Chapter 5
Network Layer: Control Plane

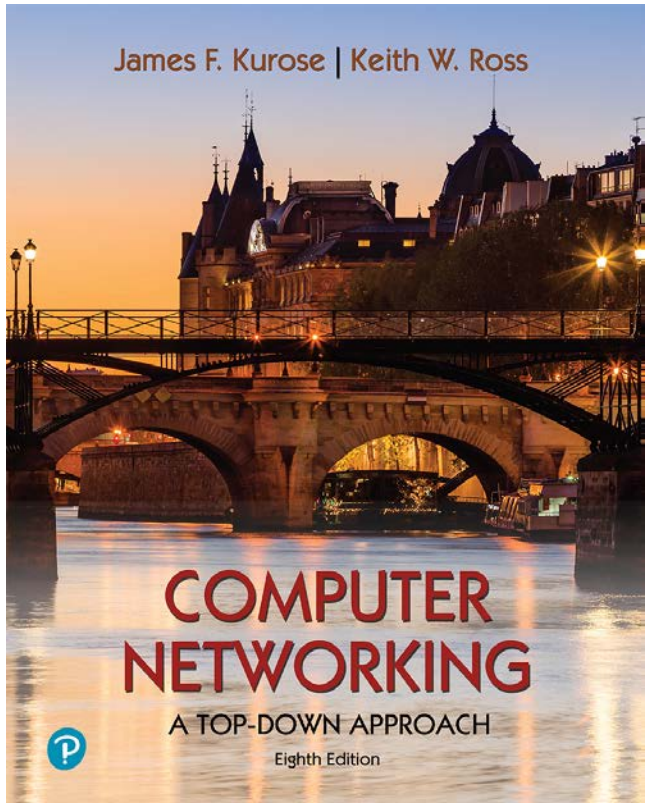
A note on the use of these PowerPoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR



Computer Networking: A Top-Down Approach

8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Network-layer functions

- forwarding: move packets from router's input to appropriate router output

data plane

- routing: determine route taken by packets from source to destination

control plane

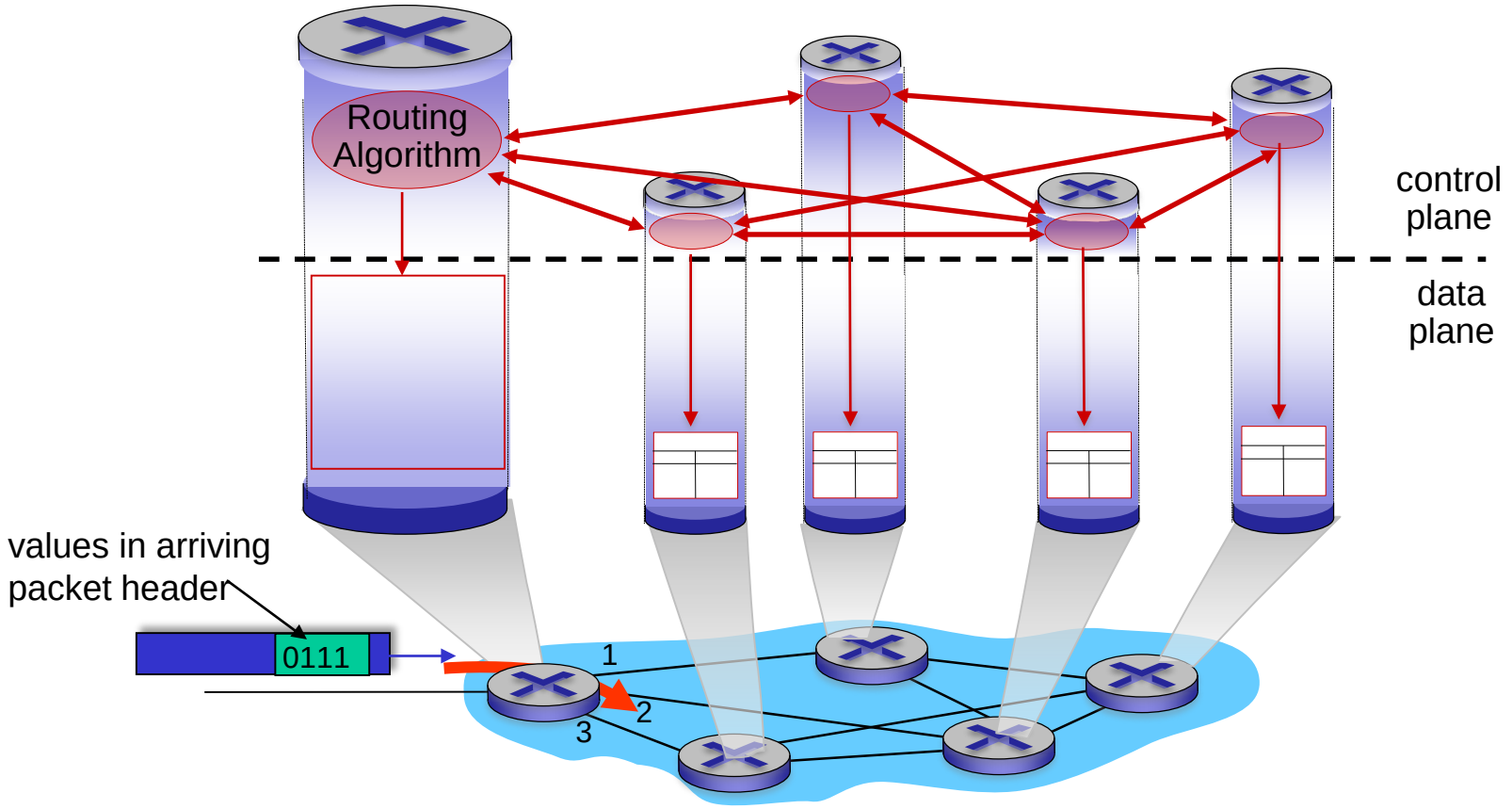
Two approaches to structuring network control plane:

- per-router control (traditional)
- logically centralized control (software defined networking)

-
اطلاعات جدول از روتینگ پروتکل ها به دست میاد
در کنترل پلن روتینگ انجام میشه با استفاده از روتینگ پروتکل ها و این اطلاعات به دست میاد
برای کنترل پلن دو دیدگاه گفتیم که گفتیم
در دیدگاه دوم ینی logically centralized control گفتیم که کنترل پلن رو از روتر جدا میکنیم

Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



-
دیدگاه سنتی این بود که ما توی روترها کنترل پلن هست و به صورت distributed یا توزیع شده با هم متحد می شن و کار کنترل پلن رو انجام میدن
و اصلی ترین کارش هم اجرای الگوریتم های روتینگ است و به دست اوردن جدول هایی که داریم اینجا ینی جدول فوروواردینگ که هر روتر نیاز داره برای بحث فوروواردینگش

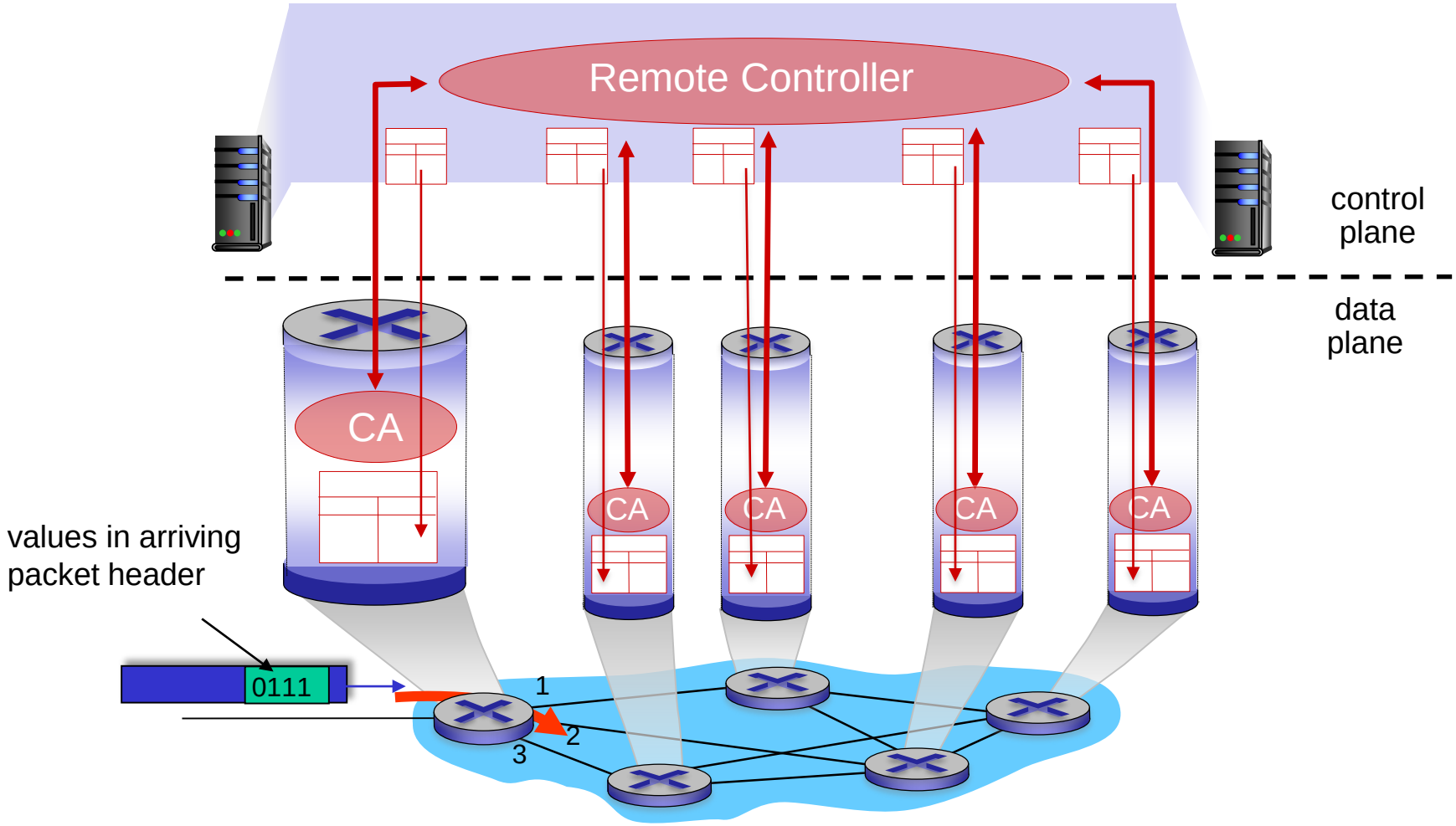
Software defined networking (SDN)

- Internet network layer: historically implemented via distributed, per-router control approach:
  - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
  - different "middleboxes" for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane

-
در دیدگاه سنتی: روتینگ هم کار همین روترهای شبکه است و مسئله روتینگ مسئله لوکال نیست و مسئله سطح شبکه است و باید توپولوژی کل شبکه باشه پس توی این حالت روترها باید با هم همکاری کنن و عملا یک پروتکل distributed خواهد بود که توش روترها مشارکت می کنن
پس توی دیدگاه سنتی روتر ما یک سیستم یکپارچه است که هم فورواردینگ رو انجام میده و هم اون پروتکل هایی که پروتکل های اختصاصی هستند یا پروتکل های عمومیت دارند
و در این دیدگاه برای کارهایی غیر از روتینگ برای مثال firewalls, load balancers, NAT boxes و .. باکس های جداگانه داریم مثلا اگر بخوایم توی شبکه firewall بذاریم باید یک باکس جدا یا در بعضی از روترها جلوی روتر گذاشته میشه

Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



-
تو دیدگاه جدید گفتیم که این کنترل پلن نیاز نیست که distributed باشه و می تونه متمرکز باشه و لازم نیست روی روترها باشه و می تونه روی سرور اجرا بشه
پس ما یک کنترلر داریم که یکپارچه برای کل شبکه است همه کارهای مربوط به مسیرها رو انجام میده و بعد control agents or CA توی هر روتر به این مرتبط است و این جدول های هر روتر رو کنترلر ما از طریق CA به روتر میده که برای فورواردینگ استفاده بشه
به این ترتیب ما کنترل پلن رو جدا کردیم و یکپارچه کردیم و متمرکز کردیم و توی یک سرور قرار دادیم در قالب کنترلر

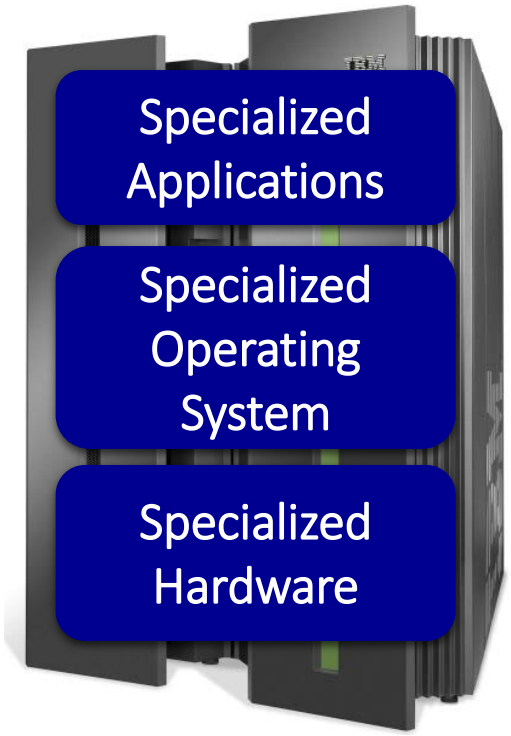
Software defined networking (SDN)

Why a *logically centralized* control plane?

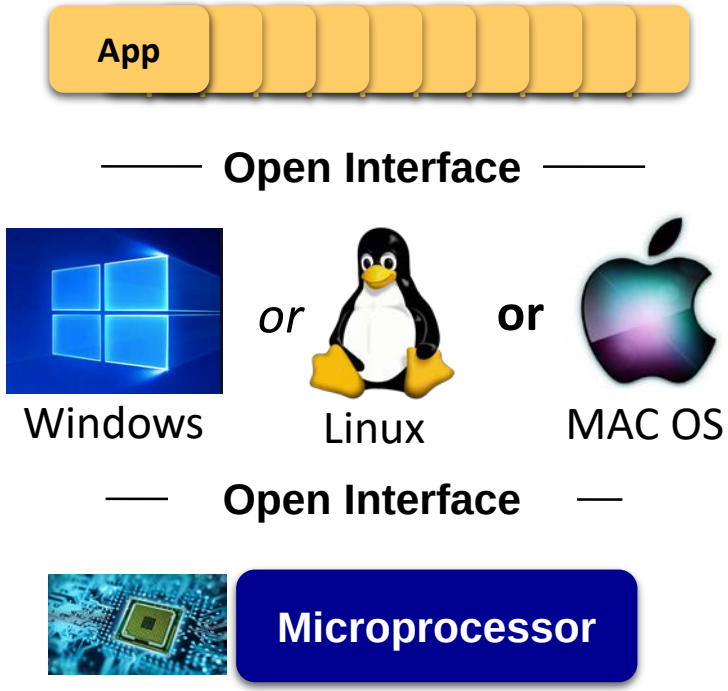
- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) allows "programming" routers
  - centralized "programming" easier: compute tables centrally and distribute
  - distributed "programming" more difficult: compute tables as result of distributed algorithm (protocol) implemented in each-and-every router
- open (non-proprietary) implementation of control plane
  - foster innovation: let 1000 flowers bloom

-
چرا این کارو انجام میدیم؟ چون یکسری مزیت هایی به وجود میاد و مهمترین مزیتش این میشه که مدیریت شبکه خیلی ساده تر میشه
مدیریت شبکه کار سختی، پرهزینه و پیچیده است و یه بخشی از پیچیدگیش هم مال این است که روترهای شبکه در واقع یک سیستم های پیچیده ای هستند و این دیدگاه جدیدی باعث میشه که مدیریت شبکه خیلی ساده تر بشه و اشکالاتی که توی شبکه ها پیش میاد اونجا کمتر است و اگر هم پیش بیاد راحت تر حل میشه و راه حلش هم ساده تر است و به جای اینکه متصل به یکسری ادمین باشه، یکسری الگوریتم و نرم افزار میتونه این کار رو بکنه
فورواردینگ ما انعطاف پذیرتر و راحت تر میشه یعنی کنترلر میاد این جدول را برای هر flow پروگرم میکنه و به این ترتیب تکلیف فورواردینگ پکت های flow مشخص میشه و این در مقایسه در مقابل با پروتکل های distributed که توی شبکه استفاده میشه و براساس اون جدول های فورواراد به دست میاد و برای اون هم یکپارچه برای همه flow ها است و این کار سختی است در کل
مزیت سوم: کنترل پلنمون دیگه محدود به یک باکس خاص نیست یعنی توی روترهای که توی شبکه استفاده می کنیم این کنترل پلن توی روتر با روتر میاد و هرچی که اون داره ما استفاده میکنیم ولی در دیدگاه SDN اینجوری نیست و این خیلی باز میشه و دیگه لازم نیست ما نرم افزار اختصاصی استفاده کنیم بلکه می تونیم هر نرم افزاری استفاده بکنیم

SDN analogy: mainframe to PC revolution



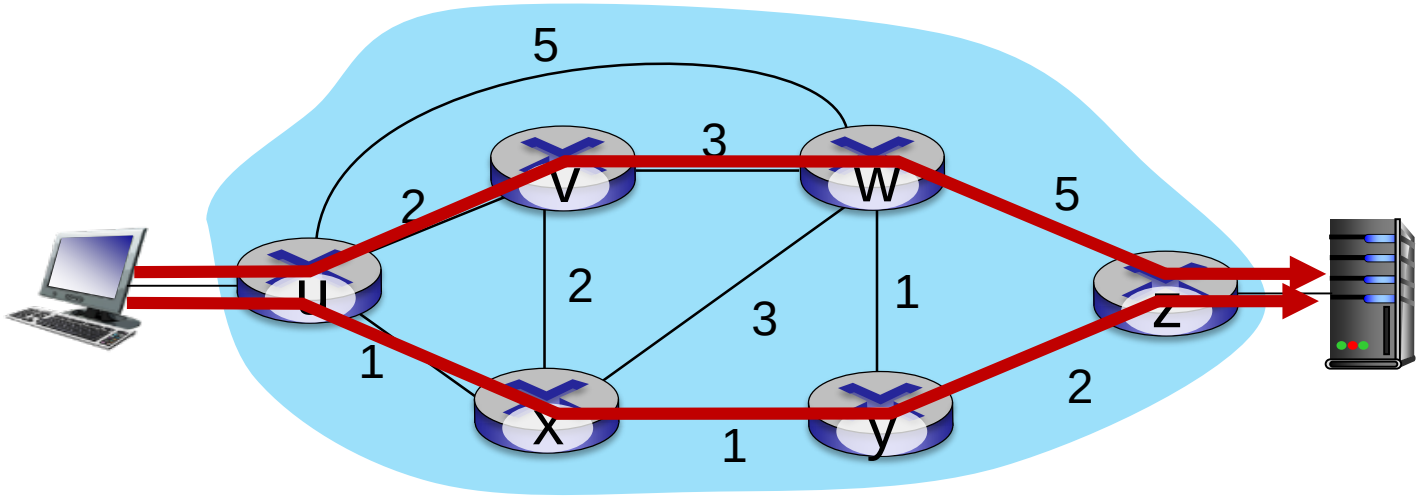
Vertically integrated
Closed, proprietary
Slow innovation
Small industry



Horizontal
Open interfaces
Rapid innovation
Huge industry

* Slide courtesy: N. McKeown

Traffic engineering: difficult with traditional routing



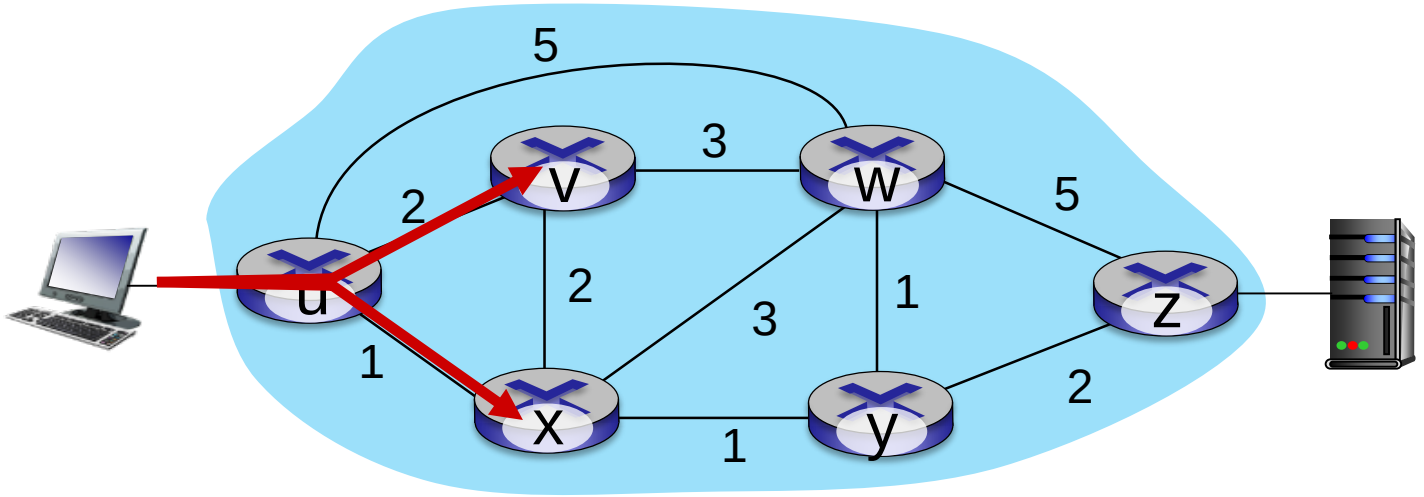
Q: what if network operator wants u-to-z traffic to flow along *uvwz*, rather than *uxyz*?

A: need to re-define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

link weights are only control "knobs": not much control!

-
مثال:
ما یک الگوریتمی داریم برای Traffic engineering و براساس اون به این نتیجه می رسیم که ترافیک u to z ما خوبه که از مسیر uvwz عبور پیدا کنه و ترافیک x to z از طریق xwyz براساس هزینه های که این لینک های شبکه داره پروتکل های که یاد گرفتیم و مثلا اگر دایکسترا رو اجرا بکنیم برای x to z مسیر xyz انتخاب میشه ولی به دلیل Traffic engineering ما می خوایم ترافیک x to z می خوایم از مسیر xwyz عبور پیدا بکنه
چجوری میتونیم توی همچین شبکه ای باعث بشیم ترافیک x to z به جای اینکه مسیر xyz بره ازمسیر xwyz بره (شاید مثلا بخاطر congestion توی پورت y بخوایم این کارو بکنیم) و ترافیک u to z به جای اینکه uxyz بره بیاد از مسیر uvwz بره ؟
تنها راهی که اینجا هست اینه که وزن ها رو تغییر بدیم ولی این باعث میشه همه ترافیکی که از x رد میشه و مقصدش z است توی این مسیر جدید بره و اگر مثلا توی پورت x ما congestion داشتیم منتقل بشه به اون یکی پورت ینی خیلی به بحث Traffic engineering کمک نمیکنه و زمانی که کمک میکنه که ما بتونیم perfllow این تغییرات رو انجام بدیم که این توی شکل سنتی امکان پذیر نیست

Traffic engineering: difficult with traditional routing

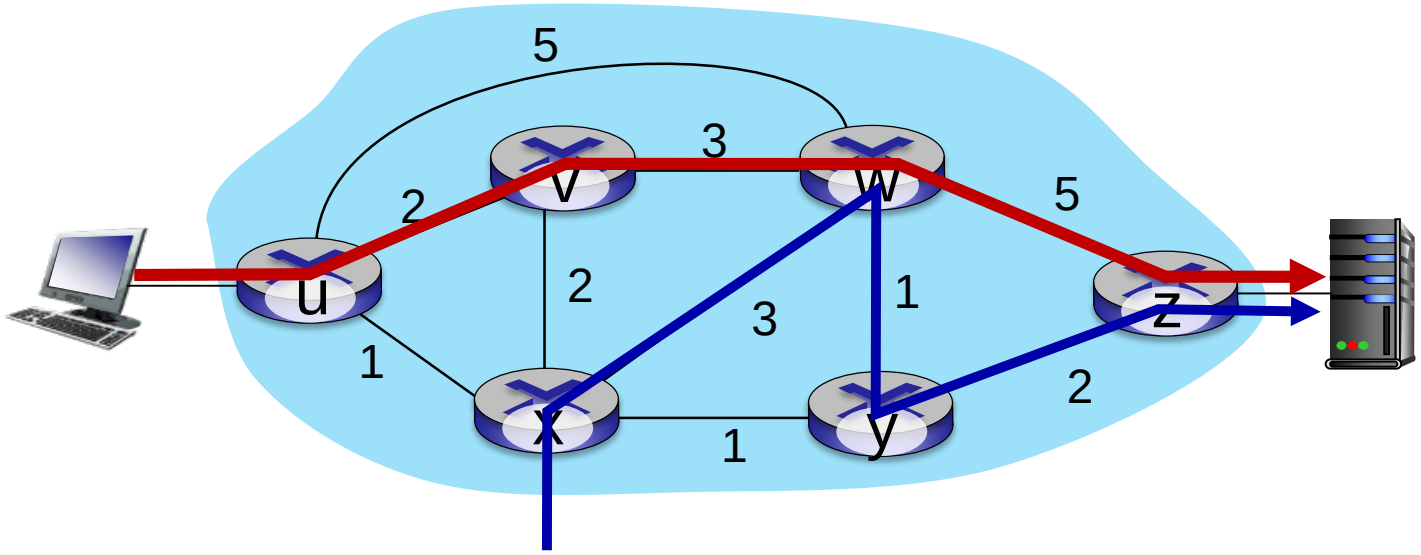


Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?

A: can't do it (or need a new routing algorithm)

-
مثال:
ما ترافیک u to z را میخوایم تقسیم بکنیم که یه بخشیش رو از طریق uvwz منتقل بکنیم و یه بخشیش رو از طریق uxyz ینی load balancing میخوایم انجام بدیم که این کار کمک میکنه به پایین اومدن لود نودهای شبکه و از بین رفتن کانجنسشن
توی شکل سنتی مشخصه ما نمی تونیم این کارو بکنیم چون وقتی بسته میاد به u و مقصدش z میگه مثلا برای z این مسیر خوبه و کل کارها دیگه از اون مسیر می ره جلو و راهی نداریم برای اینکه اینو جدا بکنیم

Traffic engineering: difficult with traditional routing



Q: what if w wants to route blue and red traffic differently from w to z?

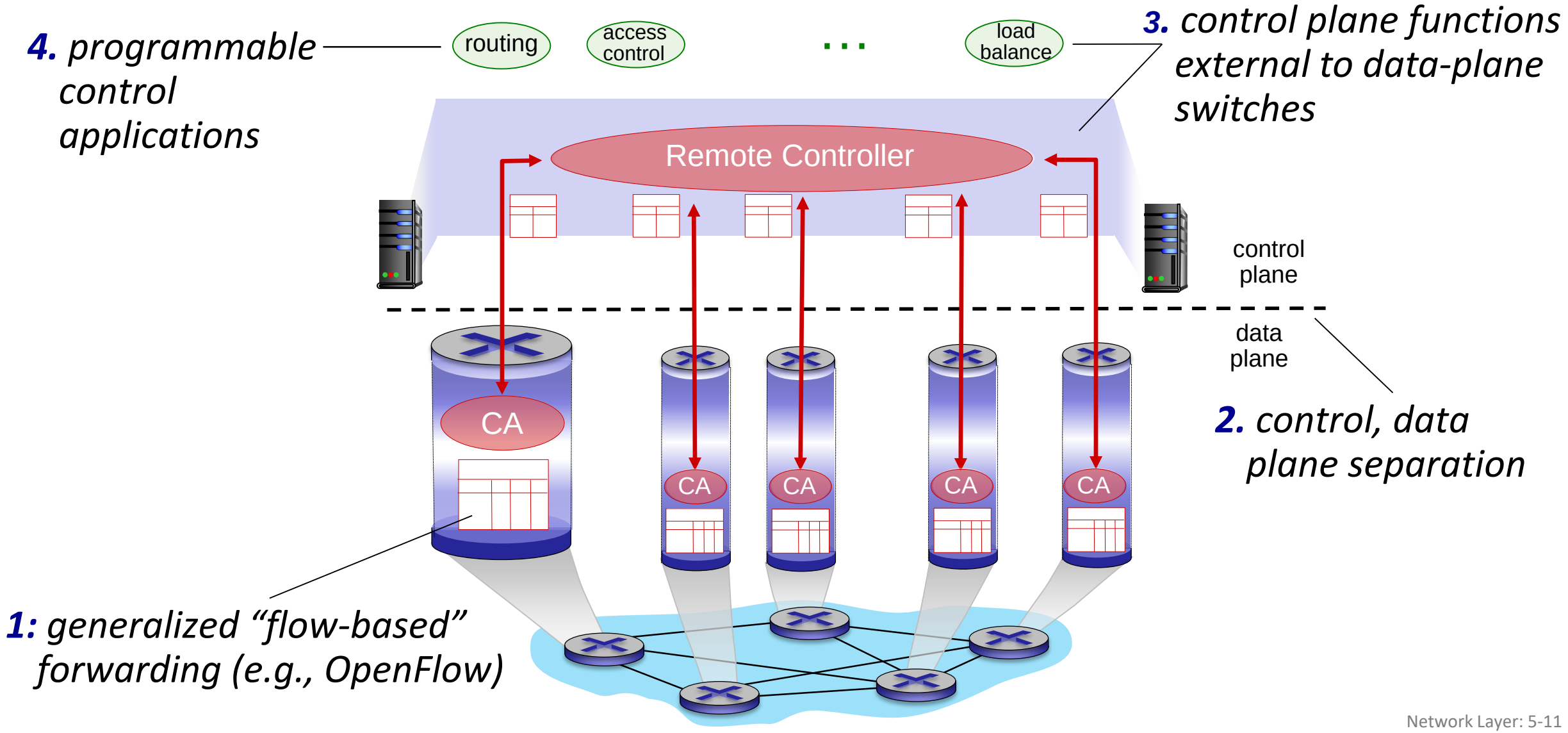
A: can't do it (with destination-based forwarding, and LS, DV routing)

We learned in Chapter 4 that generalized forwarding and SDN can be used to achieve *any* routing desired

-
مثال بعدی:
دوتا ترافیک داریم به صورت شکل روبه رو و توی مسیرهایی که نشون داده شده عبور پیدا میکنند برای ترافیک ابی رنگ ما می خوایم وقتی به w رسید ادامه مسیرش از طریق y باشه ولی قرمز رنگ مستقیم بره : ینی دوتا ترافیک از مسیرهای مختلف داریم ولی یه تیکش که مشترکه میخوایم جدا باشه و توی روش سنتی باز امکان پذیر نیست ینی ترافیک ابی و قرمز وقتی به روتر w می رسن بسته های ابی و قرمز وقتی مقصد z دوتاشون از یک مسیر می رن در حالی که ما می خوایم اینارو تفکیک بکنیم

یه مثال دیگه توی فیلمش زده که توی اسلایدها نیست فیلم 4 از دقیقه 28 تا 35 ببین
توی match یه سری فیلدهایی داریم که مشخص میکنیم چه فیلدی چه مقداری داشته باشه مچ بشه
10.3.*.* ینی همه بسته هایی که دارند از 10.3 میان و این پکت ها برن به مقصد های 10.2
و همه این ترافیک فوروارد میشه به پورت سه که اون پورتی است که می ره به S1
توی S1 میگه : اگر ترافیک هایی از 10.3 بیان و از پورت 1 و مقصدشون 10.2 است فوروارد بشه به پورت 4

Software defined networking (SDN)

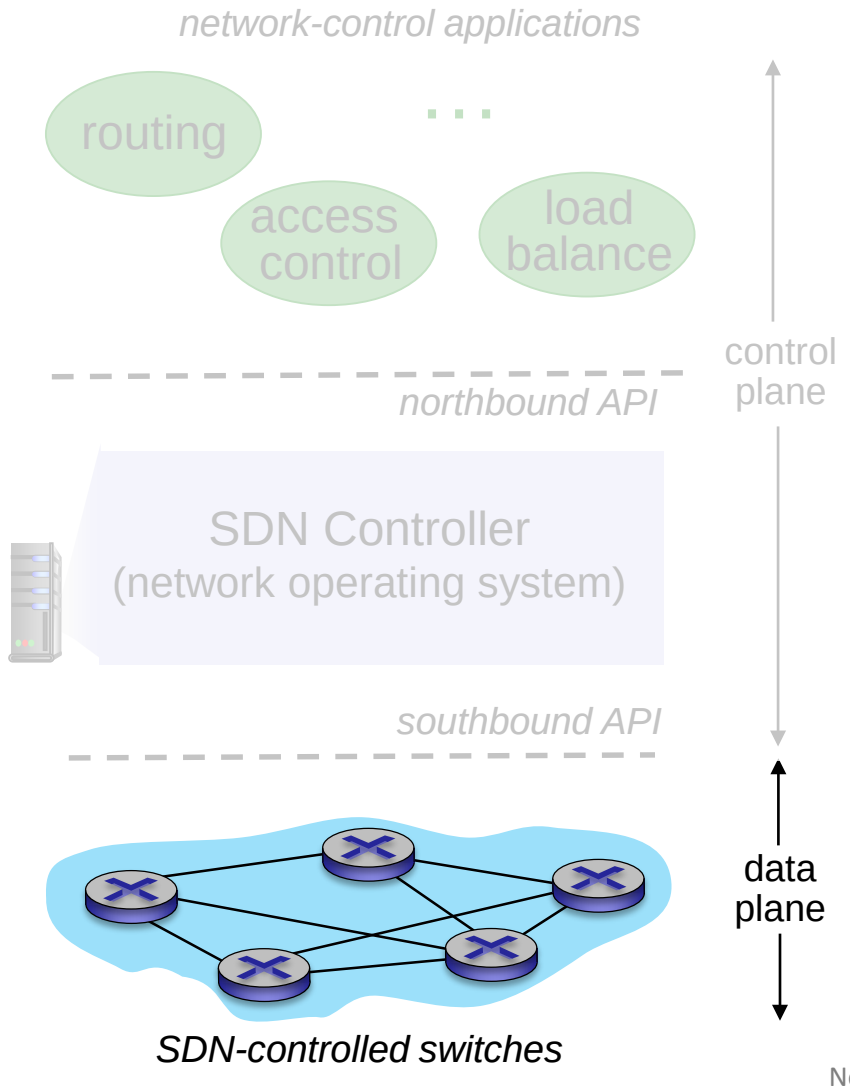


-
معماری جدیدی که داریم میشه SDN و توی این معماری سوییچ ها در سطح شبکه کار فورواردینگ رو انجام میدن که این در واقع میشه اصل کار شبکه
براساس یکسری جدول که گفتیم اینجا generalized forwarding انجام میدیم ینی جدولمون دستینیشن بیس نیست بلکه روی ویژگی های مختلف پکت ها که گفتیم براساس flow این رو دسته بندی می کنیم
اپلیکیشن ها روی کنترلر قرار میگیرند
توی بخش سوم کنترل پلن به صورت متمرکز روی سرور قرار داده میشه و بعد اپلیکیشن های دلخواه رو می تونیم روی کنترل پلنمون نصب و ران بکنیم و به این ترتیب وضع موجود شبکه راحت میشه و شبکه قابلیت های بیشتری رو میتونه پیدا بکنه

Software defined networking (SDN)

Data-plane switches:

- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating with controller (e.g., OpenFlow)

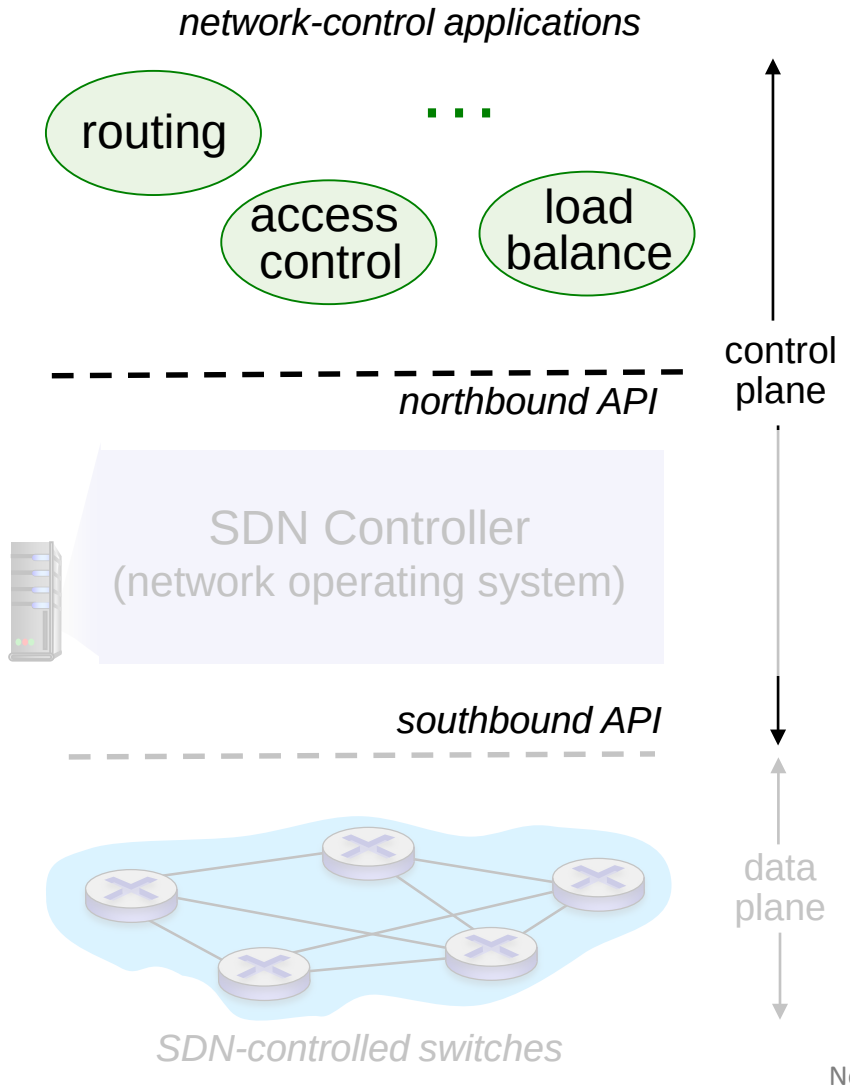


-
خلاصه:
دیتاپلن و کنترل پلن رو جدا کردیم
و دیتا پلن مستقلا توسعه داده میشه و وظایفش خیلی محدود شده و عمدتا فورواردینگ با سخت افزار میتونه انجام بگیره پس سخت افزارهای خاص منظوره برای اون فورواردینگ طراحی شده با سرعت های بالا
پس دیتاپلن سوییچ هایی هستند با ویژگی هایی که گفتیم و صرفا یک جدولی دارند که باید پروگرم بشه توسط کنترلر (با یک اینترفسی این سوییچ مرتبط میشه کنترلر که مبتنی بر open flow است و براساس اون هم این جدول پروگرم میشه و پروتکل ارتباطی که چجوری این کنترلر با سوییچ صحبت بکنه که خود این هم توی پروتکل open flow تعریف شده)

Software defined networking (SDN)

network-control apps:

- "brains" of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller



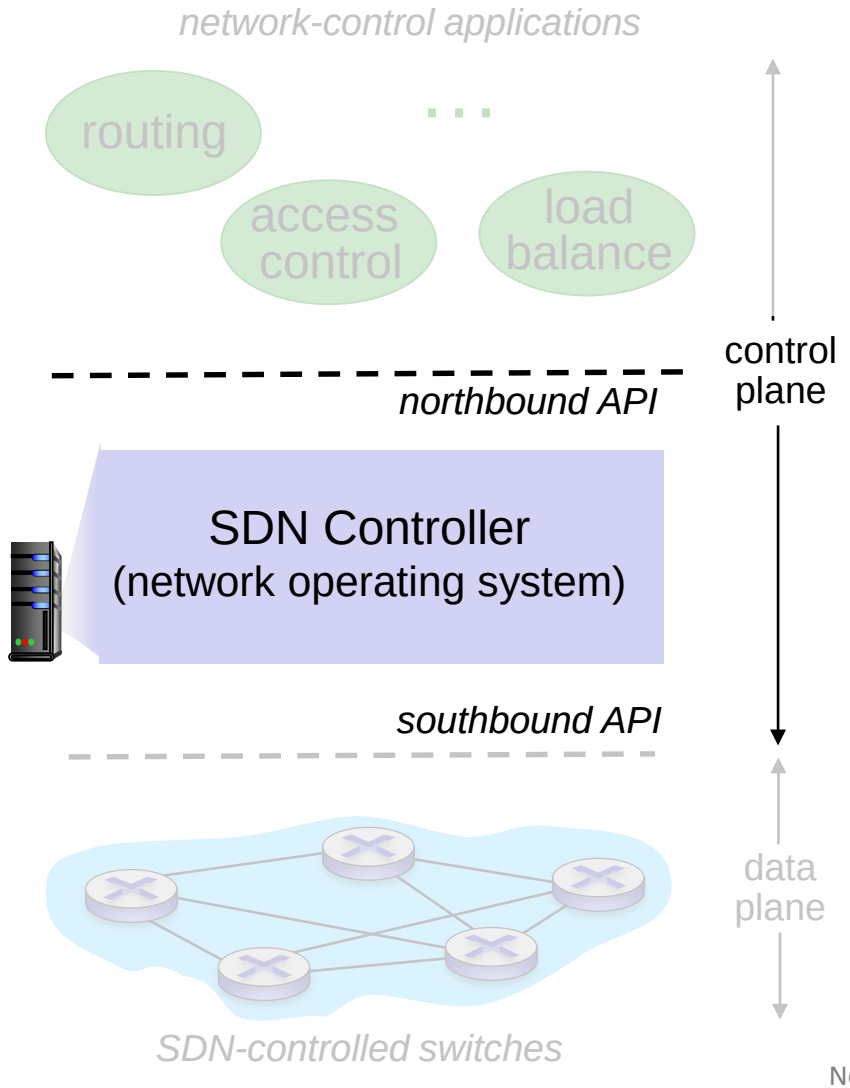
-
اپلیکیشن هایی که در کنترل پلن روی کنترلر قرار می گیرند ونصب میشن و عملا این ها اون مغز شبکه میشه ینی اون کاری که انجام می گیره دقیقا مثل کامپیوتر که اپلیکیشن کار نهایی رو انجام میده توی شبکه هم اپلیکیشن کار نهایی رو انجام میده و این اپلیکیشن میتونه یکسری کارهای سنتی شبکه مثل روتینگ باشه یا می تونه کارهای جدید مثل load balance باشه یا..

نکته: کاری که SDN امکان پذیر کرده اینه که این اپلیکیشن ها unbundled نیستن با سخت افزارمون ینی می تونن ازادانه روی این نصب بشن

Software defined networking (SDN)

SDN controller (network OS):

- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



-
کنترلر که نقش os شبکه رو داره انجام میده انگار که کل شبکه یک سخت افزاری هست و os هست روش و کنترلر عملا این هست
مشابه نقشی که os توی کامپیوتر داره در واقع مدیریت میکنه اگر دیتاپلن مثلا سخت افزار ما باشه و مدیریت میکنه سخت افزار و رویش لایه های نرم افزاری قرار داره و خودش نرم افزار است و این ارتباط رو فراهم میکنه
کنترلر یکسری api برای ارتباط با سوییچ ها که مبتنی open flow است و یکسری api برای ارتباط با اپلیکیشن هایی که روی کنترل پلن قرار میگیرن
کنترل پلن = مجموع کنترلر + اپلیکیشن هایی که روی کنترلر نصب میشن

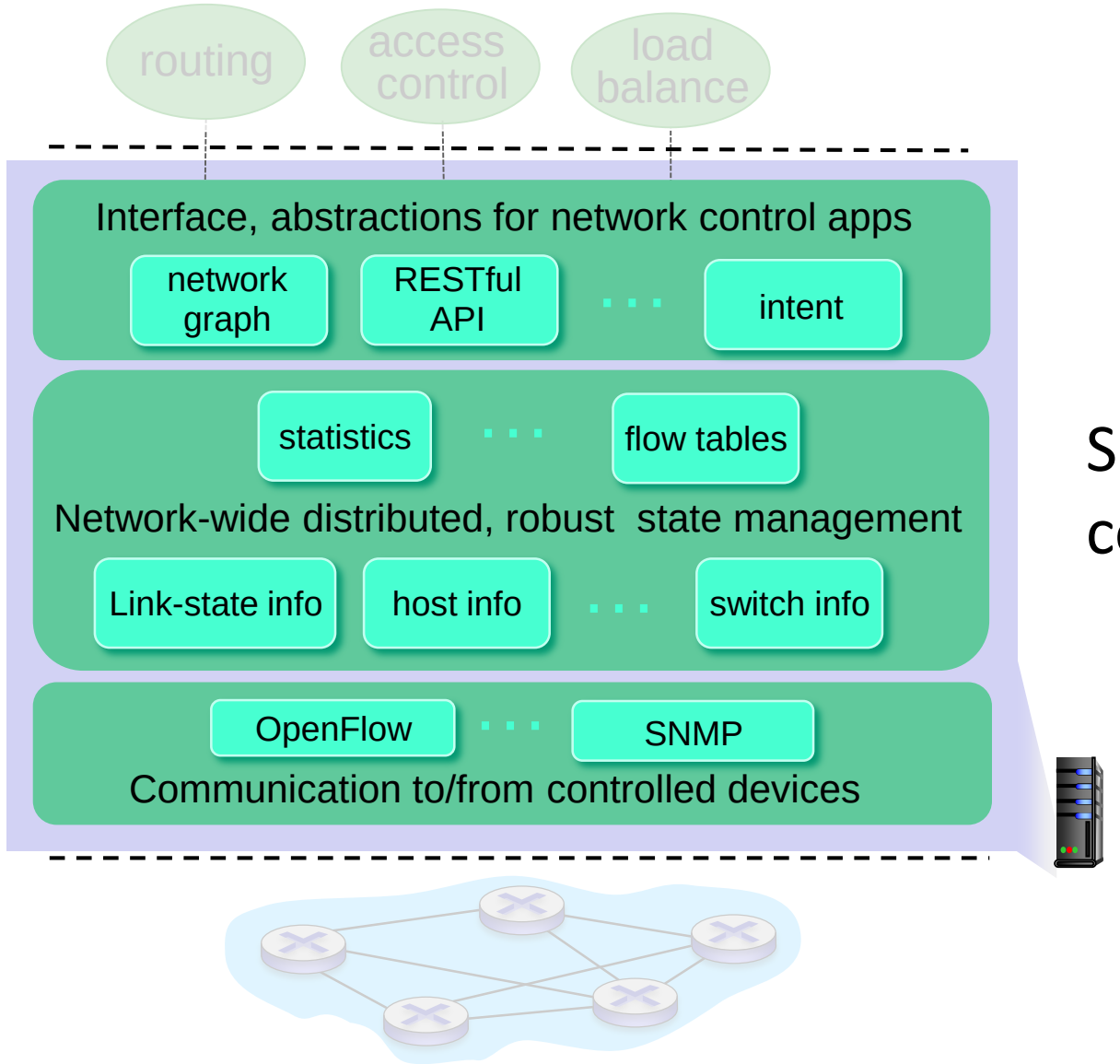
نکته: این میتونه چند تا سرور باشه اینطوری نیست که الزاما یک سرور باشه

Components of SDN controller

interface layer to network control apps: abstractions API

network-wide state management : state of networks links, switches, services: a *distributed database*

communication: communicate between SDN controller and controlled switches



-
خود کنترلر SDN از سه لایه تشکیل میشه:
1- یک لایه که ارتباط با سوییچ ها رو انجام میده
2- یک لایه که app ها روش قرار میگیرن و ارتباط با app ها رو که در واقع api استانداردهایی هستند که توسعه داده شده برای این کار
3- و توی بدنه کنترلر هم یکسری امکاناتی داریم که این نقش رو کنترلر بتونه به عنوان سیستم عامل شبکه انجام بده از جمله اطلاعات خود شبکه (توپولوژی، نودها و لینک ها و...) ینی اینا به صورت یک دیتابیس باید اینجا نگهداری بشوند - اطلاعات سوییچ ها (ادرس سوییچ ها، ارتباطاتش و نحوه ارتباط اون با سوییچ که این ارتباط inbound است ینی از طریق همین شبکه فقط میتونه انجام بشه و..) - اطلاعات اماری و همینطور کپی flow table های هر سوییچ اینجا است (و کنترلر اینو نگه میداره که توی هر سوییچی چه flow هایی به چه صورت هایی پروگرم شده است)

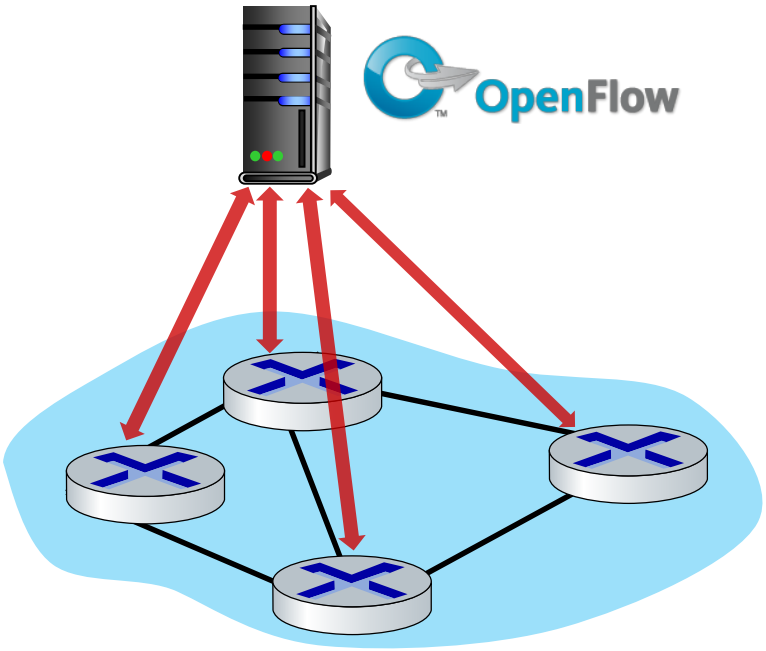
پس SDN یک معماری state full است ینی State ها باید اینجا کاملا نگهداری بشوند و مدیریت بشوند

توی بعضی از کنترلرها بعضی از app ها توی همین جا قرار میگیرند و توی بعضی دیگر هم app ها اینجا نیستند و مثل شکل بالا قرار میگیرند

OpenFlow protocol

- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc.)
- distinct from OpenFlow API
  - API used to specify generalized forwarding actions

OpenFlow Controller



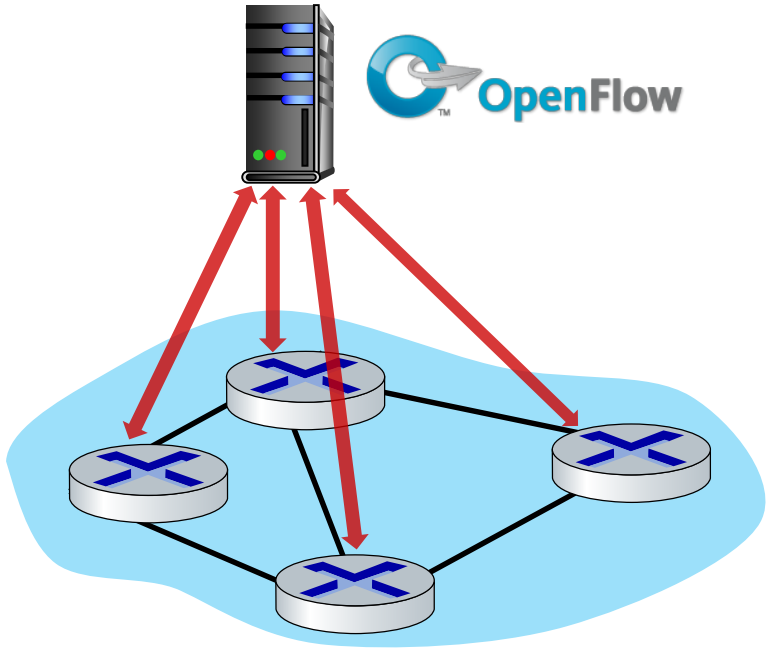
-
OpenFlow protocol هم پروتکلی است که فرمت این جدول ها و قابلیت های اون جنرالیز فورواردینگ رو تعریف میکنه و هم پروتکل ارتباطی کنترلر با سوییچ ها است
به عنوان پروتکل ارتباطی OpenFlow تعریف میکنه که چه نوع مسیج هایی و به چه صورت بین کنترلر و سوییچ ها می تونه رد و بدل بشه
سه نوع مسیج برای پروتکل OpenFlow مطرح شده:
1- مسیج هایی که کنترلر به سوییچ می فرسته
2- مسیج هایی که سوییچ به کنترلر می فرسته
3- مسیج هایی که بین این ها رد و بدل میشه در مواقع مورد نیاز

OpenFlow: controller-to-switch messages

Key controller-to-switch messages

- *features:* controller queries switch features, switch replies
- *configure:* controller queries/sets switch configuration parameters
- *modify-state:* add, delete, modify flow entries in the OpenFlow tables
- *packet-out:* controller can send this packet out of specific switch port

OpenFlow Controller



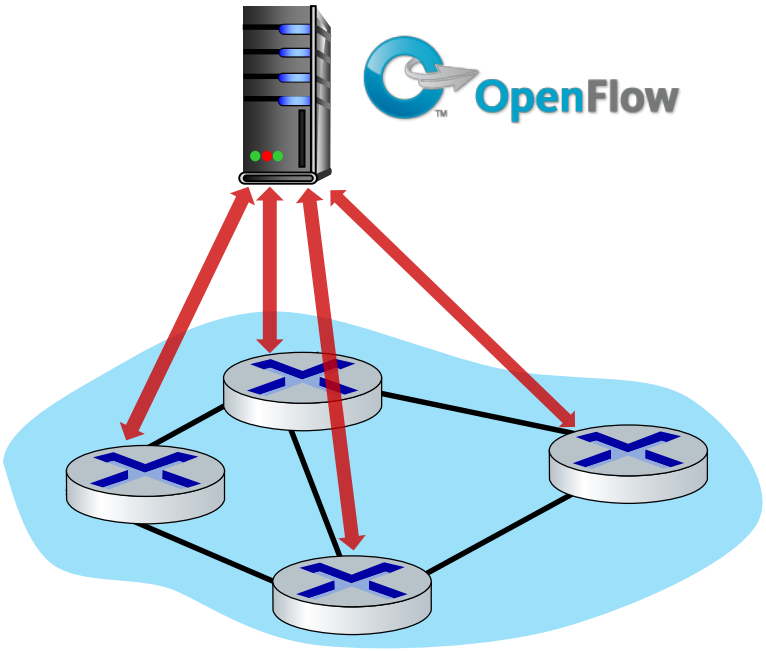
-
مسیج هایی که از کنترلر به سوییچ ارسال می شوند نمونه هاش موارد زیر هستند:
features : کنترلر مسیج هایی از این نوع رو می فرسته به یک سوییچ و از اون ها می خواد اطلاعاتشون رو بفرستن پس قابلیت های سوییچ هارو به این ترتیب جمع اوری و شناسایی میکنه
configure : این سوییچ رو کانفیگ میکنه
modify-state : این state های Flow های توی flow tabel رو modify میکنه
packet-out : یک مسیجی است که توش یک پکت توسط کنترلر encapsole میشه و ارسال میشه به یک سوییچ و به سوییچ گفته میشه که این پکت رو بفرست به پورت مثلا شماره 2 بفرست به این ترتیب کنترلر می تونه پکت رو بفرسته تو سوییچ که این بفرسته تو شبکه

OpenFlow: switch-to-controller messages

Key switch-to-controller messages

- *packet-in:* transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed:* flow table entry deleted at switch
- *port status:* inform controller of a change on a port.

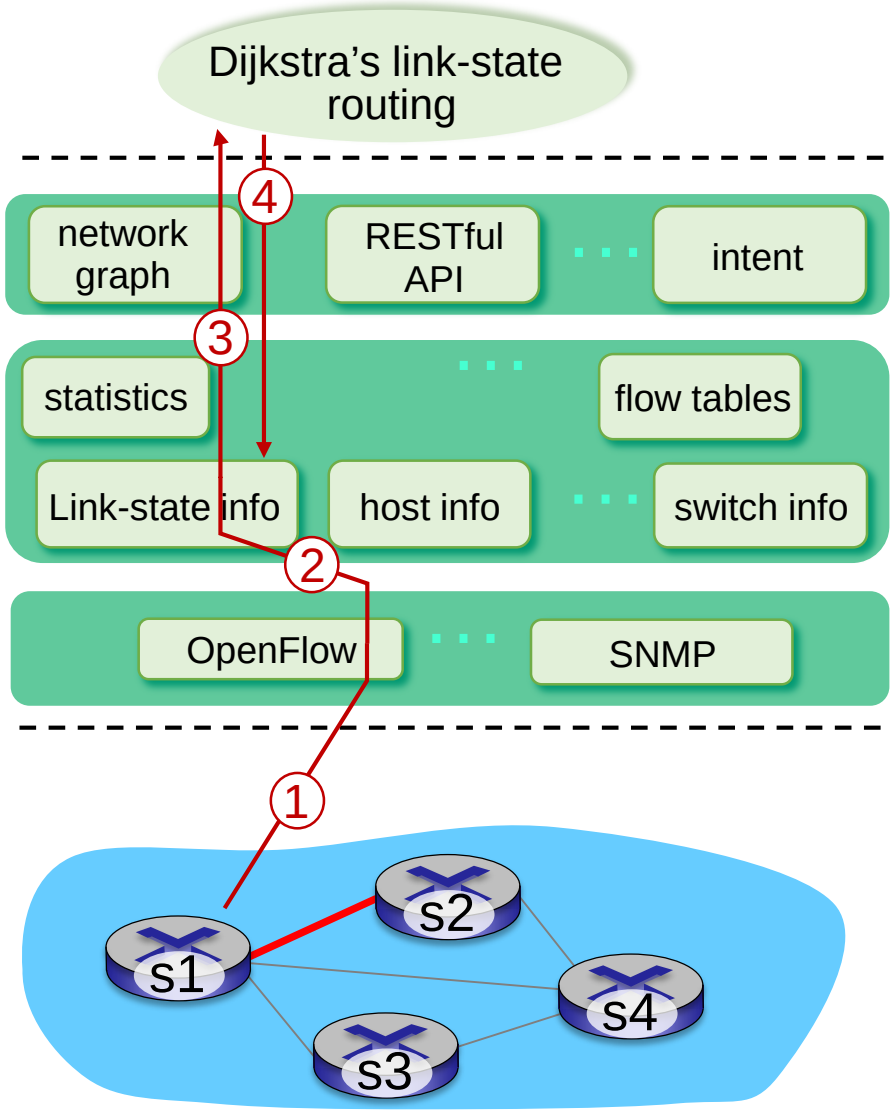
OpenFlow Controller



Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

-
سوییچ به کنترلر:
packet-in: یک پکت جدید میاد به سوییچ و نمیدونه چی کارش کنه پس encapsole میکنه در قالب یک مسیج و می فرسته به کنترلر
flow-removed: اگر یک جدول entry از flow tabel سوییچ حذف بشه این به کنترلر اینو اطلاع میده: اولا این امکان پذیر است که یک سوییچ بیاد یک entry رو خودش حذف کنه و ثانیا باید به کنترلر بگه چون توی کنترلر یک کپی از این entry هست و کنترلر باید State این هارو نگه داری بکنه و اپدیت بکنه و بدونه چه entry هایی توی جدول هر سوییچ است
port status: اگر تغییراتی توی پورت های سوییچ اتفاق بیوفته با این نوع مسیج این تغییرات رو به کنترلر اطلاع میده

SDN: control/data plane interaction example



① S1, experiencing link failure uses OpenFlow port status message to notify controller

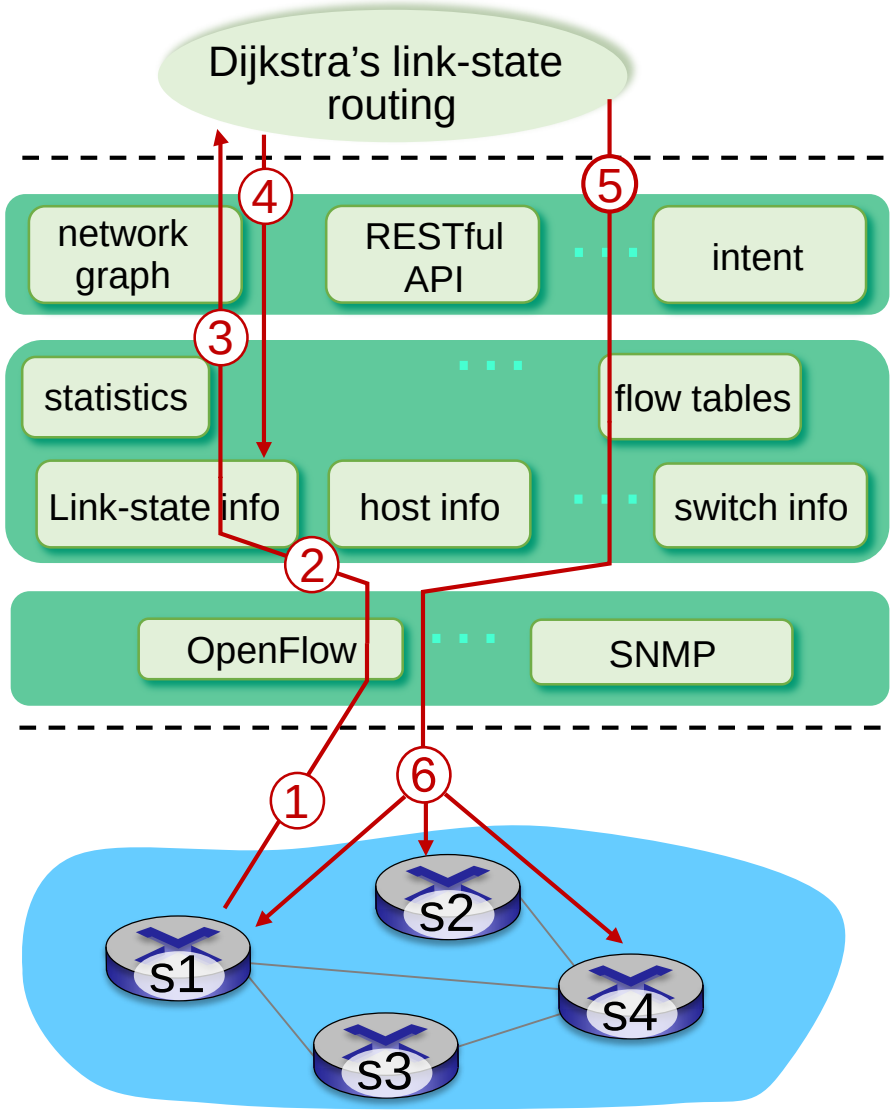
② SDN controller receives OpenFlow message, updates link status info

③ Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.

④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

-
مثال: چجوری از این مسیج ها استفاده میشه؟
توی این مثال فرض می کنیم شبکه ما به صورت شکل روبرو است و یک کنترلر داره مدیریت میکنه همه سوییچ های شبکه رو ولی برای روتینگ فرض میکنیم که همون الگوریتم دایکسترا داره استفاده میشه و این الگوریتم به صورت متمرکز است
فرض شده یک اتفاقی توی S1 می افته و یک پورتش fail میکنه و اینو تشخیص میده S1 این تغییر status پورت هاش رو باید به کنترلر اطلاع بده پس از اون مسیجی که گفتیم که برای همچین منظوری است استفده میکنه و Status خودش رو اطلاع میده از طریق OpenFlow به کنترلر و کنترلر حالا link status info رو براساس اطلاعاتی که از سوییچ گرفته اپدیت میکنه
چون توپولوژی عوض شده روتینگمون باید اطلاح بشه پس توی این مرحله از الگوریتم دایکسترا استفاده میشه پس دایکسترا رو کنترلر ران میکنه
دایکسترا میاد اطلاعات جدید توپولوژی رو از لینک استیت میگیره و مسیرهارو حساب میکنه و توی مرحله 5 میاد مسیرهای جدید رو دیتابیس flow tabel هایی که داره وارد میکنه و بعد توی اخرین مرحله از طریق OpenFlow ینی مرحله 6 این هارو توی سوییچ ها اپدیت میکنه
این اپدیت برای همه سوییچ ها انجام میشه

SDN: control/data plane interaction example

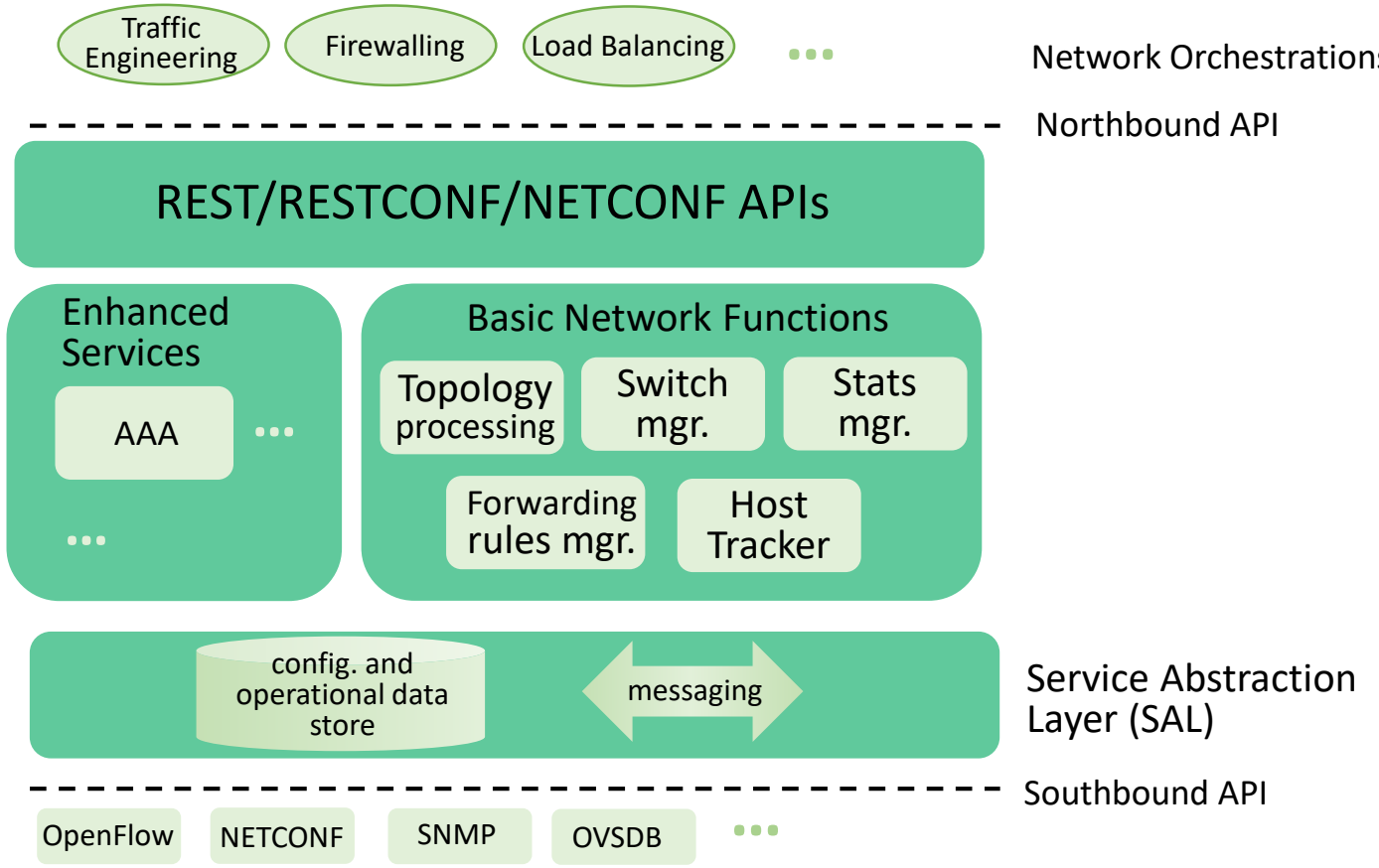


⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed

⑥ controller uses OpenFlow to install new tables in switches that need updating

5- برنامه مسیریابی وضعیت پیوند با مولفه flow-table-computation در کنترل کننده SDN تعامل دارد، که جداول جریان جدید مورد نیاز را محاسبه می کند.
6- کنترلر از OpenFlow برای نصب جداول جدید در سوئیچ هایی که نیاز به آپدیت دارند استفاده می کند

OpenDaylight (ODL) controller

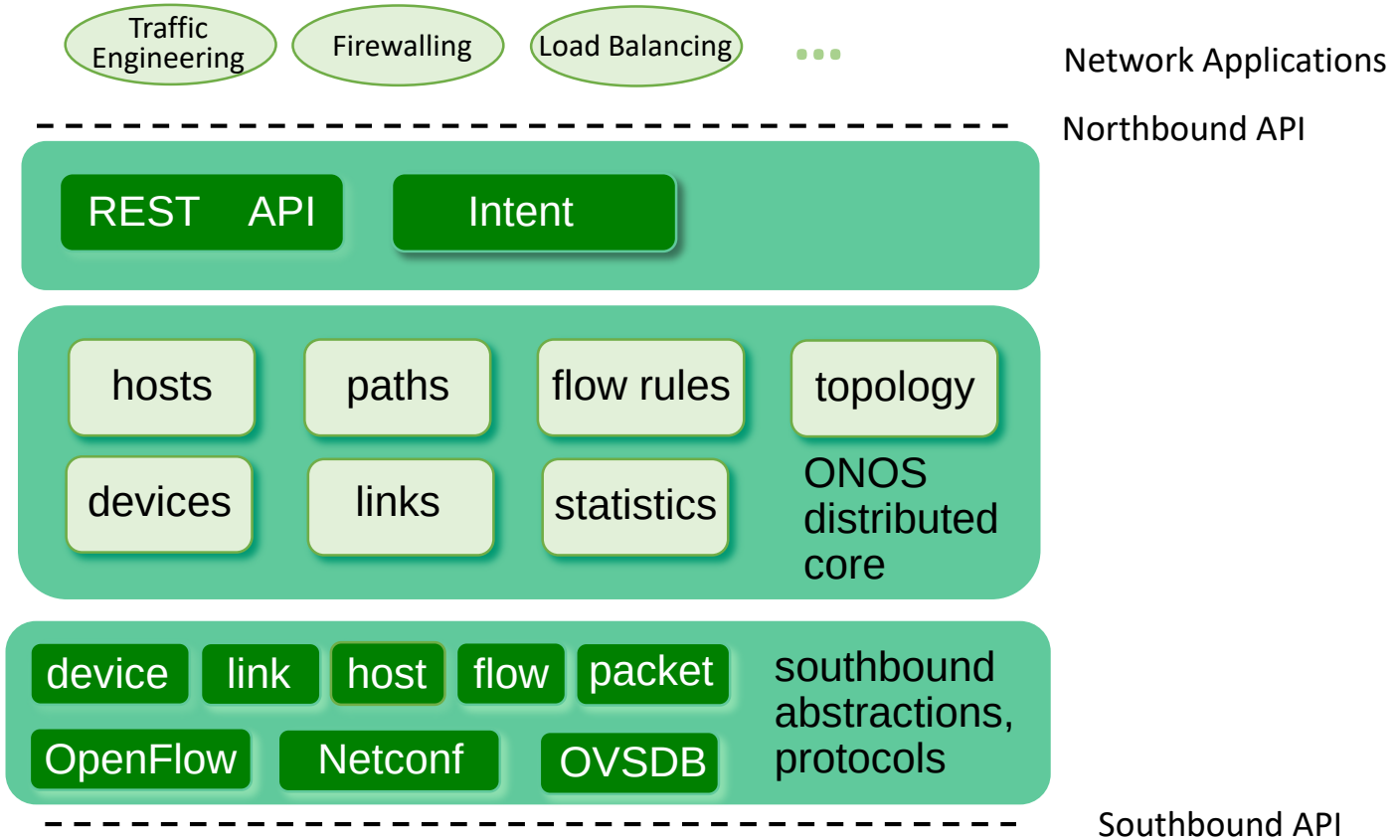


Service Abstraction Layer:

- interconnects internal, external applications and services

-
ODL از جمله کنترلرهایی است که اجازه میده یکسری از فانکشن ها توی خود کنترلر قرار داشته باشن و خودش اینارو ارائه میده در واقع

ONOS controller



- control apps separate from controller
- intent framework: high-level specification of service: what rather than how
- considerable emphasis on distributed core: service reliability, replication performance scaling

-
ONOS: یک کنترلر دیگه ای است که در اینجا هیچکدوم از app های تو کنترلر نیستند و همشون باید روش قرار داده بشن

همه این کنترلرها کار مورد نظر رو انجام میدن

SDN: selected challenges

- hardening the control plane: dependable, reliable, performance-scalable, secure distributed system
  - robustness to failures: leverage strong theory of reliable distributed system for control plane
  - dependability, security: “baked in” from day one?
- networks, protocols meeting mission-specific requirements
  - e.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling: beyond a single AS
- SDN critical in 5G cellular networks

-
چالش هایی که SDN داره:
SDN علارغم اینکه سرعت توسعه خوبی داشته و تکامل خوبی پیدا کرده ولی هنوز خیلی کار داره
بحث های robustness و مقاومت در مقابل failure ها : مثلا اگر کنترلر ما خودش هنگ کنه یا مثلا توی کنترلر مشکل پیدا بشه این ها نباید اتفاق بیوفته چون کنترلر خیلی مهمه برای شبکه و باید اطمینان حاصل بشه که اگر چنین مسائلی اتفاق افتاد شبکمون لطمه نخوره
و مسئله dependability, security مسائلی است که برای کنترلر خیلی احساس اند و خیلی مهم است
و داره روی این ها کار میشه

SDN and the future of traditional network protocols

- SDN-computed versus router-computed forwarding tables:
  - just one example of logically-centralized-computed versus protocol computed
- one could imagine SDN-computed congestion control:
  - controller sets sender rates based on router-reported (to controller) congestion levels

How will implementation of network functionality (SDN versus protocols) evolve?