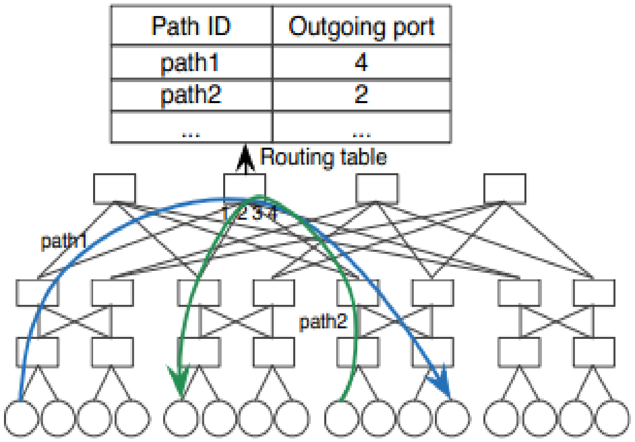
Our answer: XPath

Pre-installed table

Path	Outport
3	2
4	1
n	k

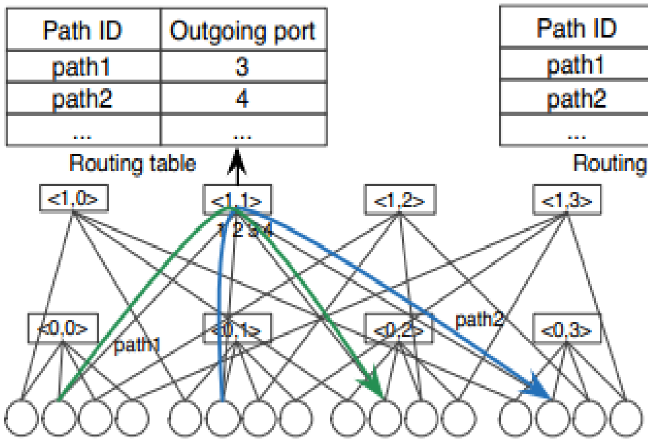
XPath Basic Idea

- Key observation motivating XPath
  - IP LPM tables in commodity switches becoming large
    - E.g., Broadcom StrataXGS Trident-II (144K)
- Natural idea of XPath
  - Leverage IP LPM table to implement explicit path control
- One sentence describing XPath
  - Explicitly identify a path with a path ID and pre-install all these IDs using IP LPM tables.



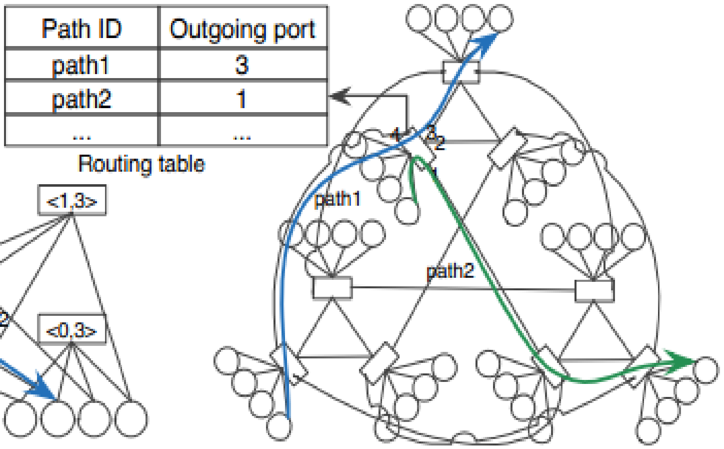
Path ID	Outgoing port
path1	4
path2	2
...	...

Fattree topology



Path ID	Outgoing port
path1	3
path2	4
...	...

BCube topology

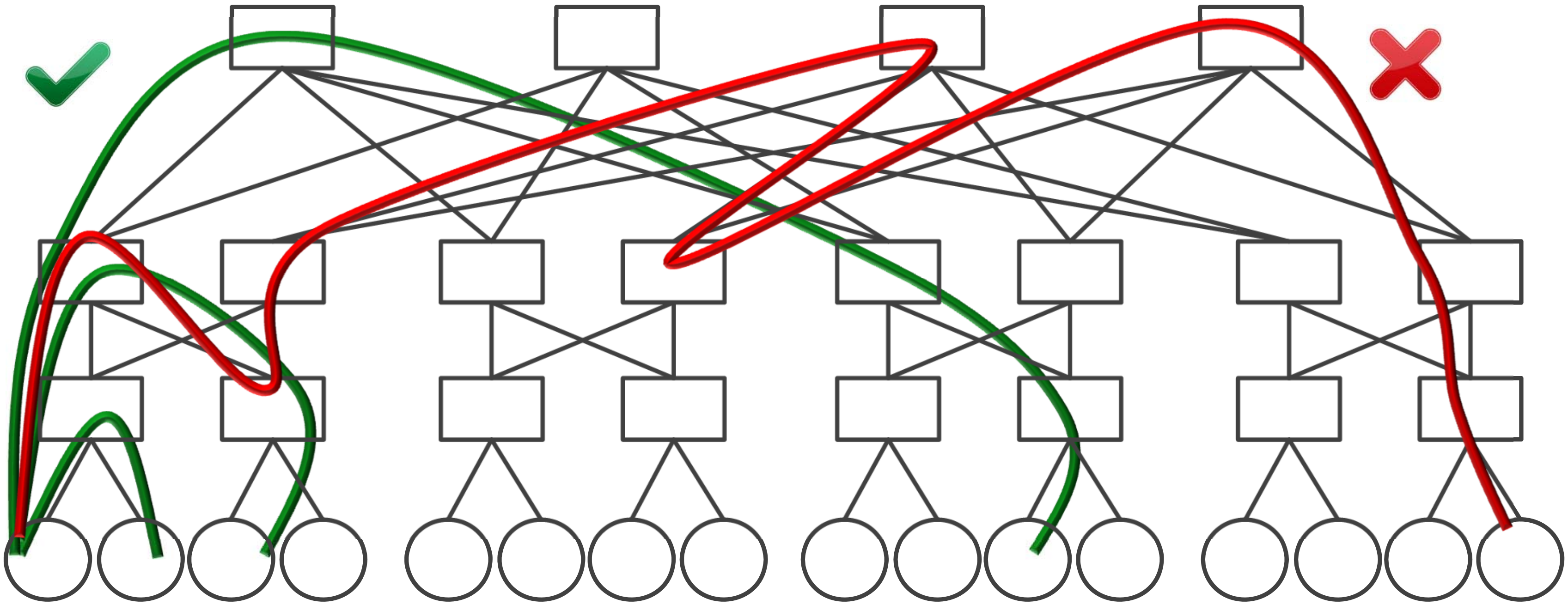


Path ID	Outgoing port
path1	3
path2	1
...	...

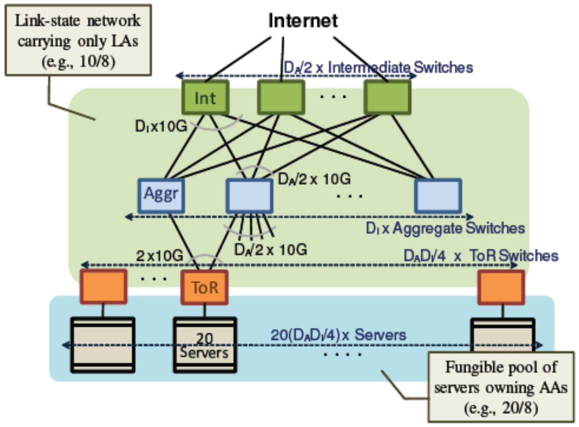
HyperX topology

XPath’s Challenges

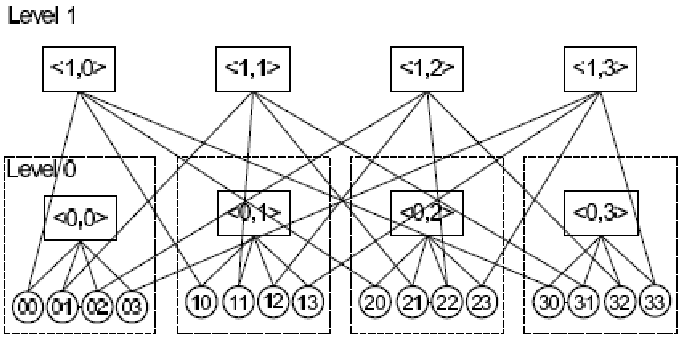
- What paths to consider?
  - Cannot enumerate all possible paths, exponential.
  - Observation: DCNs have desired paths, e.g.,
    - k-port Fattree: $k^2/4$ paths between two ToRs,
    - n-layer BCube: (n+1) paths between two servers,
    - Sufficient for high-bandwidth, fault-tolerance.
  - XPath’s first step: pre-install all these desired paths.
- How to pre-install them?
  - Desired paths # still very large
    - E.g., over 2^{32} for Fattree(64), 32-bit IP cannot express them!
- Opportunities:
  - DCN is under control
  - Two-step compression algorithm



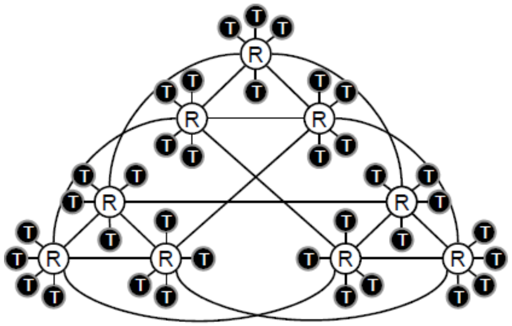
Fattree [Sigcomm'08]



VL2 [Sigcomm'09]

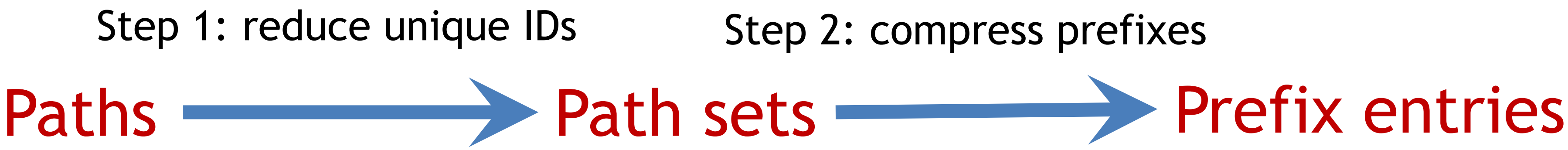


BCube [Sigcomm'09]



HyperX [SC'09]

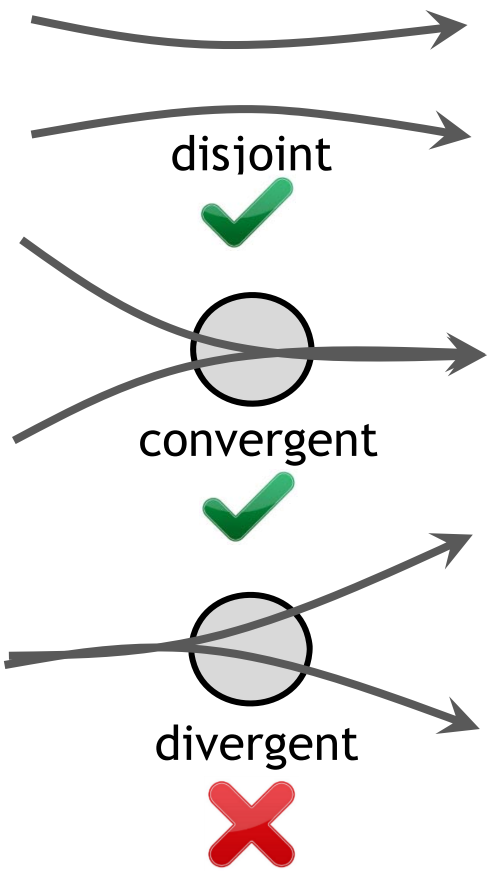
XPath’s Two-step Compression Algorithm



Step 1: reduce unique IDs

Step 2: compress prefixes

Paths → Path sets → Prefix entries



Path set	Out port	ID (bad)	ID (good)
ps$_0$	0	0	0
ps$_1$	1	1	2
ps$_2$	2	2	4
ps$_3$	0	3	1
ps$_4$	1	4	3
ps$_5$	2	5	5

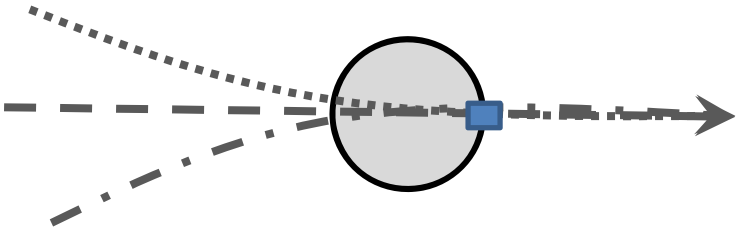
ID	Prefix	Out port
1	001	1
2	010	2
0,3	0**	0
4	100	1
5	101	2

ID	Prefix	Out port
0,1	00*	0
2,3	01*	1
4,5	1**	2

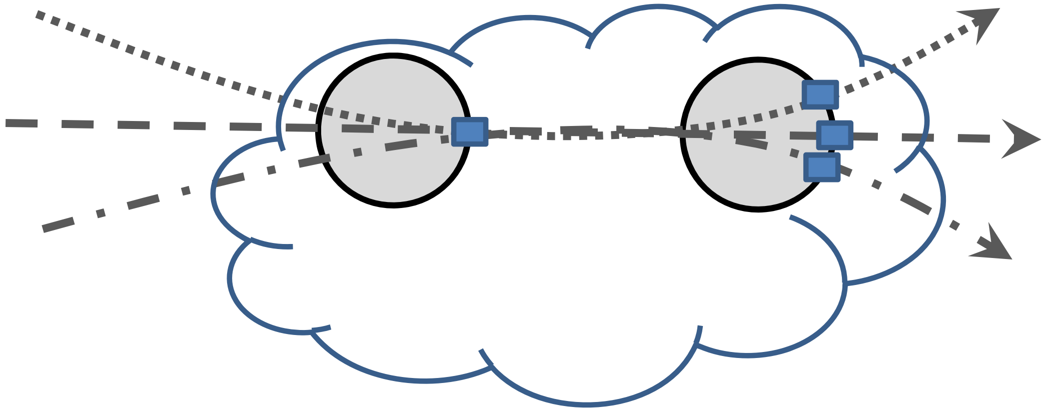
XPath’s Two-step Compression Algorithm

Step 2: compress prefixes

Path sets → Prefix entries



Simple for only one switch, just sequential encoding

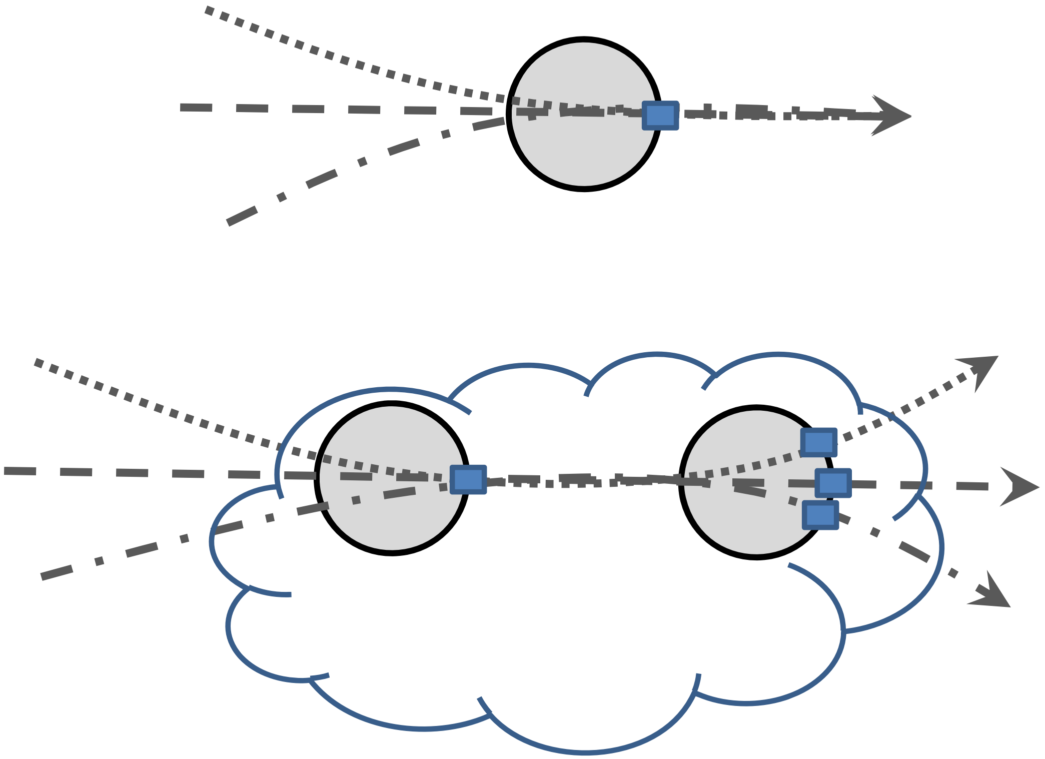


but, complex for DCN with many switches, a good ID encoding on one may be bad for another

XPath’s Two-step Compression Algorithm

Step 2: compress prefixes

Path sets → Prefix entries

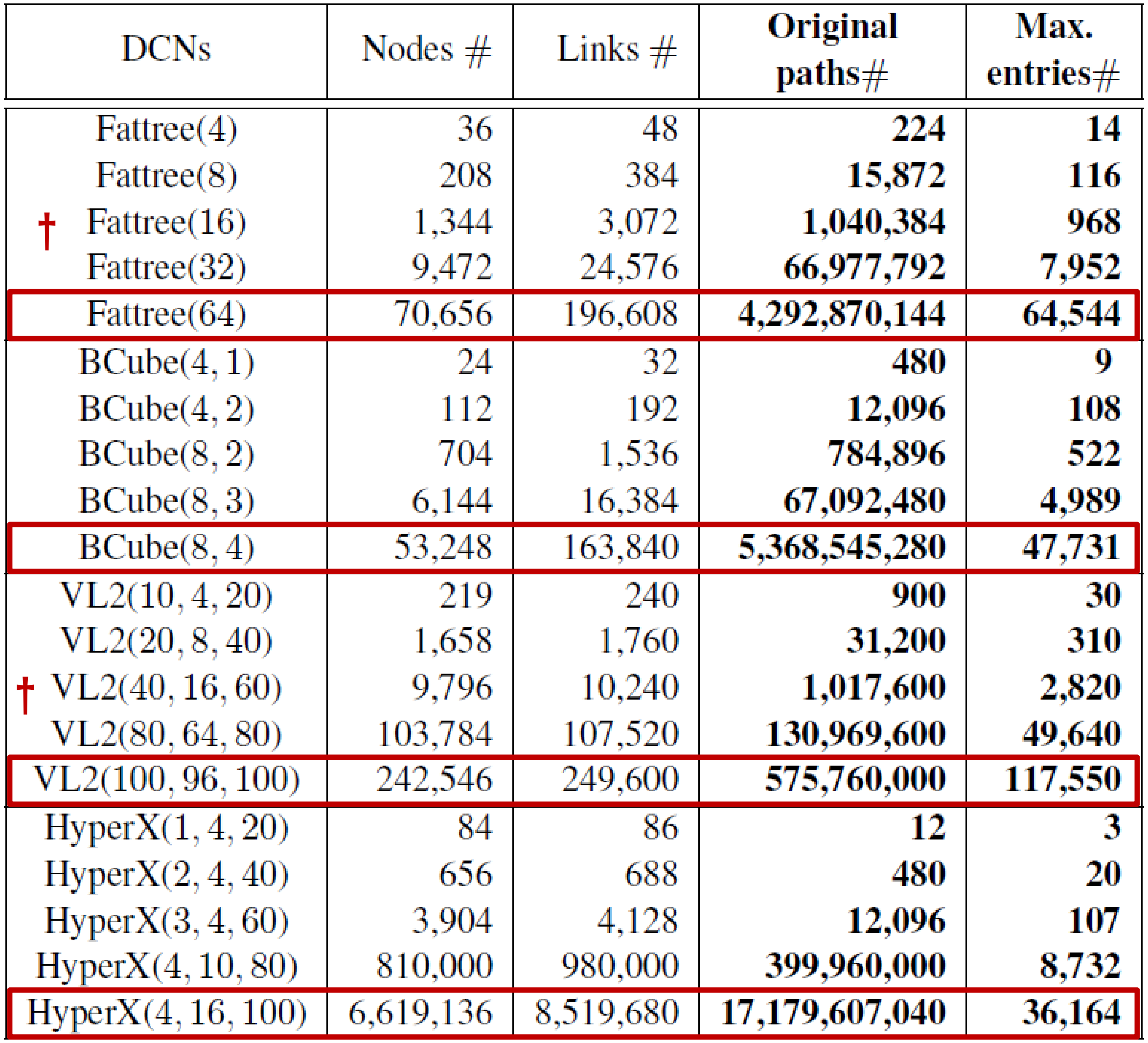


Coordinated ID assignment[†]

1. assign IDs to path sets on each switch separately
   /*optimal, but may cause ID inconsistency, i.e., one path set has multiple IDs */
2. correct inconsistent IDs with each path set incrementally
   /*choose one ID that leads to minimal entries increase for correction*/

[†] Remark: exist custom algorithm for tree-based topologies, e.g., Fattree, VL2, etc.

Scalability Evaluation



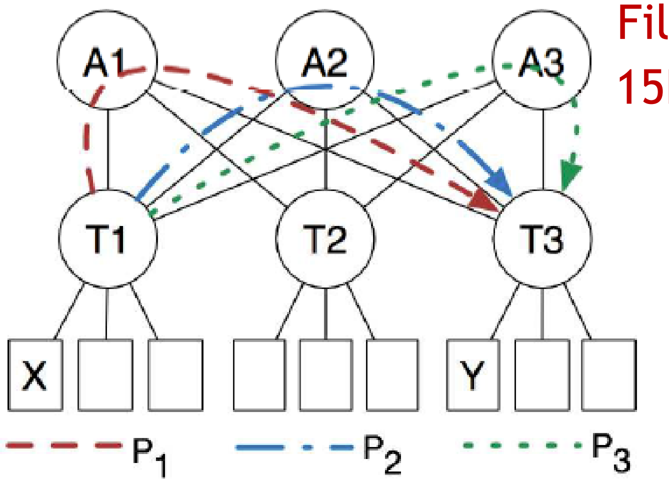
DCNs	Nodes #	Links #	**Original paths#**	**Max. entries#**
Fattree(4)	36	48	**224**	**14**
Fattree(8)	208	384	**15,872**	**116**
† Fattree(16)	1,344	3,072	**1,040,384**	**968**
Fattree(32)	9,472	24,576	**66,977,792**	**7,952**
Fattree(64)	70,656	196,608	**4,292,870,144**	**64,544**
BCube(4, 1)	24	32	**480**	**9**
BCube(4, 2)	112	192	**12,096**	**108**
BCube(8, 2)	704	1,536	**784,896**	**522**
BCube(8, 3)	6,144	16,384	**67,092,480**	**4,989**
BCube(8, 4)	53,248	163,840	**5,368,545,280**	**47,731**
VL2(10, 4, 20)	219	240	**900**	**30**
VL2(20, 8, 40)	1,658	1,760	**31,200**	**310**
† VL2(40, 16, 60)	9,796	10,240	**1,017,600**	**2,820**
VL2(80, 64, 80)	103,784	107,520	**130,969,600**	**49,640**
VL2(100, 96, 100)	242,546	249,600	**575,760,000**	**117,550**
HyperX(1, 4, 20)	84	86	**12**	**3**
HyperX(2, 4, 40)	656	688	**480**	**20**
HyperX(3, 4, 60)	3,904	4,128	**12,096**	**107**
HyperX(4, 10, 80)	810,000	980,000	**399,960,000**	**8,732**
HyperX(4, 16, 100)	6,619,136	8,519,680	**17,179,607,040**	**36,164**

†Remark: can be much smaller if apply tree-based custom algorithm

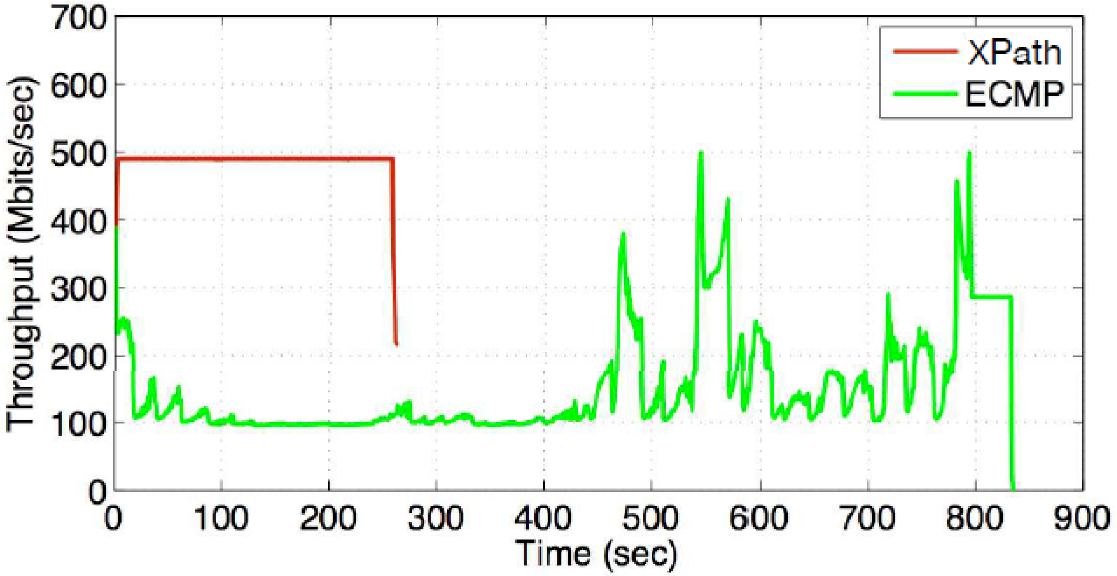
XPath application showcase #1: Provisioned IOPS

File copy: X->Y 15GB = 30 files x 500MB/each

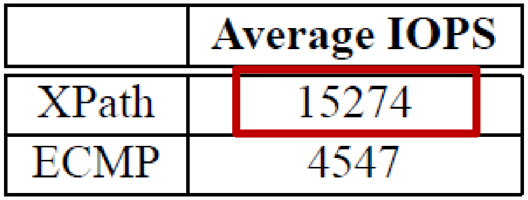
15K (IOPS) x 4KB (chunk size) x 8 ≈ 500Mbps



(a) Remaining bandwidth on P_1, P_2, P_3 is 300, 100, 100 Mbps.



(b) Throughput and completion time of XPath and ECMP.

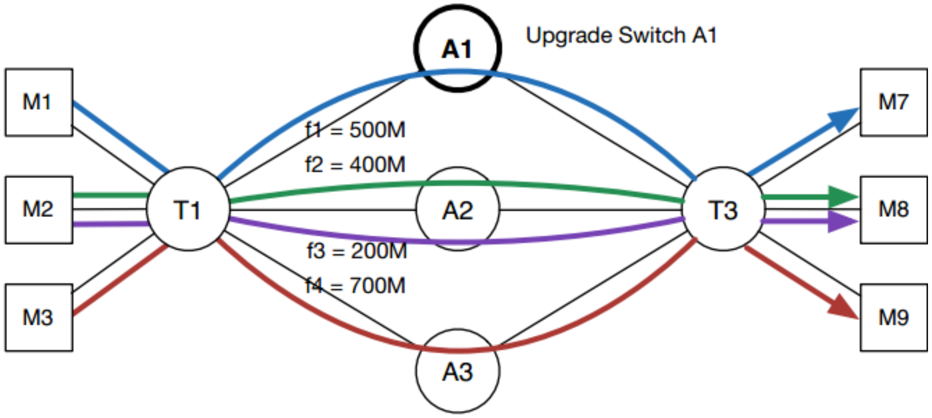


	Average IOPS
XPath	15274
ECMP	4547

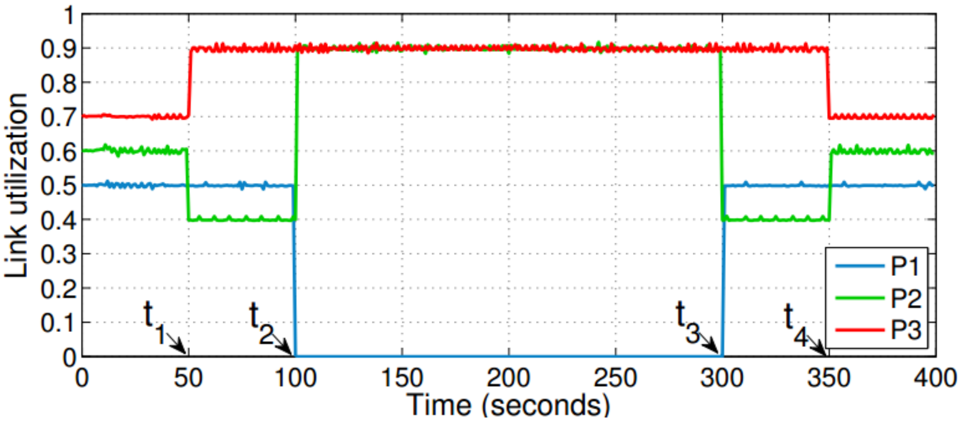
(c) Average IOPS.

We leveraged XPath to provide necessary network bandwidth to achieve the provisioned IOPS.

XPath application showcase #2: Congestion-free update



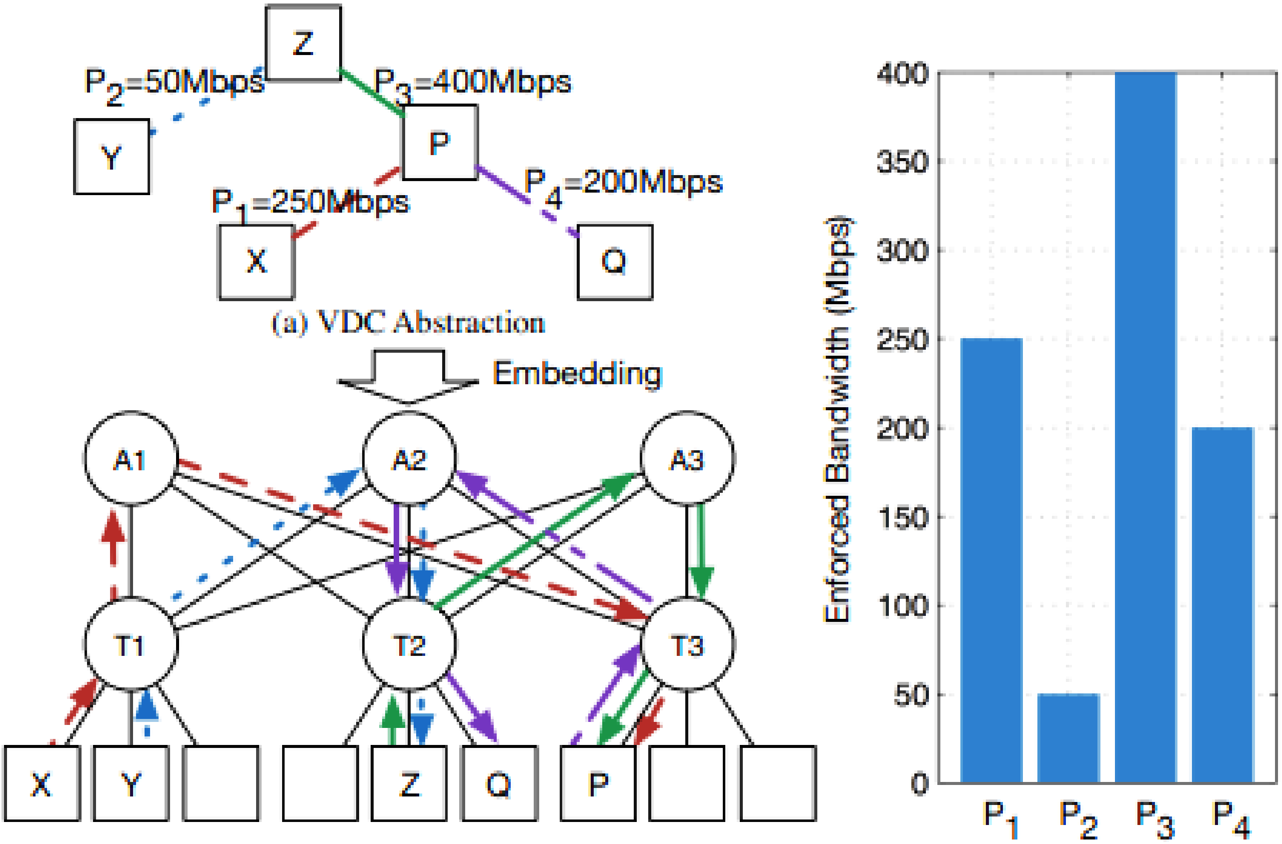
Path P1: T1 -> A1 -> T3,
Path P2: T1 -> A2 -> T3,
Path P3: T1 -> A3 -> T3.



Time t1: move f3 from P2 to P3,
Time t2: move f1 from P1 to P2,
Time t3: move f1 from P2 to P1,
Time t4: move f3 from P3 to P2.

We leveraged XPath to assist network to accomplish congestion-free update (e.g., zUpdate [Sigcomm’13]).

XPath application showcase #3: Virtual network enforcement



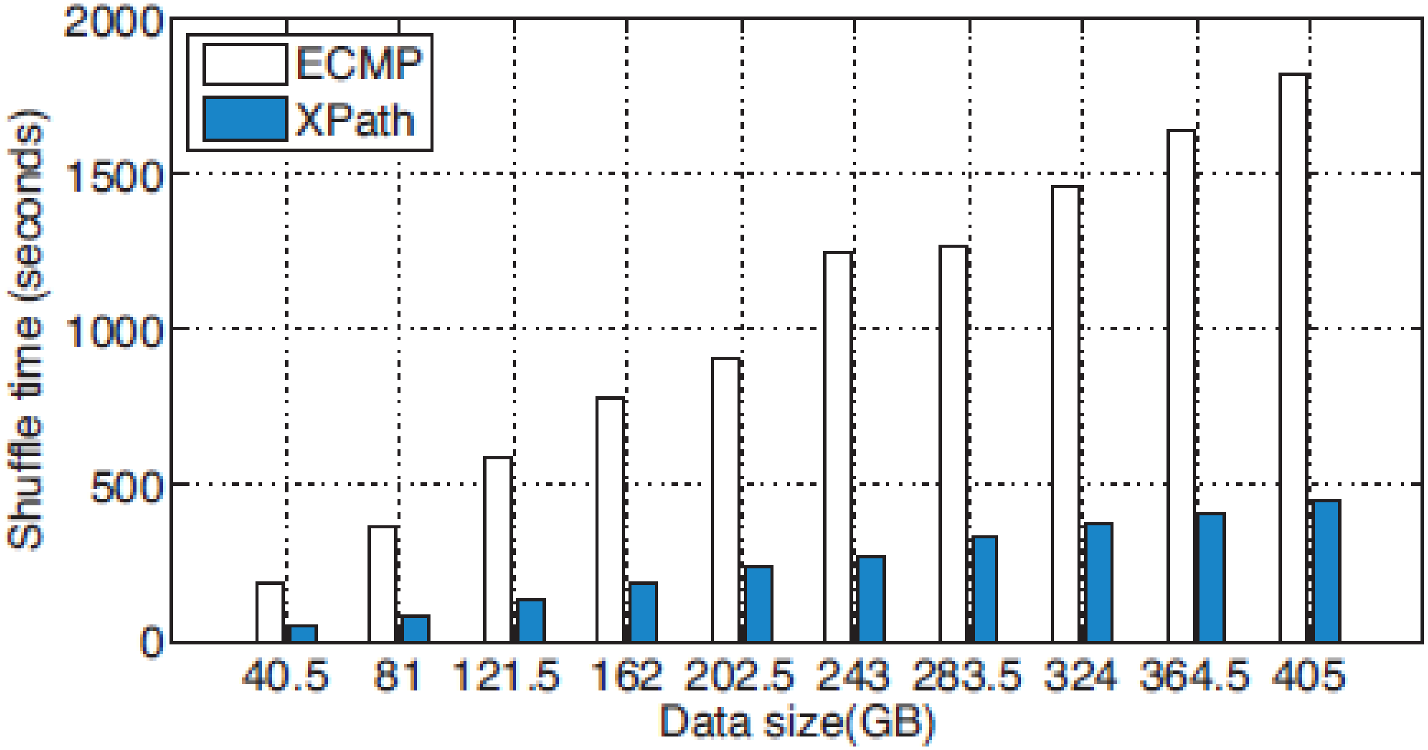
(a) VDC Abstraction

(b) Physical Topology

(c) Path

We leveraged XPath to accurately enforce VDC with bandwidth guarantees (e.g., SecondNet [CoNEXT'10], Oktopus [Sigcomm'11], TIVC [Sigcomm'12], CloudMirror [Sigcomm'14]).

XPath application showcase #4: Map-reduce data shuffle



We leveraged XPath to explicitly arrange parallel paths to speed up many-to-many Map-reduce data shuffle.

Related work

- Topology-aware DCN routings (e.g., PortLand, VL2 [Sigcomm’09])
  - Small routing tables
  - Rely on ECMP and VLB, not support explicit path control
- Source routing (e.g., BCube [Sigcomm’09])
  - Software-based, not supported by most commodity DCN switches
  - Variable header length vs fixed length in XPath
- MPLS
  - Label Distribution Protocol (LDP) for label assignment
  - Exact Matching (EM) vs LPM in XPath
- OpenFlow
  - Dynamic path setup overhead
  - Generic yet limited flow entries vs XPath leverages LPM
  - XPath complements OpenFlow in explicit path control
  - XPath can also leverage OpenFlow protocols for path selection and failure handling

Summary

- Design:
  - A concept of path ID to express an end-to-end path,
  - An idea of pre-installing all desired paths into IP LPM tables,
  - A two-step algorithm that translates the idea into practice.
- Application:
  - Scalable, work on large DCNs,
  - Practical, easy to implement, no modification on commodity switches,
  - Can be integrated into many applications and benefit them,
  - Our other projects heavily rely on XPath
- Try it out @ http://sing.cse.ust.hk/projects/XPath

Thanks, Q&A