معرفی شبکه‌های نرم‌افزار محور (SDN)

مجید طالقانی (کارشناس ارشد شبکه‌های کامپیوتری و ارتباطی)

Dec, ۲۰۱۵

چکیده

امروزه اینترنت یک جامعه دیجیتال را بوجود آورده است. به گونه‌ای که همه چیز به یکدیگر متصل‌اند و به همه چیز می‌توان از هر جایی د ستر سی دا شت. با وجود مقبولیت‌های گ سترده اینترنت، شبکه‌های آی‌پی سنتی[۱] پیچیده بوده و مدیریت آنها نیز بسیار سخت می‌باشد. به عنوان مثال پیکربندی شبکه طبق سیاست‌های از قبل تعیین شده سخت بوده و همچنین دوباره پیکربندی آنها به دلیل پاسخ به خرابی و یا تغییر بار شبکه نیز مشکل می‌باشد. شبکه‌های امروزی به طور عمودی مجتمع می‌باشند، به عبارتی دیگر سطوح داده[۲] و کنترل[۳] از یکدیگر جدا نمی‌با شند. شبکه‌های نرم‌افزار محور[۴] یک معماری شبکه جدیدی می‌با شد که قول می‌دهد اجتماع شبکه‌های امروزی را شک سته و سطوح داده و کنترل را از یکدیگر جدا نماید. به عبارت دیگر منطق کنترل شبکه را از روترها و سوییچ‌های زیرین جدا کرده و کنترل شبکه را به صـورت منطقی متمرکز کرده و قابلیت برنامه نویسـی (برنامه‌ریزی) شـبکه را فراهم می‌کند. شـبکه‌های نرم‌افزار محور، ساخت و معرفی انتزاع های[۵] شبکه را آسان کرده، در نتیجه مدیریت شبکه را ساده می‌کند و شرایط تحول شبکه را فراهم می‌کند.

در این مقاله سعی می کنیم که برر سی کاملی از شبکه های نرم افزار محور دا شته با شیم. در ابتدا ضمن معرفی این شبکه‌ها، تاریخچه این شبکه‌ها نیز برر سی شده ا ست. و در نهایت نبز به برر سی جامع اجزای ت شکیل دهنده معماری شبکه‌های نرم افزار محور و کارها و تلاش‌هایی که در خ صوص ا ستاندارد سازی این نوع شبکه ها انجام شده است، اشاره می‌کنیم.

---

[۱] IP Networks

[۲] Data plane

[۳] Control Plane

[۴] Software Defined Networking

[۵] Abstractions

واژه‌های کلیدی:

شبکه‌های‌نرم‌افزار محور، OpenFlow، سطح کنترل، سطح داده، سطح مدیریت[1]

[1] Management Plane

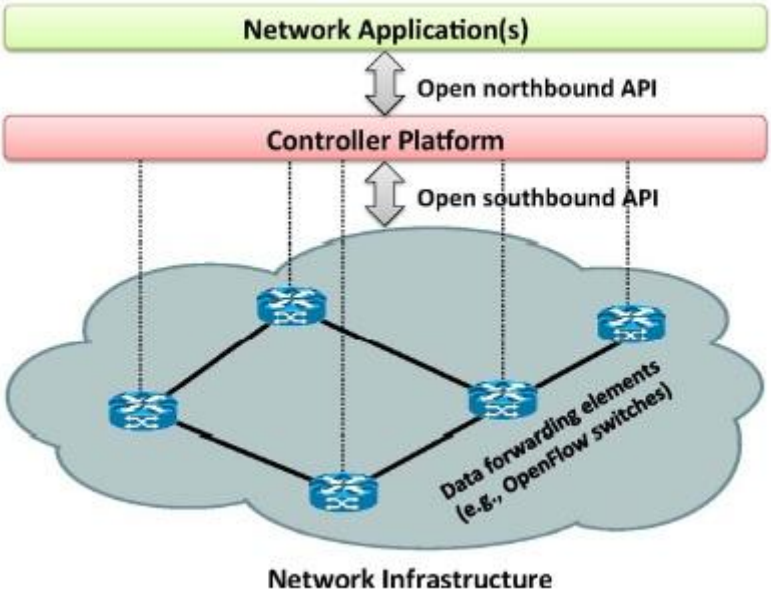
۱-۱- مقدمه

کنترل توزیع شـده و پروتکل‌های انتقال شـبکه که در روترها و سـوئیچ‌ها اجرا می‌شـوند، فناوری‌های کلیدی می‌باشـند که به اطلاعات اجازه می‌دهند در قالب بسـته‌های دیجیتال در سـطح جهان انتقال داده شوند. بر خلاف مقبولیت گ سترده اینترنت، شبکه های آی‌پی سنتی پیچیده بوده و مدیریت آنها ب سیار سخت می‌باشد، زیرا برای اینکه بتوان سیاست‌های سطح بالای دلخواه را به شبکه اعمال کرد، اپراتورهای شبکه می‌بایستی هر دستگاه شبکه را با استفاده از دستورات سطح پایین اختصاصی هر دستگاه و شرکت تولید کننده، جداگانه پیکربندی کنند. علاوه بر پیچیدگی‌های پیکربندی، محیط های شـبکه می‌بایسـتی خطاهای پویا را نیز تحمل کرده و با توجه به تغییرات بار شـــبکه، خود را وفق دهند. دوباره پیکربندی خودکار و مکانیزم‌های پا سخگویی در شبکه های امروزی به طور کامل، قابل اجرا نبوده و از طرفی اجرای سیاست‌های مورد نیاز در محیط های پویا بسیار چالش برانگیز می‌باشد.[۱]

شبکه های امروزی به صورت عمودی مجتمع می‌با شند. سطح کنترل (ت صمیم می‌گیرد که به چه صورت با ترافیک شبکه رفتار شود) و سطح داده (که ترافیک را با توجه به تصمیمات گرفته شده تو سط واحد کنترل، ارسـال میکند) در تجهیزات شـبکه یکپارچه بوده، در نتیجه باعث کاهش انعطاف پذیری آن شده و مانع تحول و نوآوری زیر ساخت شبکه می‌شود.

شـبکه‌های نرم افزار محور یک معماری نوظهور شـبکه بوده که این امید را می‌دهد که محدودیت‌های زیرسـاخت‌های شـبکه‌های کنونی را تغییر بدهد. ابتدا اتصـال عمودی آن را از طریق جداسـازی منطق کنترل شبکه (سطح کنترل) از سوئیچ ها و روترهای زیرین که ترافیک را ار سال می‌کنند (سطح داده) می‌شکند. دوماً با جدا سازی سطوح داده و کنترل، سوئیچ های شبکه تبدیل به تجهیزات ساده‌ای می‌شوند که تنها ترافیک را ار سال می‌کنند و منطق کنترلی آنها در یک کنترلر [۱] منطقاً متمرکز (یا سی ستم عامل شبکه) پیاده سازی می‌شود. که در نتیجه باعث ساده سازی اجرای سیاست‌های (دوباره) پیکربندی شبکه و تکامل شبکه می‌شود. یک نمای ساده از معماری شبکه‌های نرم افزار محور در شکل (۱-۱) نشان داده می‌شود[۱].

[۱] Controller



شکل (۱-۱) نمایی ساده از معماری شبکه های نرم افزار محور

بسیار مهم می‌باشد که بیان شود منظور از این جمله که منطق کنترل شبکه به صورت منطقی متمرکز شده است، بدین معنی نیست که تنها یک سیستم متمرکز فیزیکی وجود دارد. در واقع برای اینکه میزان کارایی به طور مناسب و کافی تضمین شود و مقیاس پذیری و قابلیت اطمینان داشته باشیم، نمی‌توان سطح کنترل را در یک کنترلر متمرکز فیزیکی پیاده سازی کرد، در عوض چندین کنترلر وجود خواهد دا شت که به صورت فیزیکی توزیع شده‌اند و سطح کنترل در این کنترلرهای توزیع شده پیاده سازی می‌شود.

جداسازی سطوح داده و کنترل با تعریف یک واسط‌های برنامه‌نویسی[۱] خوش تعریف معنا پیدا می‌کند که کنترلر از طریق این وا سط تجهیزات شبکه را برنامه‌ریزی می‌کند. در شکل (۱-۱) این مفهوم ن شان داده شده است.

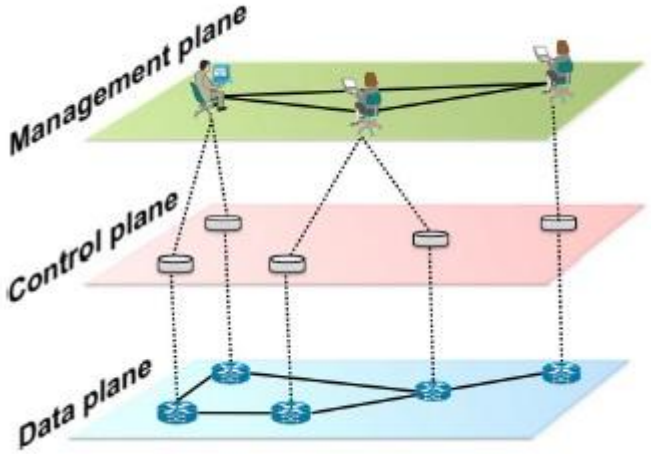
برجسته‌ترین مثال از همچین واسط برنامه‌نویسی، پروتکل OpenFlow می‌باشد که در ادامه به تشریح آن می پردازیم.

اگرچه شبکه‌های نرم افزار محور و OpenFlow ابتدا یک موضوع آکادمیک بود اما امروزه توجه صنعت نیز به آن جلب شده است. امروزه بیشتر تولید کننده‌های تجهیزات شبکه، سوییچ هایی را روانه بازار

[۱] Application Programming Interface

کرده‌اند که از OpenFlow پشـتیبانی می‌کنند. به بعضـی از این سـوییچ ها اصـطلاحا Pure OpenFlow گفته می شود به این معنی که این سوییچ ها تنها در شبکه های نرم افزار محور کاربرد داشته و نمی توان این سوییچ ها را در شبکه های سنتی نیز مورد ا ستفاده قرار داد زیرا این سوییچ ها فاقد منطق کنترلی بوده و فقط سطح داده در آنها پیاده سازی شده و تنها داده ها را فوروارد می کنند. در مقابل بعضی شرکت ها سوییچ هایی را روانه بازار کرده که هم در شبکه های فعلی می توان از این سوییچ ها ا ستفاده کرد و هم در شـبکه های نرم افزار محور، که اصـطلاحا به این سـوییچ ها OpenFlow-Enabled گفته میشـود. موضـوع شـبکه های نرم افزار محور به اندازه‌ای جالب و مهم بوده که شـرکت‌هایی نظیر Facebook ، Google ، Yahoo و Microsoft به این مو ضوع وارد شده و پروژه‌هایی را نیز در این رابطه انجام داده‌اند. به ویژه گوگل که با ا ستفاده از شبکه های نرم افزار محور، دیتا سنترهای خود که در سرا سر دنیا پخش شده‌اند را به یکدیگر متصل کرده است.[۲]

همانطور که در شکل (۱-۲) نشان داده ا ست، شبکه های کامپیوتری می‌توانند به سه سطح تقسیم شوند: سطوح داده، کنترل و مدیریت.



شکل (۱-۲) نمایی لایه ای از عملکرد های شبکه ای

سطح داده معادل تجهیزات شبکه بوده که وظیفه دارند داده ها را به صورت کارآمد پیش‌بری[۱] کنند. سـطح کنترل بیان‌کننده پروتکل‌هایی می‌باشـد که جدول‌های پیش‌بری تجهیزات شـبکه را پر می‌کنند. سطح مدیریت شامل سرویس‌های نرم افزاری شبکه نظیر برنامه‌های کاربردی مسیریابی، امنیتی، کنترل

[۱] Forward

دسترسی، مدیریتی شبکه و غیره می باشد. سیاست‌های شبکه در سطح مدیریت تعریف می‌شوند و سطح کنترل سیاست‌ها را اجرا می‌کند و سطح داده نیز طبق سیاست واحد کنترل، داده ها را ارسال می‌کند.

۱-۱-۲- شبکه های نرم افزار محور چیست؟

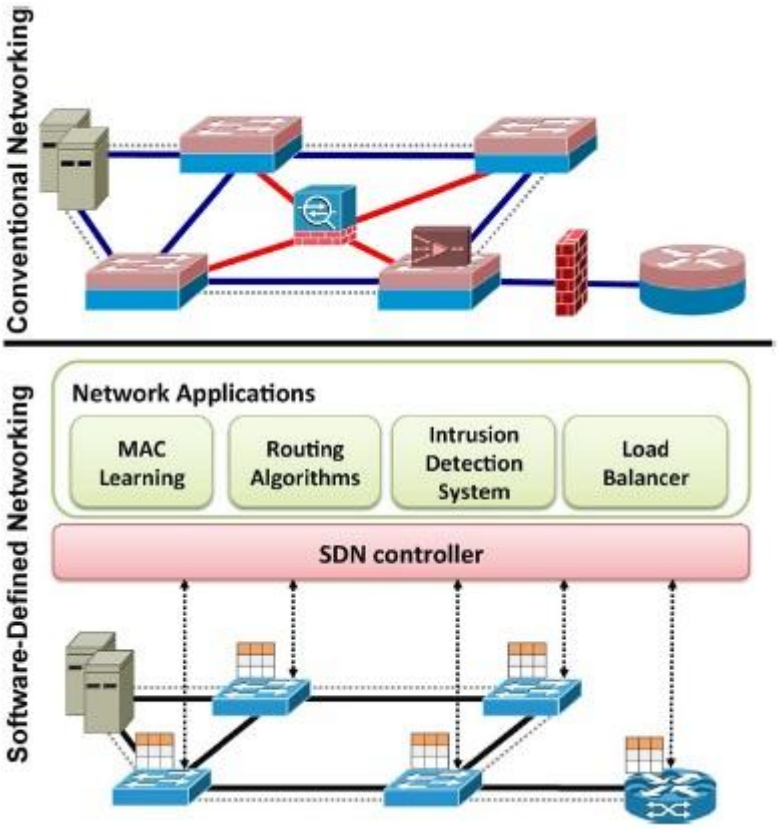
واژه شبکه های نرم‌افزار محور(SDN) اولین بار در تحقیقات صورت گرفته در مورد پروتکل OpenFlow در دانشگاه استنفورد مطرح شد [۳]. همانطور که اشاره شد SDN اشاره به معماری شبکه‌هایی دارد که در آن سطح داده‌ای که سوییچ‌ها می‌باشند، توسط یک کنترل‌کننده‌ی راه دور که از تجهیزات جدا شده‌است، مدیریت می‌شود.

ما شبکه‌های نرم افزار محور را معماری شبکه‌ای با چهار رکن زیر تعریف می‌کنیم:

۱. سطوح داده و کنترل از یکدیگر جدا هستند. عملکردهای کنترلی از تجهیزات شبکه حذف شده‌اند و در نتیجه تجهیزات شبکه به المان‌های پیش‌بری ساده مبدل شده‌اند.
۲. تصمیمات پیش‌بری بجای اینکه بر طبق آدرس مقصد انجام شوند، مبتنی بر جریان انجام می‌شوند. در شبکه های نرم افزار محور، یک جریان دنباله‌ای از بسته ها بین یک مبدا و مقصد می‌باشد که دارای مقادیر مشترک در بعضی فیلدهای سرآیندشان هستند. همه بسته‌های یک جریان، سرویس مشابهی در تجهیزات شبکه می‌گیرند.
۳. منطق کنترل به یک موجودیت خارجی انتقال داده شده که به آن کنترلر شبکه‌های نرم افزار محور و یا سیستم عامل شبکه[۱] می‌گویند.
۴. شبکه را می‌توان بوسیله برنامه‌های کاربردی که در بالای سیستم عامل شبکه در حال اجرا بوده و با دستگاه های سطح داده زیرین تعامل دارند، برنامه ریزی کرد.

همان طور که قبلاً نیز بیان شد اتصال محکم بین سطوح داده و کنترل، افزودن کارکردهای جدید به شبکه‌های سنتی را بسیار سخت کرده است. این واقعیت در شکل (۱-۳) نمایش داده شده است.

[۱] Network Operating System



شکل (۱-۳) شبکه های سنتی در مقایسه با شبکه های نرم افزار محور

اتصال محکم سطوح داده و کنترل، توسعه و استقرار ویژگی‌های شبکه‌ای جدید (مانند الگوریتم‌های مسیریابی) را بسیار مشکل کرده است. زیرا بدین منظور بایستی سطح کنترل تمام تجهیزات شبکه را بروز رسانی کنیم (از طریق نصب سفت افزارهای[۱] جدید و یا ارتقاع سخت افزاری). بنابراین معمولاً این ویژگی‌های جدید شبکه‌ای از طریق تجهیزات سخت افزاری گران و اختصاصی که نحوه پیکربندی آنها نیز سخت می‌باشد، به نام جعبه‌های میانی[۲] ارائه میشود. به عنوان یک نمونه از جعبه‌های میانی می‌توان به فایروال‌ها و سیستم‌های تشخیص نفوذ[۳] و متوازن سازهای بار اشاره کرد. این جعبه‌های میانی به صورت استراتژیک در شبکه قرار میگیرد، در نتیجه اگر در آینده شبکه نیاز به تغییر توپولوژی یا پیکربندی داشته باشد، انجام آنها با مشکل روبرو می‌شود.

[۱] Firmware

[۲] MiddleBox

[۳] Intrusion Detection Systems

در مقابل شبکه‌های نرم افزار محور، سطوح داده و کنترل را از یکدیگر جدا کرده و سطح کنترل را در یک موجودیت خارجی به نام کنترلر پیاده سازی می‌کند. این رهیافت دارای مزایای زیر می‌باشد:

۱. برنامه نویسی سرویس‌های شبکه و برنامه‌های کاربردی شبکه ساده‌تر می شود زیرا سطوح تجریدی که کنترلر و یا زبان‌های برنامه نویسی سطح بالا ارائه می‌دهند، می‌توانند به اشتراک گذاشته شوند.
۲. تمام برنامه‌های کاربردی می‌توانند از مزیت دسترسی به اطلاعات شبکه مشترک (دید جهانی) استفاده کرده که در نتیجه منجر به تصمیمات سیاست کارآمد و سازگار می‌شود.
۳. این برنامه‌های کاربردی می‌توانند از هر کجای شبکه، اقدامات لازم را انجام دهند (دوباره پیکربندی تجهیزات شبکه)، بنابراین دیگر نیازی نیست یک استراتژی دقیق طراحی کرد که یک کارکرد جدید را در کجای شبکه قرار داد.
۴. ادغام کردن برنامه‌های کاربردی ساده‌تر می‌شود. به عنوان مثال برنامه‌های کاربردی متوازن سازی بار و مسیریابی می‌توانند با یکدیگر ادغام شوند به طوری که تصمیمات متوازن سازی بار بر سیاست‌های مسیریابی مقدم تر می‌باشند.

۱-۱-۳- تاریخچه شبکه های نرم افزار محور

اگر چه شبکه‌های نرم افزار محور یک معماری شبکه‌ای است که اخیراً سر و صدا کرده ولی شبکه‌های نرم افزار محور خود از ایده‌های شبکه‌ای استفاده می‌کند که قبل از او وجود داشته است. به ویژه شبکه‌های نرم افزار محور از کارها و ایده‌هایی که برای ساخت شبکه‌های قابل برنامه ریزی[۱] از جمله شبکه قابل برنامه‌ریزی ATM، شبکه‌های فعال[۲] و همچنین طرح‌هایی که برای جدا سازی سطوح کنترل و داده ارائه شده از جمله NCP[۳] و RCP[۴] استفاده می‌کند.

برنامه ریزی سطوح داده[۵] تاریخ طولانی دارد. در واقع برای اولین بار شبکه های فعال بودند که سعی

[۱] Programmable Networks

[۲] Active Networks

[۳] Network Control Point

[۴] Routing Control Platform

[۵] DataPlane Programmability

در ساخت معماری شبکه جدیدی مبتنی بر این مفهوم داشتند. ایده اصلی پشت شبکه‌های فعال این بود که هرگره این توانایی را داشته باشد که بتواند محاسباتی روی بسته ها انجام داده و یا محتویات بسته‌ها را تغییر دهد.

از جمله کارهایی که اخیراً برای طراحی و گسترش دستگاه‌های سطح داده قابل برنامه‌ریزی انجام شده، می‌توان به ForCES، OpenFlow و POF اشاره کرد. این پروتکل‌ها از روشی متفاوت از شبکه های فعال برای رسیدن به این مقصود استفاده می‌کنند. در واقع مبتنی بر تغییر دستگاهای پیش‌بری به گونه‌ای که هر دستگاه دارای یک و یا چندین جدول جریان باشد که این جداول به وسیله یک موجودیت راه دور و از طریق عملیات هایی نظیر افزودن[۱]، حذف کردن[۲] و ویرایش[۳] پیکربندی می‌شوند.

اولین اقداماتی که برای جدا سازی سطوح داده و کنترل انجام شد به دهه ۸۰ و ۹۰ میلادی باز می‌گردد. احتمالا NCP اولین اقدام جهت جداسازی سطوح داده و کنترل بود. NCP توسط AT&T برای بهبود مدیریت و کنترل شبکه تلفن خود ارائه شد.

NCP بدلیل استفاده از دید جهانی شبکه باعث سرعت بخشیدن به ابتکار در شبکه شده و همچنین کارآمدی شبکه را نیز بهبود داد.

اخیراً پروژه‌های دیگری نظیر SANE، Ethane، OpenFlow، NOX، و POF نیز جداسازی سطوح کنترل و داده را مطرح کرده‌اند. جالب اینکه، این راه حل‌های اخیر نیازمند انجام تغییرات اساسی در تجهیزات پیش‌بری نیستند، که در این صورت نه تنها برای جامعه تحقیقاتی شبکه جالب می‌با شد بلکه برای صنعت شبکه نیز مفید می‌با شد. به عنوان مثال تجهیزات مبتنی بر OpenFlow می‌توانند به راحتی با تجهیزات اترنت سنتی هم‌زیستی دا شته با شند که در نتیجه به منظور برپا سازی OpenFlow می‌توان این شبکه‌ها را در کنار شبکه‌های سنتی بر پا کرد و به مرور زمان دستگاهای مبتنی بر OpenFlow بیشتری را به شبکه اضافه کرد.

مفهوم سیستم عامل شبکه با معرفی سیستم عامل‌های شبکه مبتنی بر OpenFlow مانند NOX، ONIX وONOS دوباره متولد شد. سیستم عامل‌های شبکه چندین دهه ا ست که وجود دارند. یکی از معروف ترین آنها IOS سیسکو می‌باشد که در اوایل دهه ۹۰ میلادی معرفی شد. دیگر سیستم عامل‌هایی

[۱] Adding

[۲] Removing

[۳] Updating

که سزاوار نام بردن هستند می‌توان به JUNOS، ExtreamXOS و SR OS اشاره کرد.

این سیستم عامل‌های شبکه، سخت افزار زیرین را برای اپراتورهای شبکه تجرید کرده، کنترل زیر ساخت شبکه را آسان‌تر کرده و همچنین توسعه و گسترش پروتکل‌های جدید و برنامه های کاربردی مدیریتی را ساده کرده است.

۱-۲- اجزای تشکیل دهنده شبکه های نرم افزار محور

همان طور که در شکل (۱-۴) (ب) نشان داده شده است، میتوان معماری شبکه‌های نرم افزار محور را به صورت ترکیبی از لایه‌های مختلف نمایش داد که هر لایه وظایف ویژه خود را دارد.

بعضی از این لایه ها همیشه در پیاده سازی‌های شبکه‌های نرم افزار محور وجود دارند از جمله واسط برنامه نویسی جنوبی[۱] و سیستم عامل شبکه و واسط برنامه نویسی شمالی[۲] و برنامه‌های کاربردی شبکه[۳]. در حالی که بعضی از لایه های شبکه‌های نرم افزار محور تنها در بعضی کاربردها و پیاده سازی‌های خاص وجود دارند مانند لایه‌های مدیر مجازی سازی[۴] و مجازی سازی مبتنی بر زبان‌های برنامه نویسی[۵].

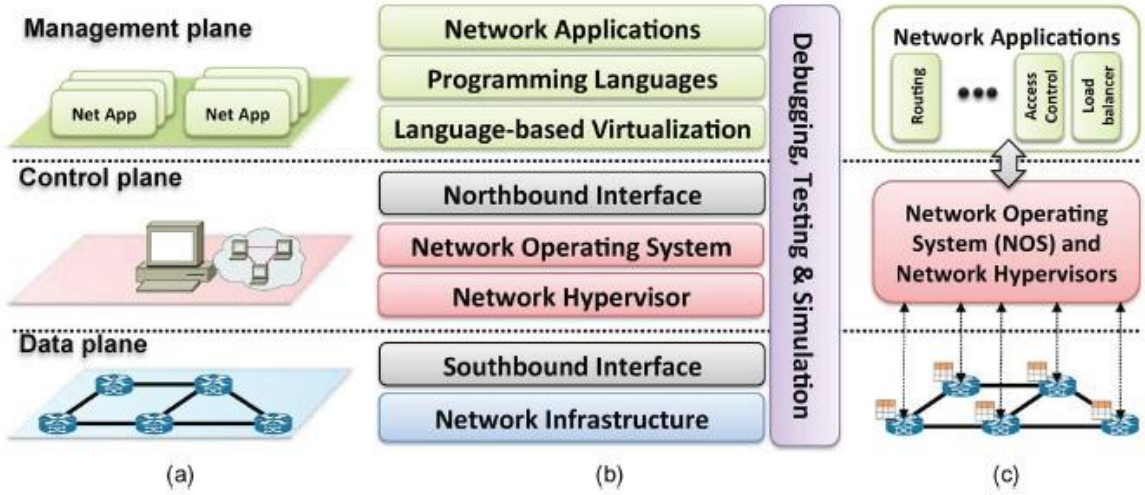
[۱] Southbound API

[۲] Northbound API

[۳] Network Applications

[۴] Hypervisor

[۵] Language-based virtualization



شکل (۱-۴) شبکه های نرم افزار محور الف) سطوح ب) لایه ها ج) معماری طراحی سیستم

لایه های SDN در شکل (۱-۴) (ب) نمایش داده شده است و شکل (۱-۴) (الف) و شکل (۱-۴) (ج) نیز شبکه‌های نرم افزار محور را از جهت سطوح آن و از جهت طراحی سیستمی نشان می‌دهد.

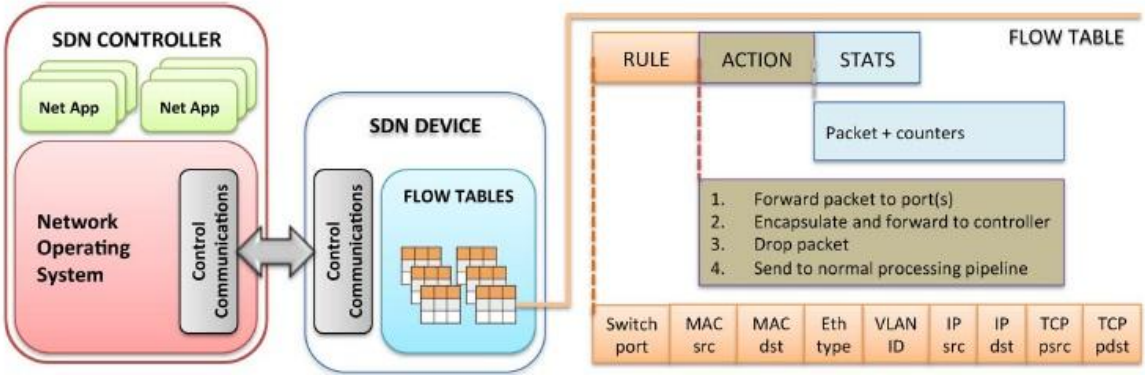
در ادامه لایه‌های شبکه‌های نرم افزار محور را از پایین به بالا برر سی کرده و ویژگی ها و مفاهیم مهم آنها نیز گفته می‌شود.

۱-۲-۲- لایه ۱: زیر ساخت

زیر ساخت[۱] شبکه‌های نرم افزار محور همانند شبکه‌های سنتی، ترکیبی از تجهیزات شبکه‌ای از جمله روتر و سوییچ و غیره می‌باشند . تفاوت اصلی در این واقعیت است که این تجهیزات اکنون به دستگاه‌های ساده‌ای تبدیل شده اند که فقط عمل پیش‌بری انجام می‌دهند و کنترلر در یک سیستم کنترل منطقاً مرکزی قرار گرفته ا ست. به عبارت دیگر به مانند قبل، منطق کنترل شبکه به صورت توزیع شده پیاده سازی نشده است و یک کنترلر مرکزی وجود دارد که کل شبکه را کنترل می‌کند. مهم تر اینکه این شبکه‌های جدید بر بالای یک واسط استاندارد و باز (به عنوان مثال OpenFlow) ساخته می‌شوند، زیرا در این صورت می‌توان پیکربندی و سازگاری ارتباط و قابلیت همکاری بین سطوح داده و کنترل دستگاه‌های مختلف را تضمین کرد. کاری که در شبکه های سنتی به دلیل طبیعت توزیعی بودن سطح کنترل و واسطه های بسته و اختصاصی به سختی انجام می‌شد.

[۱] Infrastructure

همان طور که در شکل (۱-۵) نشان داده شده است، در یک معماری SDN/OpenFlow، دو المان مهم وجود دارد. کنترلر ها و دستگاه های پیش‌بری.



شکل (۱-۵) دستگاه های شبکه های نرم افزار محور مبتنی بر OpenFlow

یک دستگاه سطح داده یک المان سخت افزاری و یا نرم افزاری می‌باشد که وظیفه اصلی آن ارسال بسته ها می‌باشد. در حالی که کنترلر یک پشته نرم افزاری می‌باشد که در واقع مغز شبکه بوده و در یک بستر سخت افزاری Comodity اجرا می‌شود.

یک دستگاه پیش‌بری که از OpenFlow پشتیبانی می‌کند دارای چندین جدول جریان می‌باشد که هر جدول جریان از سه قسمت تشکیل شده است: ۱- قوانین تطابق[۱] ۲- عملیاتی که باید بر روی بسته های تطبیقی انجام شود ۳- شمارنده‌هایی که آمار بسته های تطبیق شده را نگهداری می‌کند.

هنگامی که بسته‌ها وارد دستگاه می‌شوند، این بسته به ترتیب از تعدادی جدول جریان عبور داده می شود تا مشخص شود که چه رفتاری با بسته وارد شده انجام شود. قوانین تطابق می‌توانند ترکیبی از پورت های ورودی، اترنت و فیلدهای سرآیند آی‌پی نسخه ۴ و پورت های سطح بالا یا متادیتا باشند. اگر بسته‌های ورودی با هیچ کدام از قوانین تطابق رکوردهای جداول جریان تطابق نداشتند، بسته مورد نظر دورانداخته[۲] می شود مگر اینکه یک رکورد پیش فرض در جداول جریان تعریف شود که با این بسته‌ها به چه نحوی برخورد شود. از جمله عملیاتی که می‌شود بر روی بسته انجام داد : ۱- ارسال بسته به یکی از پورت‌های خروجی ۲- بسته کپسوله شده و به کنترلر ارسال شود ۳- بسته دورانداخته شود ۴- بسته به

[۱] Matching Rule

[۲] Drop

پردازش خط لوله[1] عادی سوییچ ارسال شود ۵- بسته به جدول جریان بعدی ارسال شود.

۱-۲-۳- لایه ۲ : واسط های برنامه نویسی جنوبی

واسطهای برنامه نویسی جنوبی، پل ارتباطی المان های پیش‌بری و کنترلر می‌باشند. بنابراین یک ابزار حیاتی برای جدا سازی عملکردهای سطوح داده و کنترل می‌با شد. این وا سط‌های ا ستاندارد قابلیت همکاری بین دستگاه‌های شبکه شرکت‌های مختلف را فراهم می‌کنند. به عبارتی می‌توان هنگام راه اندازی شبکه سازمانی، تجهیزات شبکه را از مح صولات شرکت‌های مختلف انتخاب کرد و مطمئن بود که این تجهیزات قابلیت همکاری با یکدیگر دارند.

در حال حاضر OpenFlow مقبول‌ترین و پرکاربردترین واسط برنامه نویسی جنوبی می‌باشد.

پروتکل OpenFlow سه منبع اطلاعاتی برای سیستم عامل‌های شبکه فراهم می‌کند:

۱. هنگامی که وضعیت پورت و یا لینکی تغییر کند، یک پیام رویدادی به وسیله دستگاه پیش‌بری به کنترلر ارسال می شود.
۲. آمارهای جریان که توسط دستگاه‌های پیش‌بری به وجود می آیند، توسط کنترلر جمع‌آوری می‌شوند .
۳. بسته‌های ورودی به دستگاه‌های پیش‌بری که با هیچ کدام از قوانین تطابق رکوردهای جداول جریان، تطابق ندارند به کنترلر ارسال می شوند. این کانالهای اطلاعاتی برای اینکه بتوان اطلاعات مبتنی بر جریان به سیستم عامل‌های شبکه ارائه داد، بسیار مهم می‌باشند.

اگر چه OpenFlow معروف‌ترین واسط برنامه‌سازی جنوبی برای شبکه های نرم افزار محور می‌باشد، اما تنها واسط برنامه‌سازی جنوبی نمی‌باشد. از دیگر واسط‌های برنامه‌سازی جنوبی می‌توان به ForCES[۴]، OVSDB[۷][2]، POF[۵]، OpFlex[۸]، OpenState[۹]، ROFL[۱۰][3]، HAL[۱۱]، و PAD[۱۴][4] اشاره کرد.

---

[1] Pipeline

[2] Open vSwitch Database

[3] Revised Open-Flow Library

[4] Programmable Abstraction of Data Path

۱-۲-۴- لایه ۳: مدیر مجازی سازی

مجازی سازی قبل از اینکه وارد دنیای شبکه شود پیش از این در کامپیوترهای مدرن وجود داشت. و پیشرفت‌های سریع دهه گذشته باعث شد که مجازی سازی بسترهای محاسباتی، مورد قبول همگان واقع شـوند به طوری که طبق آخرین گزارشـها، تعداد سـروهای مجازی از تعداد سـرورهای فیزیکی بیشـتر می‌باشد.[۱۲]

مدیر مجازی سـازی این امکان را به ماشـین های مجازی متمایز می‌دهد که از منابع سـخت‌افزاری یکسان به طور مشترک استفاده کنند.

در یک ابر زیر ساخت به عنوان سرویس[۱]، هر کاربر می‌تواند منابع مجازی خود، از محاسباتی گرفته تا ذخیره‌سـازی را داشـته باشـد.که این امکان، منبع درآمدی جدیدی را به وجود می‌آورد، چرا که کاربران منابع درخوا ستی خود را هر زمانی که نیاز دا شته با شند از یک زیر ساخت فیزیکی به ا شتراک گذا شته شده، به قیمت پایین دریافت می‌کنند.

به طور همزمان، ارائه دهندگان منابع نیز از زیر ساخت‌های فیزیکی ن صب شده خود، ا ستفاده بهتری می‌کنند.

اما متا سفانه برخلاف پی شرفت‌هایی که در مجازی سازی محا سباتی و ذخیره سازی دا شته‌ایم، در صعنت شبکه، مجازی سازی خیلی پیشرفت نداشته است و هنوز تجهیزات شبکه تک به تک پیکربندی می‌شوند.

نیازمندی‌های مهم شـبکه را می‌توان از طریق دو بعد برآورده کرد: توپولوژی شـبکه – فضـای آدرس دهی.

بارهای کاری متفاوت نیازمند توپولوژی شبکه متفاوت و سرویس‌های متفاوتی نیز می‌باشند. به عنوان مثال سرویس‌های لایه ۲ ای کامل و یا لایه ۳ و یا حتی سرویس‌های پیچیده لایه ۴ تا ۷ برای عملکردهای پیشـرفته. در حال حاضـر بسـیار سـخت می‌باشـد که یک توپولوژی فیزیکی واحد بتواند درخواسـت‌های گوناگون سرویس‌ها و برنامه‌های کاربردی را پشتیبانی کند. به طور مشابه، فضای آدرس دهی نیز بسیار سخت می‌با شد که در شبکه‌های جاری تغییر کند. امروزه بارهای کاری مجازی نیز باید در همان فضای آدرس دهی زیر ساخت فیزیکی فعالیت کنند. بنابراین بسیار سخت می‌باشد که پیکربندی شبکه اصلی را برای یک مستاجر حفظ کرد. همچنین ما شین‌های مجازی نمی‌توانند به سرورهای دلخواه مهاجرت کنند

[۱] Infrastructure as a Service

و طرح آدرس دهی ثابت بوده و به سختی تغییر می‌کند. به عنوان مثال اگر د ستگاه‌های پیش‌بری زیرین تنها از آی‌پی نسخه ۴ پشتیبانی کنند، ماشین‌های مجازی یک مستاجر نمی‌توانند از آی‌پی نسخه ۶ استفاده کنند.

برای ارائه مجازی سازی کامل، زیر ساخت شبکه باید این توانایی را دا شته با شد که از طرح آدرس دهی و توپولوژی شبکه دلخواه پشتیبانی کند. هر مستاجر باید بتواند گره‌های محاسباتی و شبکه‌ای را به طور همزمان پیکربندی کند. امید است که شبکه‌های نرم افزار محور با کمک تکنیک‌های تونل زنی[۱] جدید از جمله VXLAN و NVGRE بتواند این وضعیت را تغییر دهد. به عنوان مثال راه حل‌هایی نظیر FlowVisor، FlowN، NVP، OpenVirteX، IBM SDN VE، RadioVisor، AutoVFlow، eXtensible Datapath Daemon (xDPd و غیره اخیراً ارائه، ارزیابی و در سناریوهای واقعی پیاده سازی شده‌اند.

۱-۲-۵- لایه ۴ : سیستم عامل های شبکه / کنترلر ها

سیستم عامل‌های سنتی، برای دسترسی به دستگاه‌های سطح پایین و مدیریت دسترسی هم زمان به منابع زیرین (به عنوان مثال دیسک‌های سخت، کارت شبکه، پردازنده، حافظه) و فراهم کردن مکانیزم‌های حفاظتی امنیتی، سطوح تجرید را فراهم می‌کنند (به عنوان مثال واسط‌های برنامه نویسی سطح بالا). این تابع ها برای افزایش بهره‌وری و برای اینکه برنامه نویسان کاربردی و سیستمی زندگی آ سان‌تری دا شته باشند، بسیار ضروری می‌باشند.

در مقایسه، شبکه ها مدت زمان زیادی است که از طریق دستور العمل‌های سطح پایین، مخصوص هر دستگاه و سیستم‌عامل (به عنوان مثال،IOS سیسکو و Junos جونیپر) مدیریت و پیکربندی می‌شوند. به علاوه اینکه این ایده سیستم‌عامل‌ها که ویژگی‌های مخصوص هر دستگاه را تجرید کرده و به صورت یکسری تابع‌هایی فراهم می‌شود، همچنان در شبکه ها مورد استفاده قرار نمی‌گیرند. به عنوان مثال امروزه طراحان پروتکل‌های مسیریابی هنگام رفع کردن مشکلات شبکه نیاز دارند که با الگوریتم‌های توزیعی پیچیده سر و کار داشته باشند.

شبکه‌های نرم افزار محور این قول را داده که با کنترل منطقاً مرکزی ارائه شده توسط سیستم عامل شبکه، مدیریت شبکه را آسان کرده و همچنین به رفع مشکلات شبکه کمک کند. همانند سیستم

[۱] Tunneling

عامل‌های سنتی، ارزش اصلی و حیاتی سیستم عامل شبکه، فراهم کردن تجرید، سرویس‌های خاص و واسط‌های برنامه نویسی متداول به برنامه نویسان می‌باشند. از جمله سرویس‌های سیستم‌عامل شبکه قابل ارائه به برنامه‌نویسان می توان به وضعیت شبکه و اطلاعات توپولوژی شبکه، کشف دستگاه‌ها و توزیع پیکربندی شبکه اشاره کرد.

با سیستم عامل‌های شبکه، برای تعریف یک سیاست شبکه، دیگر برنامه نویسان نیازی نیست که نگران جزییات سطح پایین توزیع داده بین المان‌های مسیریابی باشند. در چنین سیستم‌هایی با کاهش پیچیدگی ذاتی ساخت پروتکل‌ها و برنامه های کاربردی جدید، نوآوری با سرعت بیشتری اتفاق می افتد.

انواع گوناگونی از کنترلرها با معماری و طراحی متفاوتی وجود دارند. کنترلرهای موجود را می‌توان از جنبه‌های مختلفی دسته بندی کرد. از دید معماری می‌توان آنها را به دو دسته متمکز و توزیع شده تقسیم کرد.

یک کنترلر متمرکز یک موجودیت واحدی می‌باشد که تمام دستگاه‌های پیش‌بری شبکه را مدیریت می‌کند. به طور طبیعی چنین رویکردی یک نقطه شکست واحد محسوب شده و همچنین محدودیت‌های مقیاس‌پذیری نیز دارد، چرا که یک کنترلر واحد ممکن است نتواند یک شبکه با تعداد زیادی المان سطوح داده را مدیریت کند .

از کنترلرهای متمرکز میتوان به NOX-MT، Maestro، Beacon و Floodlight اشاره کرد[۱۵].

در مقابل طراحی متمرکز، یک سیستم‌عامل شبکه توزیع شده می‌تواند برای اینکه نیازهای شبکه هر محیطی، از محیط‌های کوچک تا محیط‌های بزرگ را برآورده کند، اندازه خود را افزایش دهد.

کنترلرهای Onix، HyperFlow، SDN Hp VAN، ONOS، DISCO، Yane، PANE، SMaRt-Light و Fleet نمونه‌هایی از کنترلرهای توزیع شده می‌باشند. اکثر کنترلرهای توزیع شده، مدل سازگاری ضعیف را پشتیبانی می‌کنند که این مدل سازگاری به این معنی است که بروزرسانی داده در بعضی گره‌ها، نهایتاً در همه کنترلرها بروزرسانی می‌شود. این به این معنی است که یک مدت زمانی وجود دارد که بعضی گره‌ها برای بعضی ویژگی‌ها، مقدارهای متفاوتی را می‌خوانند (مقدارهای قدیم یا مقدارهای جدید) از طرف دیگر، سازگاری قوی، این اطمینان را می دهد که همه کنترلرها بعد از هر عمل نوشتن، آخرین مقدار به روز رسانی شده ویژگی مورد نظر را می‌خوانند. صرف نظر از اینکه سازگاری قوی تاثیر منفی بر کارایی سیستم می‌گذارد، اما سازگاری قوی واسط ساده‌تری را به برنامه‌نویسان برنامه‌های کاربردی ارائه می‌دهد. به عبارت دیگر کار برنامه نویسان را آسان میکند. در حال حاضر فقط کنترلرهای Onix، ONOS، و SMaRt-Light از سازگاری قوی پشتیبانی می‌کنند. ویژگی‌های متداول دیگر کنترلرهای توزیع شده،

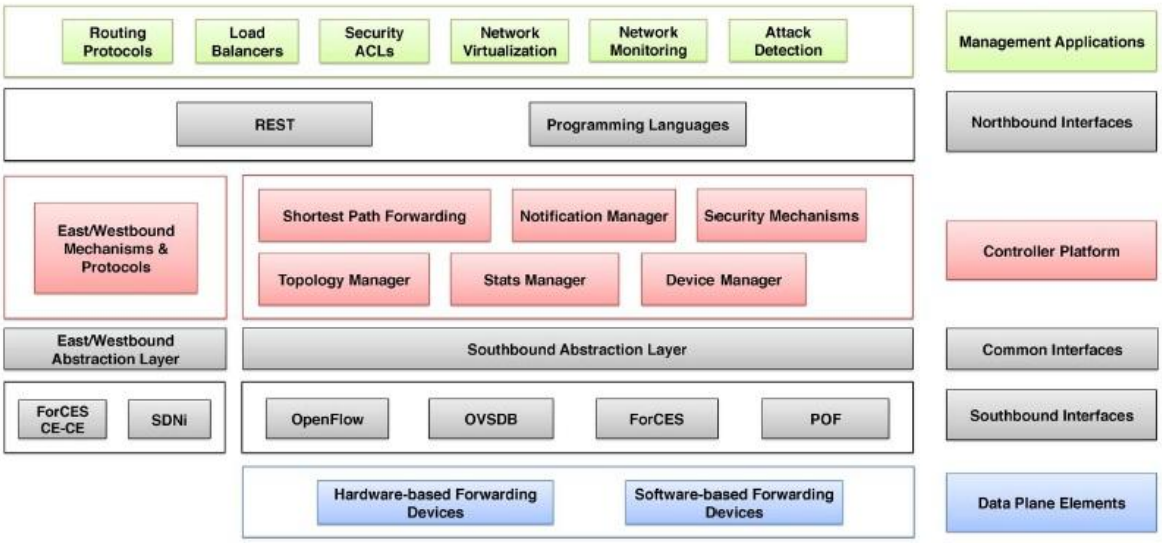
تحمل‌پذیری خطا می‌باشد. وقتی یک گره (کنترلر) خراب می‌شود، همسایه آن گره بایستی وظایف و دستگاه‌های آن گره خراب را برعهده بگیرد.

برای اینکه مروری بر معماری و مفاهیم طراحی سیستم‌عامل‌های شبکه داشته باشیم، جدول (۱-۱) ، بعضی از معماری‌های مرتبط و ویژگی‌های طراحی کنترلرهای شبکه‌های نرم افزار محور را به صورت خلا صه آورده ا ست. هر خط جدول ن شان دهنده کامپوننتی می‌با شد که در ب ستر کنترل مقیاس‌پذیر و ماژولار مهم می‌با شد. توجه به این نکته ضروری ا ست که همه کامپوننت‌ها در همه ب سترها (کنترلر ها) موجود نمی‌باشند. به عنوان مثال واسط‌های برنامه‌نویسی شرقی/غربی در کنترلرهای متمرکز مانند Beacon نیاز نمی‌باشند. در حقیقت بعضی از این کنترلرها برای کاربردهای خاصی ساخته شده‌اند به عنوان مثال برای شرکت‌های مخابراتی و یا ارائه دهندگان خدمات ابری. بنابراین نیازمندی‌ها متفاوت می‌باشند.

جدول (۱-۱) المان های طراحی و معماری بستر های کنترل

Component	OpenDaylight	OpenContrail	HP VAN SDN	Onix	Beacon
Base network services	Topology/Stats/Switch Manager, Host Tracker, Shortest Path Forwarding	Routing, Tenant Isolation	Audit Log, Alerts, Topology, Discovery	Discovery, Multi-consistency Storage, Read State, Register for updates	Topology, device manager, and routing
East/Westbound APIs	—	Control Node (XMPP-like control channel)	Sync API	Distribution I/O module	*Not present*
Integration Plug-ins	OpenStack Neutron	CloudStack, OpenStack	OpenStack	—	—
Management Interfaces	GUI/CLI, REST API	GUI/CLI	REST API Shell / GUI Shell	—	Web
Northbound APIs	REST, REST-CONF [200], Java APIs	REST APIs (configu-ration, operational, and analytic)	REST API, GUI Shell	Onix API (general purpose)	API (based on OpenFlow events)
Service abstraction layers	Service Abstraction Layer (SAL)	—	Device Abstraction API	Network Information Base (NIB) Graph with Import/Export Functions	—
Southbound APIs or connectors	OpenFlow, OVSDB, SNMP, PCEP, BGP, NETCONF	—	OpenFlow, L3 Agent, L2 Agent	OpenFlow, OVSDB	OpenFlow

با توجه به آنالیزهایی که بر روی کنترلرهای شبکه‌های نرم افزار محور مختلفی انجام شده، ما چندین المان متفاوت و معمول آن‌ها را ا ستخراج کرده که در شکل (۱-۶) ن شان داده شده ا ست. در ادامه این المان‌ها بررسی می‌شوند.



شکل (۱-۶) بستر های کنترل شبکه های نرم افزار محور : المان ها، سرویس ها، و اینترفیس ها

در اکثر بسترهای کنترلی موجود حداقل سه لایه نسبتاً به خوبی تعریف شده وجود دارد:

a. برنامه کاربردی و سرویسها
b. تابع های اصلی کنترلر
c. المان‌هایی برای ارتباطات جنوبی[۱].

۱-۲-۵-۲- تابع های اصلی کنترلر

تابع‌های سرویس شبکه پایه در واقع عملکردهای ویژه‌ای هستند که همه کنترلرها بایستی فراهم کنند. به عنوان مثال این تابع‌ها را به مانند سرویس‌های پایه‌ای که سیستم عامل‌های سنتی ارائه می‌دهند در نظر بگیرید. در سیستم عامل‌های سنتی این تابع‌های پایه عبارتند از اجرای برنامه، کنترل عملیات I/O ورودی و خروجی، ارتباطات، حفاظت و غیره. این سرویس‌ها تو سط دیگر سرویس‌های سی ستم‌عامل یا برنامه‌های کاربردی کاربران مورد استفاده قرار می‌گیرند. به طور مشابه، تابع‌هایی (عملکردهایی) نظیر توپولوژی، ارقام[۲]، پیام‌های آگاه‌سازی[۳] و مدیریت دستگاه‌ها، همگی به همراه کوتاه‌ترین مسیر حمل و نقل[۴]

[۱] Southbound Communications

[۲] Statistics

[۳] Notification

[۴] Shortest path forwarding

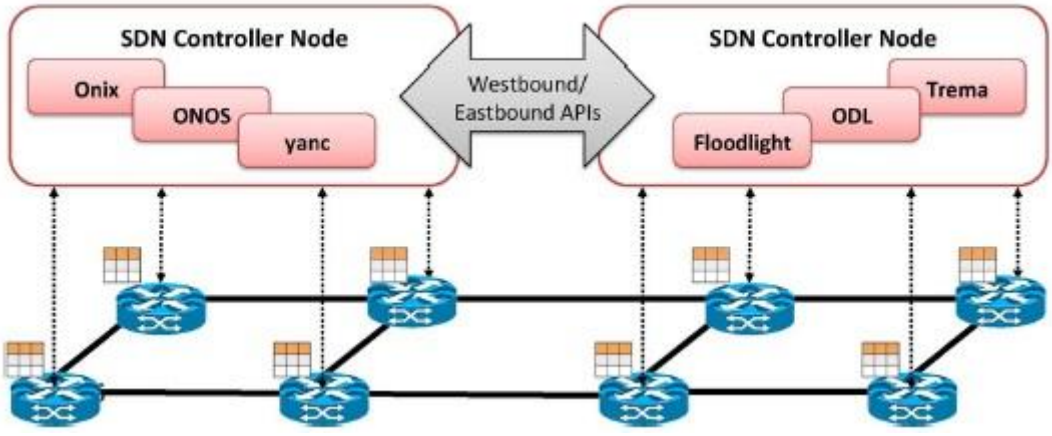
و مکانیزم‌های امنیتی، عملکردهای کنترلی ضروری شبکه می‌با شند که برنامه‌های کاربردی شبکه برای ساخت منطق خود ممکن است از آنها استفاده کنند.

۱-۲-۵-۳- واسط های جنوبی

در سطح پایین بسترهای کنترلی، واسط‌های برنامه نویسی جنوبی را می توان به عنوان درایور دستگاه‌ها دید. این لایه کنترلرها این امکان را می‌دهد که با سطوح پایین ارتباط برقرار کنند. اکثر کنترلرها فقط OpenFlow را به عنوان واسط برنامه‌نویسی جنوبی خود ارائه می‌دهند. البته تعداد کمی از آنها مانند OpenDayLight و Onix و HP VAN SDN از رنج وسیع‌تری از واسط‌های برنامه‌نویسی جنوبی پشتیبانی می‌کنند.

۱-۲-۵-۴- واسط های شرقی و غربی :

همان طور که در شکل (۱-۷) نشان داده شده است واسط‌های برنامه‌نویسی شرقی/غربی وا سط‌های ضروری برای کنترلرهای توزیع شده می‌با شند. در حال حا ضر هر کنترلر، وا سط‌های برنامه‌نوی سی شرقی/غربی خود را پیاده سازی می‌کنند. عملکردهای این وا سط ها شامل وارد و خارج کردن داده بین کنترلر ها، الگوریتم هایی برای مدل های سازگاری داده، و قابلیت های مانیتورینگ/آگاه‌سازی (به عنوان مثال بررسی کند که آیا کنترلر در حال کار می‌باشد یا نه). مشابه وا سط‌های برنامه نویسی شمالی و جنوبی، وا سط‌های برنامه‌نویسی شرقی/غربی نیز برای کنترلرهای توزیع شده ضروری می‌باشند[۱۵].



شکل (۱-۷) کنترلر های توزیع شده : واسط های برنامه سازی شرقی / غربی

۱-۲-۵-۵- واسط های برنامه نویسی شمالی :

کنترلرهای رایج طیف وسیعی از واسطهای برنامه‌نویسی شمالی از جمله واسطهای برنامه نویسی تک کاره[۱]، واسطهای برنامه‌نویسی RESTful، واسطهای برنامه‌نویسی چند سطحی[۲]، سیستم‌های فایل، در کنار دیگر واسطهای برنامه نویسی تخصصی‌تر از جمله NVP NBAPI و واسطهای برنامه نویسیSDMN را ارائه می‌دهند. نوع دیگری از واسطهای شمالی در واقع زبان‌های برنامه نویسی شبکه‌های نرم افزار محور مانند Frenetic، Procera، NetCore، Nettle و Pyretic و دیگر زبان‌های مبتنی بر پرس‌وجو می‌باشند. در بخش‌های بعد زبان‌های برنامه‌نویسی شبکه‌های نرم افزار محور بررسی می‌شوند.

۱-۲-۶- لایه ۵ : واسط های شمالی

واسطهای شمالی و جنوبی دو سطح تجرید اصلی اکوسیستم شبکه‌های نرم افزار محور می‌باشند. برای واسطهای جنوبی پروتکل OpenFlow به طور گسترده مقبولیت یافته است اما یک واسط شمالی مشترک همچنان یک موضوع داغ برای تحقیق می‌باشد. در این لحظه که این مقاله نوشته می‌شود، هنوز بسیار زود می‌باشد که یک واسط شمالی استاندارد تعریف شود زیرا حوزه‌های کاربرد آن همچنان در حال توسعه می‌باشد. اما به هر حال این انتظار می‌رود که یک واسط شمالی مشترک با پیشرفت شبکه‌های نرم افزار محور به وجود آید. برای اینکه پتانسیل کامل شبکه‌های نرم افزار محور کشف شود، تجریدی که به برنامه‌های کاربردی شبکه اجازه دهد که وابسته به پیاده سازی‌های به خصوصی نباشد، لازم است.

به منظور ارائه قابلیت حمل[۳] و قابلیت همکاری[۴] برنامه‌های کاربردی بین بسترهای کنترلی مختلف، واسطهای شمالی استاندارد و باز[۵] ضروری می‌باشند.

---

[۱] Ad hoc APIs

[۲] Multilevel Programming Interface

[۳] Portability

[۴] interoperability

[۵] Open

۱-۲-۷- لایه ۶ : مجازی سازی مبتنی بر زبان برنامه نویسی

زبان‌های برنامه‌نویسی که در بخش بعدی مورد بررسی قرار خواهند گرفت، هرکدام می‌توانند یک سطح انتزاع مناسب را برای توسعه‌دهندگان و برنامه‌نویسان فراهم کنند. برای مثال می‌توان به برنامه‌نویسان به جای ارائه مجموعه‌ای از چندین سوییچ و المان شبکه، یک "سوییچ مجازی بزرگ" ارائه شود. این نوع از انتزاع و مجازی‌سازی سبب آسان‌تر شدن کار برنامه‌نویسان می‌شود که دیگر نیازی به فکر کردن در مورد المان‌های توزیع‌شده و اینکه سیاست‌ها چگونه به آن‌ها اعمال شود، ندارند.

Pyretic به عنوان یک زبان برنامه نویسی پرکاربرد در شبکه‌های نرم افزار محور شناخته می‌شود. این زبان با ارائه یک فرم انتزاعی و مجازی‌سازی سطح‌بالا، توپولوژی شبکه را از دید برنامه نویس پنهان می‌کند.

۱-۲-۸- لایه ۷ : زبان های برنامه نویسی

زبان‌های برنامه‌نویسی زیادی در دهه‌های گذشته معرفی شده‌اند. هم زبان‌های آکادمیک و هم زبان‌های صنعتی هر دو از زبان‌های سطح پایین مخصوص هر ماشین مانند زبان اسمبلی برای معماری‌های x۸۶ به زبان‌های سطح بالا و قدرتمند مانند جاوا و پایتون ارتقاء یافته‌اند. پیشرفت‌هایی که در زمینه قابلیت استفاده مجدد از کد و قابلیت حمل انجام شده، صنعت کامپیوتر را رو به جلو حرکت داده است.

به طور مشابه قابلیت برنامه نویسی در شبکه‌ها نیز از زبان‌های ماشین سطح پایین مثل OpenFlow (اسمبلی) به زبان‌های برنامه نویسی سطح بالا تغییر جهت داده است. زبان‌های ماشین شبیه اسمبلی مانند OpenFlow و POF در واقع رفتار دستگاه‌های پیش‌بری را تقلید کرده و برنامه نویسان را مجبور می‌کند که بجای حل مسئله، زمان زیادی را صرف جزییات سطح پایین کنند. استفاده از این زبانهای سطح پایین، استفاده مجدد از نرم افزار و ساخت کدهای وسیع و ماژولار را سخت کرده و همچنین روند توسعه نرم افزار را مستعد خطا می‌کند.

تجریدهایی که توسط زبانهای برنامه نویسی سطح بالا فراهم می‌شود به طور قابل توجه‌ای می‌تواند به مقابله با چالش‌های این دستورالعمل‌های سطح پایین، کمک کند.

چالش‌های متعددی با استفاده از زبان‌های برنامه‌نویسی سطح بالا بهتر اداره می‌شوند. به عنوان مثال در شبکه‌های نرم‌افزار محور مبتنی بر OpenFlow محض، بسیار سخت می‌باشد تضمین شود که وظایف

متعدد یک برنامه کاربردی واحد، با یکدیگر تداخل پیدا نمی‌کنند. به عنوان مثال قوانین[1] تولید شده برای یک وظیفه[2] نباید عملکردهای وظیفه دیگر را باطل کند.

تکنیک‌های طراحی نرم‌افزار مهم مانند ماژولاریتی کد و قابلیت استفاده مجدد از کد با استفاده از مدل‌های برنامه‌نویسی سطح پایین بسیار سخت به دست می‌آیند. بنابراین برنامه‌های کاربردی ساخته شده، یکپارچه بوده و از اجزا تشکیل‌دهنده آنها نمی‌توان در دیگر برنامه‌های کاربردی استفاده کرد. نتیجه، یک روند توسعه مستعد خطا و بسیار زمان بر می‌باشد.

ویژگی جالب دیگری که تجریدهای زبان‌های برنامه‌نوسی فراهم می‌کنند، قابلیت ساخت برنامه‌هایی برای توپولوژی شبکه مجازی می‌باشد. این مفهوم مشابه برنامه‌نویسی شی‌گرا می‌باشد که اشیاء داده و عملکردها را از برنامه‌نویسان برنامه کاربردی تجرید کرده و در این صورت کار برنامه‌نویسان را ساده می‌کند. زیرا دیگر نیازی نیست آنها بجای تمرکز روی حل مشکل، نگران ساختمان‌داده و مدیریت آنها باشند. به عنوان مثال، در زمینه شبکه‌های نرم‌افزار محور، بجای تولید و نصب قوانین در هر دستگاه پیش‌رو، می‌توان بدین صورت فکر شود که یک توپولوژی شبکه مجازی ساخته شود که نمایش دهنده کل شبکه و یا زیر بخشی از آن است. به طور مثال برنامه نویس برنامه کاربردی باید بتواند شبکه را به صورت یک سوییچ بزرگ ببیند، بجای اینکه آن را ترکیبی از دستگاه‌های سخت افزاری زیرین ببیند. زبان برنامه نویسی یا سیستم زمان اجرا این وظیفه را دارد که دستورالعملهای سطح پایین مورد نیاز را در هر دستگاه پیش‌رو، تولید و نصب کند تا سیاست کاربر را در سطح شبکه به اجرا درآورد.

با همچین سطح تجریدی، توسعه یک برنامه کاربردی مسیریابی روند بسیار ساده‌ای دارد. به طور مشابه یک سوییچ واحد می‌تواند به صورت مجموعه‌ای از سوییچ‌ها نمایش داده شود که هر کدام از آنها متعلق به یک شبکه مجازی متفاوتی است.

این دو مثال از تجرید توپولوژی شبکه را اگر بخواهیم با استفاده از دستور العملهای سطح پایین پیاده سازی کنیم، بسیار سخت می‌باشد. اما زبان‌های برنامه نویسی مانند Pyretic به راحتی می‌توانند این سطح از تجریدها را برای توپولوژی‌های شبکه مجازی فراهم کنند.

جنبه مهم دیگر، قابلیت حمل زبان‌های برنامه‌نویسی می‌باشد که بسیار مهم است. زیرا در این صورت دیگر برنامه‌نویسان مجبور نخواهند بود که برنامه‌های کاربردی را برای بسترهای کنترلی متفاوت دوباره

[1] Rule

[2] Task

نویسی‌کنند. مشابه ماشین مجازی جاوا، یک واسط شمالی قابل حمل این اجازه را به برنامه کاربردی می‌دهد که بدون هیچ تغییری روی کنترلرهای متفاوت اجرا شود. زبان Pyretic این قابلیت را فراهم می‌کند. کنترلرهای FloodLight، OpenDayLight، Onix، ONOS از زبان برنامه نویسی جاوا پشتیبانی کرده و کنترلرهای POX و RYU نیز از زبان برنامه نویسی پایتون پشتیبانی می کنند.

۱-۲-۹- لایه ۸ : برنامه های کاربردی شبکه

برنامه‌های کاربردی شبکه را می توان مغز شبکه نامید، زیرا آنها منطق کنترلی را پیاده سازی می‌کنند که بعداً ترجمه به یک سری دستورات شده که قرار است در سطح داده نصب شوند. به عبارتی به دستگاه های پیش‌بری رفتار مورد انتظار را دیکته می‌کند. به عنوان مثال، یک برنامه کاربردی ساده مثل مسیریابی را در نظر بگیرید. منطق این برنامه کاربردی بایستی مسیری که بسته‌های تولید شده در گره A طی می‌کنند تا به گره B برسند را تعریف کند. برای رسیدن به این هدف ، برنامه کاربردی مسیریابی بایستی توپولوژی شبکه را به عنوان ورودی گرفته و سپس مسیر مورد نظر را در تمام دستگاه‌های پیش‌بری قرار گرفته در مسیر از A به B، نصب کند.

شبکه‌های نرم افزار محور را می توان در تمام محیط‌های شبکه‌های سنتی، از شبکه‌های خانگی و سازمانی تا شبکه‌های مراکز داده[۱] و حامل داده[۲] نصب کرد. این گوناگونی محیط باعث به وجود آمدن آرایه وسیعی از برنامه‌های کاربردی شده است.

برنامه‌های کاربردی جاری عملکردهای سنتی از قبیل مسیریابی، متوازن سازی بار، اجرای سیاست های امنیتی را انجام می دهند. عملکردهای نوینی که برنامه‌های کاربردی به سوی پیاده‌سازی آنها می روند شامل کاهش مصرف انرژی، اجرای کیفیت سرویس[۳] انتها به انتها، مجازی سازی شبکه، مدیریت تحرکات در شبکه‌های بیسیم و غیره می باشند.

علی رغم کاربردهای فراوان، بیشتر برنامه‌های کاربردی شبکه‌های نرم افزار محور، در یکی از این پنج

[۱] Data Center

[۲] Carrier Networks

[۳] Quality of Service

دسته قرار می‌گیرند: مهندسی ترافیک، تحرکات[۱] و بیسیم، اندازه‌گیری و مانیتورینگ، امنیت[۲] و قابلیت اتکا[۳]، و شبکه‌های مراکز داده.

[۱] Mobility

[۲] Security

[۳] Dependability

مراجع

[١] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, ‘‘A survey of software-defined networking: Past, present, future of programmable networks,’’ IEEE Commun. Surv. Tut., vol. 16, no. 3, pp. 1617–1634, Third Quart. 2014.

[٢] A. Lara, A. Kolasani, and B. Ramamurthy, ‘‘Network innovation using OpenFlow: A survey,’’ IEEE Commun. Surv. Tut., vol. 16, no. 1, pp. 493–512, First Quart. 2014.

[٣] N. Feamster, J. Rexford, and E. Zegura, ‘‘The road to SDN,’’ Queue, vol. 11, no. 12, pp. 20:20–20:40, Dec. 2013.

[٤] A. Doria et al., ‘‘Forwarding and control element separation (ForCES) protocol specification,’’ Internet Engineering Task Force, Mar. 2010. Online]. Available: http://www.ietf.org/rfc/rfc5810.txt

[٥] H. Song, ‘‘Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane,’’ in Proc. 2nd ACM IGCOMM Workshop Hot Topics Softw. Defined Netw., 2013, pp. 127–132.

[٦] J. Pan, S. Paul, and R. Jain, ‘‘A survey of the research on future internet architectures,’’ IEEE Commun. Mag., vol. 49, no. 7, pp. 26–36,Jul. 2011.

[٧] B. Pfaff and B. Davie, ‘‘The Open vSwitch database management protocol,’’ Internet Engineering Task Force, RFC 7047 (Informational), Dec. 2013. [Online]. Available**:** http://www.ietf.org/rfc/ rfc7047.txt

[٨] M. Smith et al., ‘‘OpFlex control protocol,’’ Internet Engineering Task Force,’’ Internet Draft, Apr. 2014. [Online]. Available: http:// tools.ietf.org/html/draft-smith-opflex-00

[٩] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, ‘‘OpenState: Programming platform-independent stateful OpenFlow applications inside the switch,’’ SIGCOMM Comput. Commun. Rev., vol. 44, no. 2, pp. 44–51, Apr. 2014.

[١٠] M. Sune, V. Alvarez, T. Jungel, U. Toseef,and K. Pentikousis, ‘‘An OpenFlow implementation for network processors,’’ in Proc. 3rd Eur. Workshop Softw. Defined Netw., 2014, 2 pp.

[١١] D. Parniewicz et al., ‘‘Design and implementation of an OpenFlow hardware abstraction layer,’’ in Proc. ACM SIGCOMM Workshop Distrib. Cloud Comput., 2014, pp. 71–76.

[١٢] T. J. Bittman, G. J. Weiss, M. A. Margevicius, and P. Dawson, ‘‘Magic quadrant for x86 server virtualization infrastructure,’’ Gartner, Tech. Rep., Jun. 2013.

[١٣] A. Al-Shabibi et al., ‘‘OpenVirteX: A Network Hypervisor,’’ 2014. [Online]. Available: http://ovx.onlab.us/wp-content/uploads/ 2014/04/ovx-ons14. Pdf

[١٤] B. Belter et al., ‘‘Programmable abstraction of datapath,’’ in Proc. 3rd Eur. Workshop Softw. Defined Netw., 2014, pp. 7–12.

[١٥] Diego Kreutz, Fernando M. V. Ramos, Paulo Jorge Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, Steve Uhlig:Software-Defined Networking: A Comprehensive Survey. Proceedings of the IEEE 103(1): 14-76 (2015)

[١٦] S. Yeganeh, A. Tootoonchian, and Y. Ganjali,‘‘On scalability of software-defined networking,’’ IEEE Commun. Mag.,vol. 51, no. 2, pp. 136–141, Feb. 2013.

[۱۷] A. R. Curtis et al., ‘‘DevoFlow: Scaling flow management for high-performance networks,’’ Comput. Commun. Rev., vol. 41, no. 4, pp. 254–265, Aug. 2011.

[۱۸] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, ‘‘Scalable flow-based networking with difane,’’ SIGCOMM Comput. Commun. Rev., vol. 40, no. 4, pp. 351–362, Aug. 2010.

[۱۹] J. C. Mogul and P. Congdon, ‘‘Hey, you darned counters! Get off my asic!’’ in Proc. 1st Workshop Hot Topics Softw. Defined Netw.,2012, pp. 25–30.

[۲۰] T. Koponen et al., ‘‘Onix: A distributed control platform for large-scale production networks,’’ in Proc. 9th USENIX Conf. Oper. Syst. Design Implement., 2010, pp. 1–6.

[۲۱] A. Tootoonchian and Y. Ganjali, ‘‘HyperFlow: A distributed control plane for OpenFlow,’’ in Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw., 2010, p. 3.

[۲۲] S. Hassas Yeganeh and Y. Ganjali, ‘‘Kandoo: A framework for efficient and scalable offloading of control applications,’’ in Proc. 1st Workshop Hot Topics Softw. Defined Netw.,2012, pp. 19–24.

[۲۳] Z. Cai, A. L. Cox, and T. S. E. Ng, ‘‘Maestro:A system for scalable OpenFlow control,’’ Rice Univ., Houston, TX, USA, Tech. Rep., 2011.

[۲٤] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, ‘‘On controller performance in software-defined networks,’’ in Proc. 2nd USENIX Conf. Hot Topics Manage. Internet Cloud Enterprise Netw. Services, 2012, p. 10.

[۲۵] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, ‘‘Maple: Simplifying SDN programming using algorithmic policies,’’ in Proc. ACM SIGCOMM Conf., 2013, pp. 87–98.

[۲٦] K. Govindarajan, K. Chee Meng, “A Literature Review on Software-Defined Networking (SDN) Research Topics, Challenges and Solutions”

[۲۷] Z. Yan and C. Prehofer. \Autonomic Trust Manage-ment for a Component-Based Software System". In: IEEE Trans. on Dep. and Sec. Computing 8.6 (2011).

[۲۸] M. Georgiev et al. \The most dangerous code in the world: validating SSL certi_cates in non-browser soft-ware". In: ACM CCS. 2012.

[۲۹] Y. G. Desmedt. \Threshold cryptography". In: Euro-pean Transactions on Telecommunications 5.4 994).

[۳۰] S. Neti, A. Somayaji, and M. E. Locasto. \Software diversity: Security, Entropy and Game Theory". In:
7th USENIX HotSec. 2012.

[۳۱] M. Garcia et al. \Analysis of operating system diversity for intrusion tolerance". In: Software: Practice and Experience (2013).

[۳۲] B. Heller, R. Sherwood, and N. McKeown. \The controller placement problem". In: HotSDN. ACM, 2012.

[۳۳] J. H. Perkins et al. \Automatically patching errors in deployed software". In: ACM GOPS SOSP. 2009.