

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



دانشگاه غیر دولتی - غیر انتفاعی خاتم  
دانشکده فنی و مهندسی  
گروه علوم داده

## پیش‌بینی خطاها در شبکه‌های نرم‌افزار محور

پایان‌نامه برای دریافت درجه کارشناسی ارشد  
در رشته مهندسی کامپیوتر  
گرایش علوم داده

استاد راهنما:  
دکتر حسین حجت

دانشجو:  
محمدرضا قباخلو

شهریور ماه ۱۴۰۳



## گواهی دفاع از پایان نامه کارشناسی ارشد ناپیوسته

با عنایت به آئین نامه آموزشی دوره کارشناسی ارشد ناپیوسته، جلسه دفاع از پایان نامه کارشناسی ارشد ناپیوسته محمدرضا قباخلو دانشجوی رشته علوم داده تحت عنوان "پیش بینی خطاها در شبکه های نرم افزار محور" به راهنمایی جناب آقای دکتر حسین حجت در تاریخ ۱۴۰۳/۶/۲۹ با حضور هیأت داوران تشکیل شد و براساس کیفیت ارایه پایان نامه، توضیحات و نحوه پاسخ به سوالات، رأی نهایی به شرح ذیل اعلام گردید:

عالی	۱۹.۰ - ۲۰.۰
بسیار خوب	۱۸.۰ - ۱۸.۹۹
خوب	۱۶.۰ - ۱۷.۹۹
متوسط	۱۴.۰ - ۱۵.۹۹
مردود	۰.۰ - ۱۳.۹۹

نمره پایان نامه : ۱۶ به عدد ۱۶ به حروف شانزده تمام

مشخصات هیأت داوران	نام و نام خانوادگی	مرتبه دانشگاهی	دانشگاه / مؤسسه	امضاء
استاد راهنما	جناب آقای دکتر حسین حجت	استادیار	خاتم	
استاد داور	جناب آقای دکتر مسعود صدیقین	استادیار	خاتم	
استاد داور	جناب آقای دکتر احسان خامس پناه	استادیار	تهران	

معاون آموزشی و پژوهشی  
نام و نام خانوادگی، تاریخ و امضا

مدیر گروه آموزشی  
نام و نام خانوادگی، تاریخ و امضا

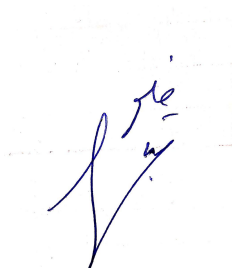
## اظهارنامه دانشجو

عنوان پایان نامه: پیش‌بینی خطاها در شبکه‌های نرم‌افزار محور

استاد راهنما: دکتر حسین حجت

اینجانب محمدرضا قباخلو دانشجوی دوره کارشناسی ارشد رشته مهندسی کامپیوتر گرایش علوم داده دانشگاه خاتم به شماره دانشجویی ۴۰۰۱۳۰۵۹۶۵۰۰۵ گواهی می‌نمایم که تحقیقات ارائه شده در این پایان نامه توسط اینجانب انجام شده است و صحت و اصالت مطالب نگارش شده مورد تأیید می‌باشد و در موارد استفاده از کار دیگر محققان به مرجع مورد استفاده اشاره شده است. به-علاوه گواهی می‌نمایم که مطالب مندرج در پایان نامه تاکنون برای دریافت هیچ نوع مدرک یا امتیازی توسط اینجانب یا فرد دیگری ارائه نشده است و در تدوین متن پایان نامه چارچوب مصوب دانشگاه را به‌طور کامل رعایت کرده‌ام.

کلیه حقوق مادی و معنوی مترتب بر نتایج مطالعات، ابتکارات و نوآوری‌های ناشی از تحقیق، همچنین چاپ و تکثیر، نسخه برداری، ترجمه و اقتباس از این پایان نامه کارشناسی ارشد، برای دانشگاه خاتم محفوظ است. نقل مطلب با ذکر منبع بلامانع است.



امضاء دانشجو:

تاریخ: ۲۳ شهریور ماه ۱۴۰۳

## تقدیم به همسر

همراهی، حمایت بی دریغ و صبوری ایشان در طول این مسیر، نیروی محرکه‌ای بود که مرا در لحظات دشوار و چالش‌های پیش رو یاری داد. بدون عشق، تشویق و دلگرمی همسر، دستیابی به این هدف ممکن نبود. همچنین از پدر و مادرم برای تمامی زحماتشان در ایجاد محیطی برای کسب دانش تشکر می‌کنم.

## سپاس

از صمیم قلب از استاد راهنمای محترم، جناب آقای دکتر حسین حجت، به خاطر راهنمایی‌ها، حمایت‌ها و کمک‌های بی‌دریغ‌شان در طول نگارش این پایان‌نامه سپاسگزارم. حضور همیشگی ایشان، نکات ارزشمند و دقیق، و همچنین صبوری و حوصله‌ای که در تمامی مراحل این پژوهش از خود نشان دادند، نقش اساسی در پیشبرد و تکمیل این کار داشت. از ایشان به خاطر وقت و انرژی‌ای که برای هدایت من در این مسیر صرف کردند، بی‌نهایت سپاسگزارم. این پژوهش بدون حمایت‌های بی‌دریغ ایشان به این مرحله از موفقیت نمی‌رسید.

## چکیده

در این پژوهش، به بررسی و تحلیل پیش‌بینی خطا در شبکه‌های نرم‌افزار محور (Software Defined Networks) با استفاده از چارچوب زبان داینتکت (DyNetKAT) پرداخته شده است. ابزار توسعه یافته با یادگیری و استخراج قوانین داینتکت از لاگ فایل‌های پروتکل اُپن‌فلو (OpenFlow) و اعمال آن‌ها بر روی توپولوژی استخراج شده، امکان بررسی رفتارها و به‌روزرسانی‌های پویا شبکه را فراهم می‌کند و می‌تواند خطاهای ناشی از این رفتارها در آینده را پیش‌بینی کند. ابزار توسعه‌یافته‌ی ما، FPSDN، به طور مؤثری از این قوانین برای اعتبارسنجی صوری رفتار شبکه استفاده کرده و بر ویژگی‌هایی همچون دسترسی‌پذیری و مدیریت پیکربندی‌های پویا تمرکز دارد. نتایج به‌دست‌آمده نشان می‌دهد که این ابزار قادر است خطاهای بالقوه را با دقت پیش‌بینی کند و از بروز قطعی‌های پرهزینه در شبکه‌های نرم‌افزارمحور جلوگیری کند. این پیشرفت در پیش‌بینی خطاها، ضمن افزایش قابلیت اطمینان شبکه، منجر به بهبود عملکرد و کاهش زمان‌های خرابی شبکه خواهد شد.

**واژه‌های کلیدی:** شبکه‌های نرم‌افزار محور، لاگ فایل، زبان داینتکت، پروتکل اُپن‌فلو، به‌روزرسانی‌های

پویا، توپولوژی، سوئیچ، دسترسی‌پذیری

## فهرست مطالب

۱	مقدمه	۱
۲	۱-۱- مقدمه	۲
۳	۱-۲- تعریف موضوع تحقیق	۳
۴	۱-۳- اهداف کلی تحقیق	۴
۵	۱-۴- نوآوری، اهمیت و ارزش تحقیق	۵
۵	۱-۴-۱- نوآوری‌ها	۵
۵	۱-۴-۲- اهمیت و ارزش تحقیق	۵
۶	۱-۵- ساختار تحقیق	۶
۷	۲- مروری بر ادبیات موضوع	۷
۸	۲-۱- مقدمه	۸
۸	۲-۲- شبکه‌های نرم‌افزار محور	۸
۹	۲-۳- لاگ شبکه	۹
۱۰	۲-۴- توپولوژی شبکه	۱۰
۱۲	۲-۵- خطا در شبکه‌های نرم‌افزار محور	۱۲
۱۴	۲-۶- زبان‌های توصیف شبکه‌های نرم‌افزار محور	۱۴
۱۵	۲-۷- کارهای پیشین	۱۵



۱۸	۳ توصیف شبکه‌های نرم‌افزار محور
۱۹	۳-۱- مقدمه
۲۰	۳-۲- جبر کلین با آزمون
۲۱	۳-۳- نتکت
۲۱	۳-۳-۱- نحو زبان نتکت
۲۳	۳-۳-۲- معنای زبان نتکت
۲۳	۳-۴- داینکت
۲۵	۳-۴-۱- نحو و معنای داینکت
۲۶	۳-۴-۲- ویژگی‌های ایمنی
۲۷	۳-۵- مثال‌ها
۲۷	۳-۵-۱- دیوار آتش حالت‌مند
۲۸	۳-۵-۲- مثال دوم: تک سوئیچ
۳۰	۴ بررسی فایل لاگ و استخراج توپولوژی
۳۱	۴-۱- مقدمه
۳۱	۴-۲- لاگ آپن‌فلو
۳۲	۴-۲-۱- پیام‌های Packet_In
۳۳	۴-۲-۲- پیام‌های Flow_Mod
۳۳	۴-۲-۳- پیام‌های Packet_Out
۳۴	۴-۳- استخراج توپولوژی

## ۵ استخراج قوانین DyNetKAT ۳۷

۳۸	۱-۵- مقدمه
۳۸	۲-۵- نمای کلی توصیف شبکه
۳۹	۳-۵- قوانین استخراج
۳۹	۱-۳-۵- قوانین پیام $\text{Packet\_In}(sw, mid, omids)$
۴۰	۲-۳-۵- قوانین پیام $\text{Flow\_Mod}(sw, mid', omids')$
۴۰	۳-۳-۵- قوانین پیام $\text{Packet\_Out}(sw, mid, mtype, ops)$
۴۱	۴-۵- مثال

## ۶ پیاده‌سازی و آزمایش‌ها ۴۳

۴۴	۱-۶- پیاده‌سازی
۴۴	۱-۱-۶- ورودی ابزار
۴۴	۲-۱-۶- مرحله پیش‌پردازش
۴۵	۳-۱-۶- مرحله یادگیری توپولوژی شبکه
۴۵	۴-۱-۶- مرحله یادگیری مدل رفتاری شبکه
۴۵	۵-۱-۶- مرحله بررسی پیکربندی‌های شبکه و ایجاد ویژگی‌های ایمنی مورد نیاز
۴۵	۶-۱-۶- مرحله بررسی ویژگی‌های ایمنی
۴۶	۷-۱-۶- خروجی ابزار
۴۶	۲-۶- آزمایش‌ها
۴۷	۱-۲-۶- آزمایش Star

۴۸	.....	Mesh	آزمایش	۲-۲-۶
۴۹	.....	Ring	آزمایش	۳-۲-۶
۴۹	.....	Black Hole	آزمایش	۴-۲-۶
۵۰	.....	Race condition	آزمایش	۵-۲-۶
۵۱	.....	Isolation	آزمایش	۶-۲-۶
۵۲	.....	Faulty Linear	آزمایش	۷-۲-۶
۵۳	.....	Faulty Fattree	آزمایش	۸-۲-۶
۵۴	.....	نصب و راه‌اندازی		۳-۶
۵۵		<b>۷ پیش‌بینی خطا و نتایج</b>		
۵۶	.....	ویژگی‌های ایمنی آزمایش‌ها		۱-۷
۵۸	.....	نتایج		۲-۷
۶۲	.....	تحلیل سوالات تحقیق		۳-۷
۶۴		<b>۸ بحث و نتیجه‌گیری</b>		
۶۵	.....	مقدمه		۱-۸
۶۵	.....	جمع‌بندی		۲-۸
۶۵	.....	محدودیت‌ها		۳-۸
۶۶	.....	پیشنهادها		۴-۸
۶۸		<b>منابع و مراجع</b>		

آ	پیوست‌ها	۷۱
آ-۱	کدها	۷۲
آ-۱-۱	توپولوژی FatTree در Mininet	۷۲

## فهرست جداول

۲۲	.....	(۱-۳) نحو زبان نتکت
۲۴	.....	(۲-۳) معنای زبان نتکت
۵۹	.....	(۱-۷) خلاصه نتایج آزمایش‌ها

## فهرست تصاویر

۹	(۱-۲) معماری شبکه‌های نرم‌افزار محور (Kreutz et al., 2015) . . . . .
۲۵	(۱-۳) نحو زبان داینکتک . . . . .
۲۷	(۲-۳) دیوار آتش حالت‌مند . . . . .
۲۸	(۳-۳) دیوار آتش حالت‌مند . . . . .
۲۹	(۴-۳) مثال با یک سوئیچ . . . . .
۳۲	(۱-۴) نمونه لاگ فایل Packet_In برای مثال (۴-۳) . . . . .
۳۳	(۲-۴) نمونه لاگ فایل Flow_Mod برای مثال (۴-۳) . . . . .
۳۳	(۳-۴) نمونه لاگ فایل Packet_Out برای مثال (۴-۳) . . . . .
۳۶	(۴-۴) الگوریتم استخراج توپولوژی از لاگ فایل . . . . .
۳۶	(۵-۴) الگوریتم تخصیص پورت . . . . .
۴۸	(۱-۶) توپولوژی آزمایش Star . . . . .
۴۹	(۲-۶) توپولوژی آزمایش Mesh . . . . .
۵۰	(۳-۶) توپولوژی آزمایش Ring . . . . .
۵۰	(۴-۶) توپولوژی خطی . . . . .
۵۱	(۵-۶) توپولوژی آزمایش Race Condition . . . . .
۵۲	(۶-۶) توپولوژی آزمایش Isolation . . . . .
۵۳	(۷-۶) توپولوژی آزمایش FatTree . . . . .

## فهرست نمودارها

## فصل ۱: مقدمه

---



## ۱-۱- مقدمه

در عصر حاضر، شبکه‌های کامپیوتری به یکی از اصلی‌ترین زیرساخت‌های فناوری اطلاعات تبدیل شده‌اند و مدیریت بهینه و موثر آن‌ها از اهمیت بالایی برخوردار است. شبکه‌های سنتی معمولاً به تجهیزات سخت‌افزاری خاصی وابسته بودند و مدیریت آن‌ها نیازمند پیکربندی<sup>۱</sup> دستی و زمان‌بر بود. این روش دستی باعث می‌شد تغییرات و به‌روزرسانی‌های شبکه با پیچیدگی‌های زیادی همراه شود. اما با ظهور شبکه‌های نرم‌افزارمحور<sup>۲</sup>، رویکرد جدیدی در مدیریت و پیکربندی شبکه‌ها معرفی شد.

شبکه‌های نرم‌افزارمحور به گونه‌ای طراحی شده‌اند که لایه کنترل<sup>۳</sup> شبکه از لایه داده<sup>۴</sup> جدا می‌باشد. این جداسازی باعث می‌شود که مدیریت ترافیک شبکه به صورت متمرکز و از طریق یک کنترل‌کننده<sup>۵</sup> مرکزی انجام شود. کنترل‌کننده شبکه می‌تواند رفتار سوئیچ‌ها<sup>۶</sup> و سایر تجهیزات شبکه را به صورت مستقیم تنظیم کند و این تنظیمات به سرعت و بدون نیاز به تغییرات فیزیکی در شبکه اعمال می‌شود. این قابلیت انعطاف‌پذیری بالایی به شبکه‌ها می‌دهد و امکان بهینه‌سازی و تغییرات سریع را فراهم می‌کند (Casado et al., 2014; Sloane., 2013).

یکی از مهم‌ترین مزایای شبکه‌های نرم‌افزارمحور، توانایی در کاهش پیچیدگی‌های مدیریتی و افزایش سرعت در اجرای تغییرات است. در شبکه‌های سنتی، هر دستگاه شبکه باید به صورت جداگانه پیکربندی می‌شد، اما در شبکه‌های نرم‌افزارمحور، کنترل‌کننده مرکزی می‌تواند به طور همزمان بر روی همه دستگاه‌ها اعمال کنترل کند. این امر باعث می‌شود که شبکه‌ها بهتر با تغییرات محیطی یا نیازهای جدید کاربران هماهنگ شوند.

با این حال، با وجود تمام این مزایا، شبکه‌های نرم‌افزارمحور با چالش‌هایی نیز مواجه هستند. مدیریت تغییرات پویای شبکه و اطمینان از عملکرد صحیح در حین به‌روزرسانی‌ها یکی از مهم‌ترین چالش‌هاست. تغییرات ناگهانی و ناهماهنگی در تنظیمات شبکه ممکن است منجر به بروز خطاهای جدی شود که عملکرد شبکه را مختل کند و حتی باعث قطعی کامل شبکه شود. به همین دلیل، نیاز به روش‌ها و ابزارهایی برای پیش‌بینی و شناسایی خطاهای احتمالی پیش از وقوع آنها بیشتر از گذشته احساس می‌شود (Elsayed et al., 2020).

<sup>۱</sup> Configuration

<sup>۲</sup> Software Defined Networks(SDNs)

<sup>۳</sup> Control Plane

<sup>۴</sup> Data Plane

<sup>۵</sup> Controllor

<sup>۶</sup> Switches

## ۲-۱- تعریف موضوع تحقیق

همانطور که گفته شد، شبکه‌های نرم‌افزارمحور به دلیل انعطاف‌پذیری و قابلیت‌های پیشرفته‌ای که در مدیریت و پیکربندی شبکه ارائه می‌دهند، به سرعت به زیرساخت اصلی بسیاری از سازمان‌ها و شرکت‌ها تبدیل شده‌اند. با این حال، تغییرات پویای مکرر در پیکربندی این شبکه‌ها، از جمله به‌روزرسانی جریان‌های شبکه، ممکن است زمینه‌ساز بروز خطاهایی شوند که عملکرد شبکه را در آینده مختل کنند. این خطاها می‌توانند شامل دسترسی‌های ناخواسته، تداخل در جریان داده‌ها، و یا کاهش کارایی کلی شبکه باشند که عموماً ناشی از عدم هماهنگی در زمان اعمال تغییرات یا اثرات جانبی ناشناخته آن‌ها هستند.

یکی از مفروضات اصلی در این پژوهش، وضعیت اولیه "امن" شبکه است. در این حالت، شبکه با مجموعه‌ای از قوانین و پیکربندی‌ها طراحی و راه‌اندازی شده است که تضمین‌کننده عملکرد صحیح و ایمن آن هستند. با گذشت زمان و اعمال چندین به‌روزرسانی در پیکربندی سوئیچ‌ها و کنترل‌کننده‌ها، شبکه باید همچنان در وضعیت "امن" باقی بماند. هدف این پژوهش تحلیل و یادگیری رفتار شبکه بین این دو وضعیت امن است. یکی از چالش‌های اصلی در شبکه‌های نرم‌افزارمحور این است که تغییرات پیکربندی، حتی در شرایطی که شبکه در وضعیت "امن" شروع به کار کرده و پس از تغییرات نیز همچنان در ظاهر امن باقی می‌ماند، ممکن است در صورت اعمال به ترتیب خاصی یا تحت شرایط مشخصی، به خطاهایی در آینده منجر شود. این خطاها می‌توانند به صورت نامحسوس و تدریجی باعث ایجاد مسیرهای غیرمنتظره یا دسترسی‌های ناخواسته شوند که ویژگی‌های امنیتی شبکه را نقض کنند. برای مثال، فرض کنید در یک شبکه ابتدا سیاستی اعمال شود که ترافیک بین دو بخش از شبکه محدود شود (مثلاً جلوگیری از دسترسی یک میزبان خاص به یک سرور حساس). سپس، تغییرات دیگری در پیکربندی سوئیچ‌ها انجام گیرد که به طور غیرمستقیم باعث ایجاد مسیری جدید برای ترافیک شود که این محدودیت اولیه را دور بزند. این وضعیت می‌تواند یک نقض<sup>۱</sup> در یکی از ویژگی‌های امنیتی شبکه، مانند جلوگیری از دسترسی غیرمجاز، ایجاد کند.

پیش‌بینی خطاهای مرتبط با تغییرات پویای پیکربندی اجزای شبکه، نقشی کلیدی در جلوگیری از ایجاد اختلالات و حفظ پایداری شبکه دارد. به صورت کلی در این پژوهش، یادگیری رفتار شبکه از فایل لاگ‌های شبکه و تحلیل دقیق تغییرات و تعاملات دینامیک در اجزای شبکه، شامل سوئیچ‌ها و کنترل‌کننده‌ها، با هدف ارائه ابزاری برای شناسایی پیشگیرانه خطاها صورت می‌گیرد. برای این منظور، داده‌های فایل لاگ شبکه<sup>۲</sup> که حاوی اطلاعاتی درباره رفتار شبکه و کنترل‌کننده است، جمع‌آوری می‌شود. با تحلیل این داده‌ها، ابزار توسعه‌یافته قادر به یادگیری رفتار شبکه، شبیه‌سازی جریان داده‌ها، و مدل‌سازی تعاملات بین سوئیچ‌ها و کنترل‌کننده خواهد بود.

در این پژوهش، یادگیری رفتار شبکه از طریق تحلیل دقیق پیام‌های ثبت‌شده در لاگ فایل‌ها، انجام می‌شود. این

<sup>۱</sup> Violation

<sup>۲</sup> Network Log

پیام‌ها اطلاعات کلیدی درباره نحوه مدیریت جریان‌ها، تغییرات پیکربندی و تعاملات بین اجزای شبکه فراهم می‌کنند. ابزار توسعه‌یافته باید قادر باشد توپولوژی شبکه را از این داده‌ها استخراج کرده و تغییرات پویا را به‌درستی مدل‌سازی کند. همچنین در نهایت با استفاده از مدل یادگرفته شده، ابزار باید توانایی شناسایی خطاهای ناشی از تغییرات در پیکربندی سوئیچ‌ها و حتی خطاهای امنیتی ناشی از دسترسی‌های غیرمجاز را داشته باشد.

در نتیجه تمرکز اصلی این تحقیق بر روی توسعه ابزاری است که از داده‌های ثبت‌شده در لاگ‌های شبکه، برای استخراج قوانین و رفتارهای شبکه استفاده کند. هدف نهایی، ایجاد یک مکانیزم برای پیش‌بینی خطاها و افزایش قابلیت اطمینان در شبکه‌های نرم‌افزارمحور است. به این ترتیب، این پژوهش تلاش می‌کند نه تنها رفتار شبکه را بین دو وضعیت امن یاد بگیرد، بلکه شرایطی را که ممکن است وضعیت شبکه را در آینده تهدید کند نیز پیش‌بینی کرده و از بروز خطاهای امنیتی جلوگیری کند.

### ۳-۱ - اهداف کلی تحقیق

به صورت کلی، هدف از این تحقیق، توسعه ابزاری است که بتواند از طریق تحلیل لاگ<sup>۱</sup>های شبکه، رفتار شبکه‌های نرم‌افزارمحور را بررسی کرده و خطاهای احتمالی ناشی از پیکربندی‌ها را در آینده پیش‌بینی کند. با این روش، امکان شناسایی زود هنگام خطاها و جلوگیری از بروز مشکلات بزرگتر در شبکه فراهم می‌شود.

این تحقیق به دنبال پاسخ برای سوالات زیر در زمینه پیش‌بینی و شناسایی خطاهای احتمالی در شبکه‌های نرم‌افزارمحور است:

- آیا می‌توان ابزاری برای یادگیری رفتار شبکه‌های نرم‌افزار محور از داده‌های لاگ شبکه توسعه داد؟
- آیا ابزار می‌تواند در شبیه‌سازی سناریوهای واقعی، مانند توپولوژی‌های پیچیده عملکرد مناسبی داشته باشد و نتایج دقیقی ارائه دهد؟ یا وابستگی‌ای به توپولوژی خاصی از شبکه وجود دارد؟
- آیا ابزار توسعه‌یافته در چارچوب این پژوهش می‌تواند در پیش‌بینی خطاهای شبکه‌های نرم‌افزارمحور موفق باشد؟
- چه نوع خطاهایی را می‌توان با استفاده از این ابزار پیش‌بینی کرد؟
- مقیاس پذیری ابزار چگونه است؟ حداکثر چه تعداد به‌روزرسانی‌های همزمان در پیکربندی سوئیچ‌ها را می‌توان بررسی کرد؟

<sup>۱</sup> Log

## ۴-۱- نوآوری، اهمیت و ارزش تحقیق

### ۴-۱-۱ نوآوری‌ها

در این تحقیق، نوآوری اصلی به استفاده از چارچوب دابنتکت برای بررسی رفتارهای پویا و پیش‌بینی خطاها در شبکه‌های نرم‌افزارمحور متمرکز شده است. برخلاف رویکردهای سنتی که معمولاً بر پایه تست‌های پس‌ازواقع قرار دارند، این روش امکان پیش‌بینی زود هنگام خطاها و جلوگیری از بروز مشکلات را فراهم می‌کند. استفاده از دابنتکت به عنوان یک زبان رسمی<sup>۱</sup> که با تحلیل دقیق و استدلال صوری ترکیب شده، نوآوری مهمی در این تحقیق محسوب می‌شود که تحلیل دقیق شبکه‌ها را ممکن می‌سازد.

علاوه بر این، نوآوری دیگری که در این تحقیق مشاهده می‌شود، توسعه ابزار FPSDN است که از لاگ‌های واقعی شبکه برای استخراج و تحلیل داده‌های شبکه استفاده می‌کند. برخلاف بسیاری از روش‌های دیگر که به نوع خاصی از خطا، شبیه‌سازی‌های محدود و دسترسی مستقیم به شبکه متکی هستند (El-Hassany et al., 2016; Menaceur et al., 2023)، ابزار ما با یادگیری توپولوژی و رفتار شبکه، توانایی تحلیل رفتارهای پویای شبکه و شناسایی خطاهای احتمالی را افزایش می‌دهد. این ابزار به کاربران شبکه امکان می‌دهد تا بدون نیاز به ایجاد دسترسی به شبکه و یا تغییری در وضعیت آن، پیش‌بینی‌های دقیق‌تری از وضعیت شبکه خود داشته باشند. نکته دیگری که به نوآوری این پژوهش اضافه می‌کند، تمرکز آن بر شبکه‌های نرم‌افزارمحور با ساختارهای پیچیده است. این توپولوژی‌ها به دلیل پیچیدگی و استفاده گسترده در مراکز داده، نیازمند روش‌های پیشرفته برای مدیریت خطاها هستند. روش‌های ارائه‌شده در این تحقیق برای بررسی توپولوژی‌های پیچیده و پویای شبکه‌ها بهینه شده‌اند و به کاربران امکان می‌دهند تا به‌صورت بهینه و کارآمد رفتارهای شبکه خود را تحلیل و خطاهای احتمالی را پیش‌بینی کنند.

### ۴-۱-۲ اهمیت و ارزش تحقیق

اهمیت این تحقیق در درجه اول به دلیل افزایش استفاده از شبکه‌های نرم‌افزارمحور در سازمان‌های بزرگ و مراکز داده است. شبکه‌های نرم‌افزارمحور به دلیل انعطاف‌پذیری و قابلیت‌های مدیریت متمرکز، توانسته‌اند تحول بزرگی در مدیریت شبکه‌ها ایجاد کنند. اما این شبکه‌ها با وجود تمام مزایای خود، همچنان با چالش‌هایی از جمله مدیریت تغییرات پویا و جلوگیری از بروز خطاها مواجه هستند. ارزش این تحقیق در ارائه راهکارهایی برای شناسایی و مدیریت این چالش‌ها نهفته است.

<sup>۱</sup> Formal Language

این پژوهش با ارائه یک ابزار عملی و قابل استفاده در دنیای واقعی، به طور مستقیم بر روی افزایش قابلیت اطمینان شبکه‌های نرم‌افزار محور اثر می‌گذارد. این ابزار به کاربران شبکه‌ها امکان می‌دهد تا به صورت پیشگیرانه خطاهای احتمالی را شناسایی کرده و از بروز مشکلات جدی جلوگیری کنند. این قابلیت برای شبکه‌هایی که نیازمند پایداری بالا هستند، از اهمیت ویژه‌ای برخوردار است.

از منظر علمی، این تحقیق با گسترش دانش موجود در زمینه استفاده از چارچوب‌های رسمی برای تحلیل شبکه‌ها، ارزشی دوچندان دارد. چارچوب‌های رسمی مانند داینکتک به دلیل امکان تحلیل دقیق و صوری، ابزارهای قدرتمندی برای بررسی و مدیریت رفتار شبکه‌ها هستند. نتایج این تحقیق می‌تواند به عنوان مبنای پژوهش‌های آینده در زمینه پیش‌بینی خطاها و مدیریت پویای شبکه‌ها قرار گیرد و بهبودهای چشم‌گیری در عملکرد و پایداری شبکه‌ها ایجاد کند.

## ۱-۵- ساختار تحقیق

این پژوهش شامل هشت فصل است که در هر فصل موضوعات مختلفی در رابطه با شبکه‌های نرم‌افزار محور و ابزارهای پیش‌بینی خطا مورد بررسی قرار گرفته است. در فصل اول، به معرفی کلی تحقیق، تعریف موضوع و اهداف تحقیق پرداخته شده است. سپس در فصل دوم، مروری بر ادبیات تحقیق انجام می‌شود که شامل شبکه‌های نرم‌افزار محور، لاگ فایل‌ها، توپولوژی شبکه و روش‌های مدیریت خطا در این شبکه‌ها است.

در فصل سوم، به توصیف شبکه‌های نرم‌افزار محور پرداخته شده و دو زبان رسمی نتکت و داینکتک به تفصیل مورد بررسی قرار می‌گیرند. در فصل چهارم، نحوه پردازش فایل‌های لاگ و استخراج توپولوژی از آن‌ها شرح داده شده است. سپس در فصل پنجم، به روش‌های استخراج قوانین داینکتک از لاگ فایل پرداخته می‌شود و مثال‌هایی عملی ارائه شده است. لازم به ذکر است که این فصل جزئی از نتایج تحقیق بنده نمی‌باشد و توسط آقای دکتر حسین حجت و خانم دکتر جورجیانا کلتیس نوشته شده است. در فصل ششم، پیاده‌سازی ابزار پیش‌بینی خطا و نتایج به دست آمده از آزمایش‌های انجام شده بر روی توپولوژی‌های مختلف شبکه توضیح داده شده است. در فصل هفتم، سناریوهای خطا در شبکه‌های نرم‌افزار محور و نتایج حاصل از تحلیل آن‌ها بررسی شده‌اند. نهایتاً، فصل هشتم شامل بحث و نتیجه‌گیری، بیان محدودیت‌ها و پیشنهادهایی برای تحقیقات آینده است.

## فصل ۲: مروری بر ادبیات موضوع

---

## ۲-۱- مقدمه

در حال حاضر عمده تحقیقات و پژوهش‌ها بر ریشه‌یابی خطاها در شبکه‌های نرم‌افزار محور بعد از وقوع خطا است (Gonzalez et al., 2017; Yinbo et al., 2018) و پژوهش‌های اندکی (Serban and Bota, 2020) در زمینه پیش‌بینی و شناسایی خطاهای احتمالی در شبکه‌های نرم‌افزار محور انجام شده است. این تحقیقات به دنبال ارائه راهکارهایی برای بهبود پایداری و کارایی شبکه‌ها و جلوگیری از بروز خطاهای ناگهانی هستند. در این فصل، به بررسی مبانی نظری و پژوهش‌های پیشین در زمینه شبکه‌های نرم‌افزار محور، تحلیل لاگ‌های شبکه، توپولوژی شبکه و مدیریت خطاها می‌پردازیم.

این فصل شامل بخش‌های مختلفی است که هر کدام به بررسی یکی از موضوعات کلیدی مرتبط با شبکه‌های نرم‌افزار محور پرداخته‌اند. ابتدا به معرفی شبکه‌های نرم‌افزار محور و مفهوم آن می‌پردازیم. سپس لاگ‌های شبکه و نحوه استفاده از آن‌ها در تحلیل شبکه‌ها را بررسی خواهیم کرد. در ادامه، توپولوژی شبکه و اهمیت آن در عملکرد شبکه‌ها تحلیل خواهد شد. همچنین، به موضوع خطاها در شبکه‌های نرم‌افزار محور و روش‌های پیش‌بینی و مدیریت آن‌ها پرداخته و در نهایت، زبان‌های رسمی مورد استفاده برای توصیف این شبکه‌ها و مروری بر پژوهش‌های پیشین خواهیم داشت.

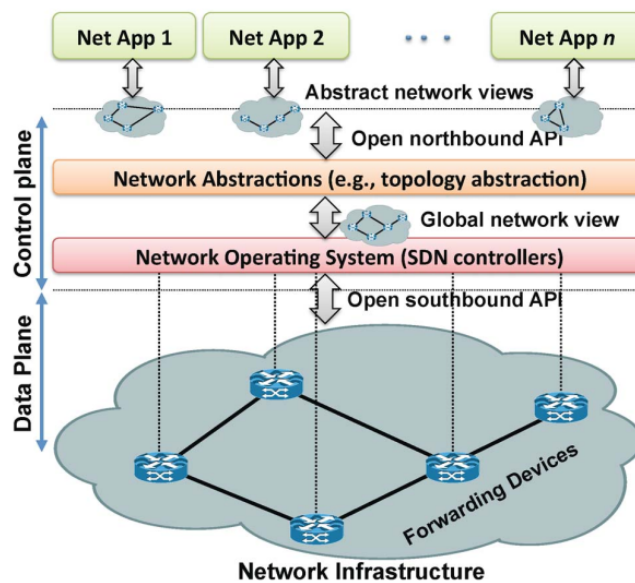
## ۲-۲- شبکه‌های نرم‌افزار محور

شبکه‌های سنتی بر پایه معماری‌هایی بنا شده‌اند که لایه کنترل و لایه داده در آن‌ها به صورت یکپارچه عمل می‌کنند. این بدان معناست که تصمیم‌گیری در مورد چگونگی هدایت بسته‌های داده و فرآیند هدایت این بسته‌ها هر دو در داخل تجهیزات شبکه (مانند سوئیچ‌ها و روترها) انجام می‌شوند. لایه کنترل مسئول تصمیم‌گیری در مورد مسیر حرکت بسته‌های داده است، در حالی که لایه داده وظیفه هدایت فیزیکی بسته‌های داده از یک نقطه به نقطه دیگر را بر عهده دارد. این یکپارچگی باعث پیچیدگی در مدیریت شبکه و کاهش انعطاف‌پذیری آن می‌شود، زیرا هر گونه تغییر یا به‌روزرسانی در سیاست‌های شبکه نیازمند تنظیمات پیچیده و دستی بر روی تجهیزات مختلف است (Kreutz et al., 2015).

با گذشت زمان و افزایش نیاز به مدیریت پویا و انعطاف‌پذیر شبکه‌ها، شبکه‌های سنتی به دلیل محدودیت‌های خود پاسخگوی نیازهای پیچیده و تغییرات سریع در محیط‌های ارتباطی نبودند. این موضوع منجر به ظهور شبکه‌های نرم‌افزار محور<sup>۱</sup> شد. در شبکه‌های نرم‌افزار محور، لایه کنترل از لایه داده جدا می‌شود و کنترل شبکه به یک کنترل‌کننده متمرکز واگذار می‌گردد. این جداسازی به شبکه‌ها اجازه می‌دهد که به صورت متمرکز و با استفاده از نرم‌افزار، تصمیم‌گیری‌های شبکه‌ای را مدیریت کنند و جریان‌های داده را به صورت پویا هدایت نمایند.

<sup>۱</sup> Software Defined Networks

در شبکه‌های نرم‌افزارمحور، همان‌طور که در شکل (۱-۲) پیوست شده نشان داده شده است، لایه کنترل از طریق ای‌پی‌ای‌های جنوب‌گرا<sup>۱</sup> با تجهیزات لایه داده (مانند سوئیچ‌ها) ارتباط برقرار می‌کند و جریان‌های داده را مدیریت می‌کند. همچنین، ای‌پی‌ای‌های شمال‌گرا<sup>۲</sup> به برنامه‌های کاربردی شبکه امکان می‌دهند که با کنترل‌کننده شبکه ارتباط برقرار کرده و سیاست‌های خود را اعمال کنند. این ساختار سه‌لایه شامل لایه داده شبکه (شامل تجهیزات زیرساخت)، لایه کنترل، و برنامه‌های کاربردی شبکه می‌باشد که به یکپارچه‌سازی و مدیریت ساده‌تر شبکه کمک می‌کند (Sloane, 2013).



شکل (۱-۲): معماری شبکه‌های نرم‌افزار محور (Kreutz et al., 2015)

این معماری شبکه‌های نرم‌افزارمحور مزایایی مانند افزایش انعطاف‌پذیری، ساده‌سازی مدیریت شبکه، و امکان به‌روزرسانی‌های سریع و پویا را به همراه دارد. در این شبکه‌ها، مدیران شبکه می‌توانند به سرعت سیاست‌های شبکه را اعمال کرده و جریان‌های داده را با توجه به نیازهای موجود تغییر دهند، بدون آنکه نیاز به تنظیمات دستی بر روی هر یک از تجهیزات شبکه داشته باشند (Casado et al., 2014; Elsayed et al., 2020).

## ۲-۳- لاگ شبکه

لاگ فایل‌ها در شبکه‌های نرم‌افزارمحور به عنوان یکی از منابع کلیدی برای نظارت و تحلیل رفتار شبکه به کار می‌روند. این فایل‌ها شامل اطلاعات دقیقی از فعالیت‌های شبکه، تغییرات در جریان‌های داده، پیام‌های کنترلی، و ارتباطات بین دستگاه‌های مختلف هستند. لاگ‌ها به مدیران شبکه امکان می‌دهند تا عملکرد شبکه را بررسی کرده و از صحت

<sup>۱</sup> Open southbound API

<sup>۲</sup> Open northbound API



عملکرد آن اطمینان حاصل کنند. استفاده از لاگ فایل‌ها به ویژه در شناسایی و پیش‌بینی خطاها اهمیت دارد. در شبکه‌های نرم‌افزارمحور، لاگ‌ها به عنوان یک ابزار مهم برای تحلیل رفتارهای ناهنجار و شناسایی مشکلات پنهان در شبکه عمل می‌کنند. با تحلیل دقیق لاگ‌های شبکه، می‌توان تغییرات ناخواسته یا ناهماهنگی‌های ناشی از خطاهای پیکربندی یا باگ‌های نرم‌افزاری را شناسایی کرد. به این ترتیب، لاگ فایل‌ها به ابزارهای نظارت شبکه تبدیل شده‌اند.

در نهایت، یکی از چالش‌های اساسی در استفاده از لاگ فایل‌ها، حجم بالای داده‌ها است. شبکه‌های بزرگ و پیچیده تولید حجم زیادی از لاگ‌ها می‌کنند که تحلیل دستی آن‌ها غیرممکن است. از این رو، نیاز به ابزارهای هوشمند و خودکار برای پردازش و تحلیل سریع لاگ‌ها اهمیت بالایی دارد.

## ۲-۴- توپولوژی شبکه

توپولوژی شبکه به ساختار فیزیکی یا منطقی شبکه‌های کامپیوتری اشاره دارد که نحوه اتصال تجهیزات مختلف شبکه، از جمله سوئیچ‌ها، روترها، سرورها و کامپیوترها را نشان می‌دهد. این توپولوژی‌ها نقش مهمی در کارایی، پایداری و قابلیت گسترش شبکه دارند. انتخاب توپولوژی مناسب برای یک شبکه به نیازهای خاص آن شبکه و کاربردهایی که از آن انتظار می‌رود، بستگی دارد. توپولوژی‌های مختلف شبکه می‌توانند بر اساس نوع داده‌هایی که باید منتقل شوند، میزان ترافیک شبکه، و حساسیت شبکه به قطعی‌ها و خرابی‌ها به کار گرفته شوند. توپولوژی شبکه تأثیر بسزایی در عملکرد و بهره‌وری یک شبکه دارد و انتخاب درست آن می‌تواند به بهبود کارایی و کاهش هزینه‌های نگهداری و مدیریت کمک کند. در شبکه‌های نرم‌افزارمحور نیز توپولوژی به دلیل جداسازی لایه کنترل و لایه داده، همچنان نقشی کلیدی در پایداری و کارایی شبکه بازی می‌کند (Hammadi and Mhamdi, 2014).

از مهم‌ترین توپولوژی‌ها می‌توان به موارد زیر اشاره کرد:

- **توپولوژی ستاره:** توپولوژی ستاره یکی از رایج‌ترین و ساده‌ترین ساختارهای شبکه است که در آن تمام گره‌ها به یک دستگاه مرکزی مانند یک سوئیچ یا هاب متصل می‌شوند. این دستگاه مرکزی نقش مدیریت و هدایت ترافیک شبکه را بر عهده دارد و به عنوان نقطه اصلی ارتباطی عمل می‌کند. یکی از مزایای اصلی توپولوژی ستاره، ساده بودن نصب، راه‌اندازی و مدیریت آن است. علاوه بر این، در صورت خرابی یکی از گره‌ها یا کابل‌های متصل به آن، عملکرد سایر گره‌ها مختل نمی‌شود؛ زیرا هر اتصال به صورت مستقل به دستگاه مرکزی متصل است. با این حال، نقطه ضعف اصلی این توپولوژی، وابستگی زیاد به دستگاه مرکزی است؛ در صورت خرابی این دستگاه، کل شبکه دچار اختلال می‌شود. توپولوژی ستاره به دلیل ساختار ساده و کارایی بالا، معمولاً در شبکه‌های محلی کوچک مانند دفاتر کاری، مدارس و محیط‌های خانگی مورد استفاده قرار می‌گیرد. این توپولوژی همچنین از نظر مقیاس‌پذیری مناسب است، زیرا افزودن گره‌های جدید تنها با اتصال آن‌ها به دستگاه مرکزی امکان‌پذیر است.

- **توپولوژی مش:** توپولوژی مش<sup>۱</sup> یکی از ساختارهای پیشرفته در طراحی شبکه است که در آن هر گره به صورت مستقیم به یک یا چند گره دیگر متصل می‌شود. این توپولوژی به دو صورت مش کامل و مش جزئی قابل پیاده‌سازی است. در مش کامل، تمام گره‌ها به یکدیگر متصل هستند، در حالی که در مش جزئی تنها برخی از گره‌ها ارتباط مستقیم دارند. توپولوژی مش به دلیل وجود مسیرهای متعدد بین گره‌ها، از افزونگی بالایی برخوردار است و امکان بازدهی و پایداری بیشتری را در شبکه فراهم می‌کند. یکی از مزایای اصلی این توپولوژی، تحمل خطای بالا است؛ به این صورت که اگر یکی از مسیرها دچار مشکل شود، مسیرهای جایگزین به طور خودکار استفاده می‌شوند. این ویژگی، مش را به گزینه‌ای ایده‌آل برای شبکه‌های حیاتی مانند مراکز داده و شبکه‌های نظامی تبدیل می‌کند. با این حال، پیچیدگی در پیاده‌سازی و هزینه بالای کابل کشی و تجهیزات مورد نیاز، از معایب این توپولوژی به شمار می‌روند. توپولوژی مش معمولاً در شبکه‌هایی با نیاز به اطمینان بالا، پایداری و قابلیت تحمل خطا به کار گرفته می‌شود.
- **توپولوژی رینگ:** توپولوژی رینگ<sup>۲</sup> یکی از ساختارهای شبکه‌ای است که در آن هر گره به دو گره مجاور خود متصل می‌شود و یک حلقه یا دایره بسته را تشکیل می‌دهد. در این توپولوژی، داده‌ها از یک گره به گره بعدی منتقل می‌شوند تا به مقصد نهایی برسند. ارتباط در توپولوژی رینگ می‌تواند به صورت یک‌طرفه یا دوطرفه باشد؛ در حالت یک‌طرفه، داده‌ها تنها در یک جهت خاص حرکت می‌کنند، در حالی که در حالت دوطرفه، داده‌ها می‌توانند در هر دو جهت جریان داشته باشند. مزیت اصلی این توپولوژی، مدیریت موثر ترافیک شبکه در مقیاس کوچک و هزینه پایین کابل کشی است. با این حال، یکی از معایب قابل توجه آن این است که خرابی یک گره یا قطع شدن یک اتصال می‌تواند کل شبکه را تحت تأثیر قرار دهد.
- **توپولوژی خطی:** توپولوژی خطی یکی از ساده‌ترین انواع توپولوژی‌ها است که در آن تمامی دستگاه‌ها به یک کابل اصلی یا "باس"<sup>۳</sup> متصل می‌شوند. هر دستگاه می‌تواند به طور مستقیم از طریق این کابل با دستگاه‌های دیگر ارتباط برقرار کند. این توپولوژی به دلیل سادگی و هزینه کم، در شبکه‌های کوچک یا موقتی استفاده می‌شود. با این حال، یک نقطه ضعف اصلی آن این است که اگر کابل اصلی قطع شود، کل شبکه مختل می‌شود. همچنین، مدیریت ترافیک در شبکه‌های با ترافیک بالا در این توپولوژی مشکل‌ساز می‌شود.
- **توپولوژی درختی:** توپولوژی درختی<sup>۴</sup> ترکیبی از توپولوژی ستاره و توپولوژی خطی است که ساختاری سلسله‌مراتبی را ایجاد می‌کند. در این توپولوژی، گره‌های شبکه به صورت لایه‌بندی شده به یکدیگر متصل هستند، به طوری که یک گره مرکزی در راس ساختار (ریشه) قرار دارد و گره‌های دیگر به صورت زیرمجموعه‌ای از این گره سازمان‌دهی می‌شوند. هر گره می‌تواند به یک یا چند گره در سطح پایین‌تر متصل باشد، مشابه شاخه‌های

<sup>۱</sup> Mesh Topology<sup>۲</sup> Ring Topology<sup>۳</sup> Bus<sup>۴</sup> Tree Topology

یک درخت. یکی از مزایای توپولوژی درختی، مقیاس‌پذیری بالای آن است، زیرا می‌توان به‌سادگی گره‌های جدید را به سطوح پایین‌تر اضافه کرد. علاوه بر این، مدیریت و عیب‌یابی شبکه در این توپولوژی به دلیل ساختار سلسله‌مراتبی آن نسبتاً آسان است. با این حال، خرابی گره‌های مرکزی در هر سطح می‌تواند باعث قطع ارتباط گره‌های زیرمجموعه شود. توپولوژی درختی معمولاً در شبکه‌های بزرگ و پیچیده مانند مراکز داده و شبکه‌های سازمانی استفاده می‌شود، جایی که نیاز به پشتیبانی از تعداد زیادی دستگاه و ایجاد ساختاری منظم وجود دارد.

- **توپولوژی فت‌تری:** توپولوژی فت‌تری<sup>۱</sup> یک ساختار شبکه‌ای سلسله‌مراتبی است که به‌طور گسترده در مراکز داده مورد استفاده قرار می‌گیرد و نسخه‌ای اصلاح‌شده از توپولوژی درختی است. این توپولوژی با ارائه چندین مسیر موازی بین گره‌ها، پهنای باند بیشتری در مقایسه با توپولوژی درختی معمولی فراهم می‌کند. توپولوژی فت‌تری شامل سه لایه اصلی است: لایه هسته، لایه تجمیع، و لایه دسترسی. لایه هسته وظیفه اتصال تمام بخش‌های شبکه را بر عهده دارد، لایه تجمیع برای ایجاد ارتباط میان لایه دسترسی و لایه هسته به کار می‌رود، و لایه دسترسی مستقیماً به سرورها یا گره‌های نهایی متصل است. یکی از ویژگی‌های برجسته این توپولوژی، افزونگی بالا و کاهش نقاط شکست است، زیرا مسیرهای متعددی برای ارتباط بین گره‌ها وجود دارد. این ساختار به مراکز داده امکان می‌دهد تا ترافیک شبکه را به‌طور یکنواخت توزیع کرده و از ازدحام جلوگیری کنند. با این حال، توپولوژی فت‌تری به دلیل پیچیدگی در طراحی و نیاز به سوئیچ‌های متعدد، هزینه‌بر است. این توپولوژی انتخابی مناسب برای محیط‌هایی با ترافیک سنگین و نیاز به پایداری بالا مانند ابررایانه‌ها و مراکز داده بزرگ است.

## ۲-۵- خطا در شبکه‌های نرم‌افزار محور

شبکه‌های نرم‌افزار محور مستعد بروز خطاها و باگ‌هایی هستند که می‌تواند بر عملکرد شبکه تأثیرات منفی بگذارد. خطاهای شبکه‌های نرم‌افزار محور ممکن است ناشی از پیکربندی نادرست، خطاهای نرم‌افزاری، یا رویدادهای پیش‌بینی‌نشده در شبکه باشند که در نهایت منجر به از دست رفتن داده‌ها، افزایش تأخیر و حتی قطعی کل شبکه می‌شوند (Al-Fares et al., 2008; Beckett et al., 2017).

در پژوهش‌هایی که بر روی تحلیل باگ‌ها در شبکه‌های نرم‌افزار محور انجام شده، نشان داده شده است که بسیاری از باگ‌ها قابل پیش‌بینی و تکرارپذیر هستند. در واقع، خطاهای پیکربندی یکی از رایج‌ترین انواع خطاها در شبکه‌های نرم‌افزار محور به شمار می‌روند. این خطاها می‌توانند در نتیجه تنظیمات اشتباه کنترل‌کننده‌ها یا تعاملات نادرست بین لایه کنترل و لایه داده ایجاد شوند. به عنوان مثال، در برخی از موارد، کنترلر ممکن است به دلیل پیکربندی نادرست قادر به هدایت صحیح جریان‌های داده نباشد که منجر به ایجاد اختلال در عملکرد شبکه می‌شود (Bhardwaj et al., 2021; Hamilton., 2009). علاوه بر خطاهای پیکربندی، خطاهای عملکردی نیز به دلیل ضعف‌های نرم‌افزاری در

<sup>۱</sup> FatTree Topology

کنترلرها یا مشکلات برنامه‌نویسی بروز می‌کنند. این خطاها ممکن است باعث از کار افتادن کامل کنترلر یا کاهش کارایی آن شوند. به عنوان مثال، در برخی از پژوهش‌ها مشاهده شده که باگ‌های نرم‌افزاری می‌توانند منجر به بروز تأخیرهای غیرمنتظره یا خرابی کنترلرهای شبکه شوند که این امر می‌تواند تأثیرات مخربی بر شبکه داشته باشد. این نوع خطاها ممکن است به دلیل عدم تطابق بین نسخه‌های مختلف نرم‌افزار کنترلر یا اشکالات در پیاده‌سازی پروتکل‌های ارتباطی رخ دهند (Bhardwaj et al., 2021; Elsayed et al., 2020).

یکی دیگر از چالش‌های اصلی در شبکه‌های نرم‌افزار محور، خطاهای ناشی از تغییرات پویا است. شبکه‌های نرم‌افزار محور به دلیل پویایی بالا و امکان تغییر سریع پیکربندی‌ها، ممکن است در حین به‌روزرسانی‌های لحظه‌ای با خطاهایی مواجه شوند. این خطاها به ویژه در شبکه‌هایی که از توپولوژی‌های پیچیده مانند فرتری استفاده می‌کنند، بیشتر مشاهده می‌شود. مدیریت صحیح تغییرات و جلوگیری از بروز خطاهای ناشی از ناهماهنگی بین لایه کنترل و لایه داده یکی از چالش‌های مهم در این نوع شبکه‌هاست (Estan and Varghese, 2002; Olmezoglu, 2022; Zeng et al., 2012).

از مهم‌ترین ویژگی‌ها و خطاهای شبکه می‌توان به موارد زیر اشاره کرد:

- **ویژگی دسترسی پذیری:** ویژگی دسترسی پذیری<sup>۱</sup> یکی از ویژگی‌های کلیدی در شبکه‌ها است که به توانایی ارسال بسته‌های داده از یک گره به گره دیگر در یک توپولوژی مشخص اشاره دارد. این ویژگی تضمین می‌کند که مسیرهای ارتباطی مورد نیاز بین گره‌ها وجود دارند و داده‌ها می‌توانند به مقصد مورد نظر برسند. دسترسی پذیری برای عملکرد صحیح شبکه‌ها، به ویژه در محیط‌هایی مانند شبکه‌های نرم‌افزار محور یا مراکز داده، اهمیت حیاتی دارد. این ویژگی می‌تواند تحت تأثیر عواملی مانند تنظیمات نادرست جریان‌ها، خرابی لینک‌ها یا سوئیچ‌ها، و سیاست‌های امنیتی محدودکننده قرار گیرد.
- **خطای سیاه چاله:** خطای سیاه چاله<sup>۲</sup> یکی از مشکلات جدی در شبکه‌ها است که در آن بسته‌های داده‌ای که باید به مقصد خاصی منتقل شوند، در یک گره یا مسیر خاص "گم" می‌شوند و به مقصد نمی‌رسند. این مشکل معمولاً زمانی رخ می‌دهد که یک گره یا مسیر به درستی پیکربندی نشده باشد یا دچار خرابی شود و نتواند بسته‌ها را به گره بعدی منتقل کند. در این سناریو، بسته‌ها به گره معیوب وارد می‌شوند اما به دلیل عدم وجود مسیریابی صحیح، یا حذف می‌شوند یا در گره معیوب باقی می‌مانند. این خطا می‌تواند ناشی از مسائل مختلفی مانند خطای نرم‌افزاری، خرابی سخت‌افزاری، مشکلات در جدول‌های مسیریابی، یا حملات عمدی برای اختلال در شبکه باشد.
- **خطای جداسازی:** خطای جداسازی<sup>۳</sup> به حالتی در شبکه اشاره دارد که یک یا چند گره به صورت هدفمند از

<sup>۱</sup> Reachability

<sup>۲</sup> Black Hole

<sup>۳</sup> Isolation

بعضی از گره‌ها جدا شده‌اند و قادر به برقراری ارتباط با سایر گره‌ها یا بخش‌های شبکه نیستند. در صورت ایجاد ارتباط با گره‌های گروه جداسازی شده خطا رخ خواهد داد. این خطا معمولاً به دلیل قطع ارتباطات فیزیکی، خرابی در لینک‌ها یا سوئیچ‌ها، تنظیمات نادرست جریان‌ها یا سیاست‌های مسیریابی ناصحیح رخ می‌دهد. در این وضعیت، گره‌های جدا شده نمی‌توانند داده‌ها را ارسال یا دریافت کنند، که می‌تواند به کاهش کارایی و قطع خدمات در بخش‌های خاصی از شبکه منجر شود.

• **خطای وضعیت مسابقه:** خطای وضعیت مسابقه<sup>۱</sup> زمانی در شبکه رخ می‌دهد که دو یا چند رویداد همزمان یا نزدیک به هم اتفاق می‌افتند و نتیجه آن‌ها به ترتیب وقوع این رویدادها وابسته است. این وضعیت می‌تواند باعث بروز رفتارهای غیرقابل پیش‌بینی یا ناهماهنگ در شبکه شود. در شبکه‌های نرم‌افزار محور، این خطا معمولاً زمانی رخ می‌دهد که کنترل‌کننده دستورات متناقض یا هماهنگ‌نشده‌ای را به سوئیچ‌ها ارسال کند، به‌طوری‌که ترتیب اجرای این دستورات باعث ایجاد حالت‌های نامطلوب در جدول‌های جریان یا مسیرهای داده شود. برای مثال، اگر یک سوئیچ پیش از دریافت تنظیمات جدید یک جریان، جریان قبلی را حذف کند، ممکن است بسته‌های داده به مقصد نرسند یا به مسیرهای نادرست هدایت شوند. این خطا می‌تواند باعث قطع ارتباط، نقض امنیت، یا کاهش کارایی شبکه شود. جلوگیری از این خطا نیازمند مکانیزم‌های هماهنگ‌سازی قوی بین کنترل‌کننده و سوئیچ‌ها، قفل‌گذاری در تنظیمات جریان‌ها، و تحلیل دقیق ترتیب دستورات است. این خطا به‌ویژه در شبکه‌هایی با تغییرات پویای مکرر و مقیاس بزرگ، چالشی مهم محسوب می‌شود.

• **خطای حلقه:** خطای حلقه<sup>۲</sup> زمانی در شبکه رخ می‌دهد که یک حلقه ناخواسته در مسیرهای مسیریابی ایجاد شود و بسته‌های داده به‌طور مداوم در این حلقه گیر کنند، بدون اینکه به مقصد نهایی برسند. این خطا معمولاً به دلیل تنظیمات نادرست جدول‌های مسیریابی، مشکلات در پروتکل‌های مسیریابی، یا خطاهای پیکربندی سوئیچ‌ها رخ می‌دهد. این خطا می‌تواند باعث اشباع پهنای باند، افزایش تاخیر، و کاهش کارایی شبکه شود، زیرا بسته‌های گیر کرده در حلقه به‌طور مکرر ارسال می‌شوند و منابع شبکه را مصرف می‌کنند.

## ۲-۶- زبان‌های توصیف شبکه‌های نرم‌افزار محور

زبان‌های برنامه‌نویسی<sup>۳</sup> و روش‌های صوری<sup>۴</sup> به عنوان ابزارهای قدرتمند برای طراحی، تحلیل، و بررسی سیستم‌های پیچیده مانند شبکه‌های نرم‌افزار محور به کار می‌روند. این روش‌ها از طریق مدل‌سازی دقیق رفتارهای شبکه و بررسی صحت عملکرد آن‌ها، به مهندسان شبکه امکان می‌دهند تا قبل از پیاده‌سازی نهایی، از درستی عملکرد شبکه اطمینان

<sup>۱</sup> Race Condition

<sup>۲</sup> Loop

<sup>۳</sup> Programming Languages

<sup>۴</sup> Formal Methods

حاصل کنند. زبان‌های برنامه‌نویسی سطح بالا با انتزاع از پیچیدگی‌های فنی و ارائه ابزارهای ساده‌تر برای تعریف و مدیریت سیاست‌های شبکه، به بهبود فرآیند مدیریت شبکه کمک می‌کنند. از طرف دیگر، روش‌های رسمی از ریاضیات و منطق برای تعریف و اثبات صحت برنامه‌ها و سیاست‌های شبکه استفاده می‌کنند و تضمین می‌کنند که سیستم باگ‌ها و خطاهای احتمالی را پیش‌بینی و اصلاح کند. برای اطمینان از عملکرد صحیح شبکه‌ها و جلوگیری از بروز خطاها، استفاده از زبان‌های صوری برای توصیف و مدل‌سازی رفتار شبکه ضروری است.

در دهه گذشته، زبان‌های مختلفی برای توصیف و مدل‌سازی شبکه‌های نرم‌افزارمحور معرفی شده‌اند که هر کدام با هدف بهبود کارایی، امنیت و صحت عملکرد شبکه طراحی شده‌اند. یکی از شناخته‌شده‌ترین این زبان‌ها نتکت<sup>۱</sup> (Anderson et al., 2014) است که برای توصیف رفتار شبکه‌های نرم‌افزارمحور به کار می‌رود. نتکت با استفاده از جبرهای منطقی و ریاضی، به کاربران این امکان را می‌دهد تا سیاست‌های شبکه را به صورت صوری تعریف و تحلیل کنند. این زبان نه تنها از ساده‌سازی فرآیند توصیف شبکه حمایت می‌کند، بلکه به اطمینان از صحت اجرای سیاست‌ها در شبکه کمک می‌کند (Smolka et al., 2015).

داینکت<sup>۲</sup> (Caltais et al., 2021) به عنوان توسعه‌ای بر نتکت، قابلیت‌های بیشتری برای تحلیل شبکه‌های پویا ارائه می‌دهد. در حالی که نتکت بیشتر به توصیف شبکه‌های ایستا می‌پردازد، داینکت با افزودن ویژگی‌های مرتبط با پویایی شبکه، امکان مدل‌سازی تغییرات لحظه‌ای و به‌روزرسانی‌های شبکه را فراهم می‌کند. این زبان توانسته است به عنوان یک ابزار قدرتمند در پیش‌بینی و مدیریت خطاها و تغییرات در شبکه‌های نرم‌افزارمحور شناخته شود. توضیح دقیق و جامع دو زبان رسمی نتکت و داینکت، از جمله ساختار، کاربردها، و نحوه عملکرد آن‌ها در مدیریت و توصیف شبکه‌های نرم‌افزارمحور، در فصل بعدی مورد بررسی قرار خواهد گرفت (Olmezoglu, 2022).

## ۲-۷- کارهای پیشین

در زمینه شبکه‌های نرم‌افزارمحور، پژوهش‌های متعددی به بررسی خطاها، مشکلات همزمانی، و چالش‌های اشکال‌زدایی در این نوع شبکه‌ها پرداخته‌اند. یکی از مسائل اصلی در شبکه‌های نرم‌افزار محور، مدیریت پیچیدگی‌های ناشی از کنترلرها و تعاملات پیچیده بین کنترلر و تجهیزات شبکه است. این پیچیدگی‌ها اغلب منجر به بروز خطاهای پیش‌بینی‌نشده و تداخل در عملکرد شبکه می‌شوند. به همین دلیل، بسیاری از پژوهش‌ها بر روی تحلیل و مدیریت این مشکلات و توسعه ابزارهایی برای بهبود کارایی و پایداری شبکه‌های نرم‌افزار محور تمرکز کرده‌اند. به عنوان مثال، مطالعه‌ای که توسط Bhardwaj و همکاران انجام شد، به تحلیل بیش از ۵۰۰ باگ بحرانی در سه کنترلر مهم شبکه‌های نرم‌افزار محور (FAUCET، ONOS، CORD) پرداخت و نشان داد که این کنترلرها به دلیل پیچیدگی نرم‌افزار، به طور مداوم در معرض خطاهایی قرار دارند که می‌توانند بر عملکرد شبکه تأثیر بگذارند (Bhardwaj et al., 2021).

<sup>۱</sup> NetKAT

<sup>۲</sup> DyNetKAT

پژوهش‌های دیگری نیز به طبقه‌بندی انواع باگ‌ها پرداخته‌اند. این طبقه‌بندی‌ها به درک بهتری از نوع و ریشه این باگ‌ها کمک می‌کنند و راهکارهای بهبود عملکرد کنترلرها و کاهش خطاها را ارائه می‌دهند. برای مثال، در برخی مطالعات نشان داده شده که بیشتر باگ‌های شبکه‌های نرم‌افزار محور از نوع باگ‌های قطعی<sup>۱</sup> هستند که این امکان را فراهم می‌کند تا با استفاده از تکنیک‌های مشخص، این باگ‌ها شناسایی و رفع شوند. با این حال، همچنان برخی از باگ‌ها ناشی از رویدادهای پویا در شبکه هستند که نیازمند تحقیقات بیشتری در زمینه بهبود ابزارهای شناسایی و بازیابی خطاها هستند.

یکی از مهم‌ترین کارهای پیشین در این زمینه، مقاله

### Troubleshooting Blackbox SDN Control Software with Minimal Causal Sequences

است. این پژوهش به بررسی مشکلات اشکال‌زدایی در شبکه‌های نرم‌افزارمحور با سیستم‌های جعبه سیاه<sup>۲</sup> پرداخته و راهکاری به نام دنباله‌های علی حداقلی<sup>۳</sup> را ارائه داده است. این دنباله‌ها به مدیران شبکه امکان می‌دهند تا بدون دسترسی کامل به جزئیات داخلی سیستم، علت بروز خطاها را با کمترین اطلاعات ممکن شناسایی کنند. این روش با کاهش پیچیدگی‌های اشکال‌زدایی و استفاده از ورودی و خروجی‌های شبکه، امکان شناسایی سریع و دقیق خطاها را فراهم می‌کند (Scott et al., 2014).

پژوهش دیگر، مقاله

### SDNRacer: Concurrency Analysis for Software-Defined Networks

است که به تحلیل مشکلات همزمانی در شبکه‌های نرم‌افزار محور پرداخته است. در این پژوهش، ابزار SDNRacer معرفی شده که قادر به شبیه‌سازی تعاملات مختلف شبکه و شناسایی Race Conditions یا شرایط رقابتی است. این شرایط زمانی رخ می‌دهند که چندین رویداد به صورت همزمان به منابع دسترسی پیدا می‌کنند و باعث تداخل در عملکرد شبکه می‌شوند. (El-Hassany et al., 2016).

در حالی که بسیاری از پژوهش‌های پیشین در زمینه شبکه‌های نرم‌افزار محور بر تحلیل خطاهای موجود، اشکال‌زدایی، و مدیریت مشکلات همزمانی متمرکز بوده‌اند، کار این پژوهش رویکردی متفاوت و پیشگیرانه را دنبال می‌کند. هدف اصلی این پژوهش، بدست آوردن رفتار رسمی شبکه از روی لاگ فایل‌های موجود است. این رفتار رسمی با استفاده از زبان‌های برنامه‌نویسی و ابزارهای تحلیلی، مدل‌سازی و استخراج می‌شود تا رفتار کلی شبکه و تعاملات آن بین کنترلرها و سوئیچ‌ها به صورت دقیق شبیه‌سازی شود. برخلاف بسیاری از تحقیقات که به ریشه‌یابی خطاهای فعلی می‌پردازند، در این پروژه

<sup>۱</sup> Deterministic

<sup>۲</sup> Blackbox

<sup>۳</sup> Minimal Causal Sequences

فرض می‌شود که شبکه در وضعیت امن قرار دارد و هیچ خطایی رخ نداده است.

تمرکز این کار بر پیش‌بینی خطاهای احتمالی در آینده است. به این معنا که بررسی می‌کنیم اگر چندین تغییر اشتباه در پیکربندی شبکه انجام شود، آیا این تغییرات می‌توانند به شرایطی منجر شوند که ویژگی‌های امنیتی یا عملکردی شبکه نقض شود. به عبارت دیگر، به جای تمرکز بر ردیابی و رفع خطاهای موجود، این پژوهش تلاش می‌کند مسیرهای بالقوه ایجاد خطا در آینده را شناسایی کند و رفتار شبکه را پیش از وقوع خطا مدل‌سازی کند.



## فصل ۳: توصیف شبکه‌های نرم‌افزار محور

---

## ۳-۱- مقدمه

روش‌های صوری<sup>۱</sup> به عنوان ابزاری قدرتمند برای طراحی، تحلیل و اثبات صحت سیستم‌های پیچیده مانند شبکه‌های نرم‌افزار محور مورد استفاده قرار می‌گیرند. هدف از استفاده از روش‌های صوری، ایجاد مدل‌های ریاضی دقیق از سیستم‌ها است تا بتوان رفتار آن‌ها را پیش از پیاده‌سازی نهایی به طور کامل بررسی کرد. این روش‌ها با بهره‌گیری از ابزارهای منطقی و ریاضیاتی، به طراحان شبکه امکان می‌دهند تا از صحت عملکرد شبکه، بدون نیاز به تست‌های پیاده‌سازی شده و پرهزینه، اطمینان حاصل کنند.

زبان‌های برنامه‌نویسی<sup>۲</sup> صوری یکی از مهم‌ترین ابزارهای روش‌های صوری هستند که به صورت ریاضی و منطقی برای توصیف رفتار سیستم‌ها طراحی شده‌اند. این زبان‌ها به طور خاص به کاربران امکان می‌دهند تا به جای استفاده از روش‌های معمول برنامه‌نویسی که در آن‌ها خطاهای احتمالی ممکن است نادیده گرفته شوند، از روش‌های رسمی و قابل اثبات برای توصیف سیستم‌های خود استفاده کنند. زبان‌های صوری نه تنها به تعریف سیاست‌ها و قوانین شبکه کمک می‌کنند، بلکه از طریق مدل‌سازی و اثبات، صحت عملکرد شبکه را تضمین می‌کنند.

یکی از کاربردهای مهم این زبان‌ها در شبکه‌های نرم‌افزار محور، بررسی رفتارهای ایستا و پویا در شبکه‌ها است. در این راستا، زبان‌هایی مانند نتکت و داینکت با بهره‌گیری از جبر کلین<sup>۳</sup> و مفاهیم ریاضیاتی دیگر، ابزارهای موثری را برای توصیف و تحلیل شبکه‌های نرم‌افزار محور ارائه می‌دهند. نتکت به عنوان یک زبان رسمی برای توصیف رفتار ایستای شبکه‌ها طراحی شده است و با استفاده از اصول ریاضی و منطقی، صحت سیاست‌های شبکه را تضمین می‌کند. از سوی دیگر، داینکت با گسترش قابلیت‌های نتکت، امکان توصیف به‌روزرسانی‌های پویا و مدیریت شبکه‌های دینامیک را فراهم می‌کند.

در این فصل ابتدا مروری بر جبر کلین با آزمون<sup>۴</sup> می‌کنیم و سپس، با تمرکز بر زبان‌های نتکت و داینکت، به بررسی دقیق این روش‌های صوری می‌پردازیم و نحوه استفاده از آن‌ها برای مدیریت و تحلیل شبکه‌های نرم‌افزار محور را مورد بحث قرار خواهیم داد. در نهایت، یک مثال عملی از کاربرد این زبان‌ها ارائه خواهد شد تا نشان دهد چگونه این ابزارهای صوری به بهبود کارایی و اطمینان از صحت عملکرد شبکه‌ها کمک می‌کنند.

<sup>۱</sup> Formal Methods

<sup>۲</sup> Formal Programming Languages

<sup>۳</sup> Klenner Algebra

<sup>۴</sup> Kleene Algebra with Tests

### ۳-۲- جبر کلین با آزمون

جبر کلین با آزمون یکی از ابزارهای مهم در حوزه روش‌های صوری و منطق ریاضی است که برای مدل‌سازی و تحلیل سیستم‌های پیچیده به کار می‌رود. این سیستم جبری به‌عنوان یک ابزار مهم در علوم کامپیوتر و نظریه محاسبات شناخته می‌شود و در تحلیل و بررسی فرآیندهای بازگشتی، زبان‌های صوری، و سیستم‌های کنترلی استفاده می‌شود. روش‌های صوری مانند جبر کلین و نسخه توسعه‌یافته آن، به طراحان سیستم‌ها کمک می‌کنند تا رفتار سیستم‌های خود را به صورت دقیق و ریاضیاتی توصیف کرده و از صحت عملکرد آن‌ها اطمینان حاصل کنند.

جبر کلین یک سیستم جبری است که توسط استیفن کلین<sup>۱</sup> در دهه ۱۹۵۰ معرفی شد. این سیستم در اصل برای توصیف و تحلیل فرآیندهای بازگشتی و مسیرهای محاسباتی به کار می‌رود. جبر کلین شامل مجموعه‌ای از عملیات‌های ریاضیاتی است که به کمک آن‌ها می‌توان رفتارهای مختلف سیستم را مدل‌سازی کرد. این جبر به طور گسترده در علوم کامپیوتر، به ویژه در نظریه زبان‌های صوری و اتوماتا<sup>۲</sup>، استفاده می‌شود. در جبر کلین، عملیات‌هایی مانند اتحاد (Union)، ترکیب (Concatenation) و ستاره کلین (Kleene Star) به کار گرفته می‌شوند. این عملیات‌ها به توصیف رشته‌ها و مسیرهای مختلف در سیستم‌های محاسباتی کمک می‌کنند. به عنوان مثال، عملگر ستاره کلین برای توصیف تکرار یک عملیات یا فرآیند تا زمان نامحدود استفاده می‌شود. با ترکیب این عملگرها، می‌توان زبان‌های صوری یا رفتارهای محاسباتی مختلف را به‌طور دقیق توصیف کرد (Kozen, 1997).

جبر کلین با آزمون نسخه‌ای توسعه‌یافته از جبر کلین است که برای مدل‌سازی سیستم‌هایی طراحی شده است که در آن‌ها علاوه بر عملیات محاسباتی، شرایط و تست‌ها نیز دخالت دارند. در سیستم‌های پیچیده مانند شبکه‌های نرم‌افزار محور، تنها توصیف مسیرها کافی نیست، بلکه باید شرایط و قوانین خاصی که مسیرهای محاسباتی را تحت تأثیر قرار می‌دهند نیز در نظر گرفته شود. جبر کلین با آزمون این قابلیت را اضافه می‌کند که بتوان تست‌ها و شرایط منطقی را به جبر کلین اضافه کرد. در جبر کلین با آزمون، علاوه بر عملیات‌های معمول جبر کلین، تست‌ها نیز به‌عنوان یک جزء اصلی وارد می‌شوند. این تست‌ها به‌صورت عناصر بولی در جبر عمل می‌کنند و می‌توانند شرایطی مانند "آیا این مسیر ممکن است؟" یا "آیا این قانون معتبر است؟" را بررسی کنند. به بیان دیگر، جبر کلین با آزمون به شما اجازه می‌دهد که نه تنها مسیرهای ممکن در یک سیستم را مدل‌سازی کنید، بلکه شرایط لازم برای تحقق آن مسیرها را نیز تعیین کنید (Kozen, 1997).

یکی از کاربردهای اصلی جبر کلین با آزمون در توصیف و تحلیل سیاست‌های شبکه‌های نرم‌افزار محور است. با استفاده از جبر کلین با آزمون، می‌توان سیاست‌های مختلف شبکه را به صورت صوری تعریف کرد و اطمینان حاصل کرد که این سیاست‌ها به درستی در شبکه اعمال می‌شوند.

<sup>۱</sup> Stephen Kleene

<sup>۲</sup> Automata

## ۳-۳- نتکت

نتکت<sup>۱</sup> یک زبان برنامه‌نویسی شبکه است که بر اساس مبانی ریاضیاتی جامعی توسعه یافته و برای توصیف، برنامه‌ریزی و تحلیل شبکه‌های نرم‌افزار محور استفاده می‌شود. این زبان در ابتدا به عنوان یک ابزار ساده برای توصیف رفتار سوئیچ‌ها و مسیرهای شبکه توسعه داده شده است، اما به مرور زمان قابلیت‌هایی به آن اضافه شده که امکان تحلیل رفتار کلی شبکه و سیاست‌های دسترسی را فراهم می‌کند. هدف اصلی نتکت، فراهم کردن ابزاری برای توصیف رفتار شبکه‌ها به صورت ریاضی و منطقی است، که از یک نظریه کامل و معتبر برخوردار است. این زبان قابلیت‌های مختلفی مانند فیلتر کردن، تغییر، و انتقال بسته‌ها را در خود جای داده است و از عملگرهای ترکیب دنباله‌ای و ستاره کلین برای تکرار برنامه‌ها بهره می‌برد. نتکت بر مبنای جبر کلین با آزمون طراحی شده است. با بهره‌گیری از این ساختار، نتکت امکان تحلیل و بررسی دقیق رفتار شبکه‌ها را فراهم می‌کند (Anderson et al., 2014; Smolka et al., 2015).

به صورت کلی، در نتکت، بسته‌های شبکه به صورت رکوردهایی با فیلدهای مختلف مانند آدرس مبدأ، مقصد، نوع پروتکل، سوئیچ و پورت تعریف می‌شوند. سیاست‌های نتکت شامل فیلترها، تغییرات و ترکیبات ترتیبی و موازی هستند. عملگرهای این زبان مانند جمع (+) و ضرب (•) برای ترکیب سیاست‌ها به کار می‌روند و عملگر ستاره کلین (\*) برای تکرار سیاست‌ها استفاده می‌شود. در نتکت، نحو<sup>۴</sup> و معنا<sup>۵</sup>ی زبان به صورت رسمی در جدول (۱-۳) تعریف شده است. عبارات نتکت به دو دسته تقسیم می‌شوند: پیش‌شرط‌ها و سیاست‌ها. پیش‌شرط‌ها به عنوان فیلترها عمل می‌کنند و بسته‌هایی که شرایط مشخصی را دارند انتخاب می‌کنند. سیاست‌ها شامل عملیات‌هایی مانند تغییر فیلدهای بسته و اعمال قوانین مختلف بر روی آن‌ها هستند.

## ۳-۳-۱- نحو زبان نتکت

همانطور که در جدول (۱-۳) مشاهده می‌کنید نحو نتکت به صورت زیر می‌باشد:

• **Fields:** فیلدها به عنوان اجزای اطلاعاتی بسته‌ها در شبکه تعریف می‌شوند که هر یک از آن‌ها دارای مقادیری

هستند. به عنوان مثال، هر بسته می‌تواند فیلدهایی مانند آدرس مبدأ، مقصد، پورت و غیره داشته باشد.

• **Packets:** بسته‌ها شامل مجموعه‌ای از فیلدها هستند که مقدار مشخصی به آن‌ها اختصاص داده شده است.

<sup>۱</sup> Network Kleene Algebra with Tests (NetKAT)

<sup>۲</sup> Packet

<sup>۳</sup> Policy

<sup>۴</sup> Syntax

<sup>۵</sup> Syntax

Syntax	
$f ::= f_1   \dots   f_k$	<b>Fields</b>
$pk ::= \{f_1 = v_1, \dots, f_k = v_k\}$	<b>Packets</b>
$h ::= pk :: \langle \rangle \mid pk :: h$	<b>Histories</b>
(Identity) $a, b ::= 1$ (Drop) $0$ (Test) $f = n$ (Disjunction) $a + b$ (Conjunction) $a \cdot b$ (Negation) $\neg a$	<b>Predicates</b>
(Filter) $p, q ::= a$ (Modification) $f \leftarrow n$ (Union) $p + q$ (Sequential composition) $p \cdot q$ (Kleene star) $p^*$ (Duplication) $dup$	<b>Policies</b>

جدول (۳-۱): نحو زبان نتکت

• **Histories:** تاریخچه شامل دنباله‌ای از بسته‌ها است که نشان‌دهنده مسیر و تغییراتی است که یک بسته در طول سفر خود از طریق شبکه طی کرده است.

• **Predicates:** پیش‌شرط‌ها شامل شرایطی هستند که برای فیلتر کردن بسته‌ها به کار می‌روند. این شرایط می‌توانند شامل موارد زیر باشد:

○ 1: عمل همانی

○ 0: حذف بسته

○  $f = n$ : تست برای بررسی یک مقدار خاص در یک فیلد

○  $a + b$ : اجتماع دو پیش‌شرط

○  $a \cdot b$ : اشتراک دو پیش‌شرط

○  $\neg a$ : نقیض یک پیش‌شرط

• **Policies:** سیاست‌ها، عملیات‌هایی هستند که بر روی بسته‌ها اعمال می‌شوند. این سیاست‌ها شامل فیلتر کردن، تغییر فیلدها، ترکیب ترتیبی سیاست‌ها و تکرار آن‌ها است. از جمله عملیات‌های موجود می‌توان به موارد زیر اشاره کرد:

- Filter: فیلتر کردن بسته‌ها
- Modification: تغییر مقدار یک فیلد
- Union: اجتماع دو سیاست
- Sequential Composition: ترکیب ترتیبی سیاست‌ها
- Kleene Star: تکرار یک سیاست به تعداد نامحدود
- Duplication: تکثیر بسته‌ها

### ۳-۳-۲ معنای زبان نتکت

در مقاله نتکت (Anderson et al., 2014)، معنای سیاست‌ها و پیش‌شرط‌ها به صورت فرمول‌هایی نشان داده شده که رفتار آن‌ها را در تاریخچه بسته‌ها توصیف می‌کند. می‌توانید معنای زبان نتکت در جدول (۳-۲) مشاهده کنید.

### ۳-۴-۲ دایننتکت

دایننتکت<sup>۱</sup> یک توسعه پیشرفته از زبان نتکت است که برای مدل‌سازی رفتارهای پویا و پیچیده در شبکه‌های نرم‌افزار محور طراحی شده است. در حالی که نتکت به عنوان ابزاری برای توصیف سیاست‌های شبکه‌های ایستا معرفی شده است، دایننتکت برای مدیریت و توصیف شبکه‌هایی به کار می‌رود که به صورت پیوسته و پویا<sup>۲</sup> تغییر می‌کنند. یکی از مزایای اصلی دایننتکت این است که به کنترلرهای شبکه اجازه می‌دهد بدون نیاز به توقف شبکه، تغییرات را در جریان داده‌ها و جداول جریان اعمال کنند و این امکان را فراهم می‌کند که مدیریت شبکه با انعطاف‌پذیری بیشتری انجام شود (Caltais et al., 2021).

در دایننتکت، سیاست‌ها و تعاملات شبکه از طریق جبرهای ریاضی و عملگرهای مشخص توصیف می‌شوند. این زبان از ترکیب عملگرهای موازی و بازگشتی استفاده می‌کند که امکان توصیف رفتارهای پیچیده‌تر شبکه را فراهم می‌سازد. به عنوان مثال، عملگرهای موازی در دایننتکت این امکان را می‌دهند که چندین تغییر به‌طور همزمان در شبکه رخ دهد،

<sup>۱</sup> Dynamic NetKAT (DyNetKAT)

<sup>۲</sup> Dynamic

Semantics
$\llbracket p \rrbracket \in H \rightarrow P(H)$
$\llbracket 1 \rrbracket h \triangleq \{h\}$ $\llbracket 0 \rrbracket h \triangleq \emptyset$
$\llbracket f = n \rrbracket (pk :: h) \triangleq \begin{cases} \{pk :: h\} & \text{if } pk.f = n \\ \emptyset & \text{otherwise} \end{cases}$
$\llbracket \neg a \rrbracket h \triangleq \{h\} \setminus \llbracket a \rrbracket h$
$\llbracket f \leftarrow n \rrbracket (pk :: h) \triangleq \{pk[f \leftarrow n] :: h\}$
$\llbracket p + q \rrbracket h \triangleq \llbracket p \rrbracket h \cup \llbracket q \rrbracket h$ $\llbracket p \cdot q \rrbracket h \triangleq (\llbracket p \rrbracket \bullet \llbracket q \rrbracket)h$
$\llbracket p^* \rrbracket h \triangleq \bigcup_{i \in \mathbb{N}} F^i h$ where $F^{i+1} h \triangleq (\llbracket p \rrbracket \bullet F^i) \{h\}$ and $F^0 h \triangleq \{h\}$
$\llbracket dup \rrbracket (pk :: h) \triangleq \{pk :: (pk :: h)\}$

جدول (۳-۲): معنای زبان نتکت

بدون اینکه تغییرات یکی بر دیگری تأثیر بگذارد. این ویژگی باعث می‌شود این زبان بتواند به طور موثری تغییرات لحظه‌ای شبکه را مدل‌سازی کرده و رفتارهای مختلف را به‌طور همزمان تحلیل کند.

یکی از ویژگی‌های برجسته داینکت استفاده از بازگشت‌های محافظت‌شده<sup>۱</sup> است که به این زبان امکان می‌دهد تغییرات پویای شبکه را با حفظ رفتارهای ایستا و پایدار مدل‌سازی کند. این بازگشت‌ها به کنترلرها این امکان را می‌دهند که تغییرات مستمر در شبکه را بدون نیاز به تغییرات بنیادین در سیاست‌ها و قوانین شبکه مدیریت کنند. به عنوان مثال، در سناریوهای شبکه‌ای که نیاز به تغییرات مکرر در جریان‌های داده دارند، داینکت می‌تواند این تغییرات را به صورت ایمن و کارآمد مدیریت کند.

<sup>۱</sup> Guarded Recursion

## ۳-۴-۱- نحو و معنای داینتکت

در داینتکت، نحو و معنای این زبان با استفاده از اصول جبری و منطقی (شکل (۳-۱)) توصیف می‌شود. سیاست‌ها در این زبان شامل عملیات‌هایی مانند اجتماع، ترکیب ترتیبی، و انتخاب غیرقطعی<sup>۱</sup> هستند. این عملیات‌ها به کنترلرها اجازه می‌دهند که رفتارهای مختلف شبکه را در سناریوهای متنوع مدل‌سازی کنند. به عنوان مثال، در یک شبکه پیچیده که شامل چندین جریان داده و سیاست‌های متفاوت است، از این زبان می‌توان برای ترکیب و تحلیل این جریان‌ها به صورت همزمان استفاده کرد.

## NetKAT Syntax:

$$\begin{aligned} Pr &::= 0 \mid 1 \mid f = n \mid Pr + Pr \mid Pr \cdot Pr \mid \neg Pr \\ N &::= Pr \mid f \leftarrow n \mid N + N \mid N \cdot N \mid N^* \end{aligned}$$

## DyNetKAT Syntax:

$$\begin{aligned} D &::= \perp \mid N; D \mid x?N; D \mid x!N; D \mid D \parallel D \mid D \oplus D \mid X \\ X &\triangleq D \end{aligned}$$

شکل (۳-۱): نحو زبان داینتکت

در شکل (۳-۱)، نحو داینتکت را مشاهده می‌کنیم. پیش‌شرط برای حذف یک بسته با 0 نشان داده می‌شود، در حالی که عبور دادن یک بسته (بدون هیچ تغییری) با 1 نشان داده می‌شود. پیش‌شرطی که بررسی می‌کند آیا فیلد  $f$  یک بسته دارای مقدار  $n$  است یا خیر، با  $f = n$  نشان داده می‌شود؛ اگر این پیش‌شرط در بسته جاری درست نباشد، نتیجه آن حذف بسته است، در غیر این صورت بسته عبور می‌کند. اجتماع و اشتراک بین پیش‌شرط‌ها به ترتیب با  $Pr + Pr$  و  $Pr \cdot Pr$  نشان داده می‌شود. نقیض یک پیش‌شرط با  $\neg Pr$  نمایش داده می‌شود. سیاستی که فیلد  $f$  بسته جاری را به مقدار  $n$  تغییر می‌دهد، با  $f \leftarrow n$  نشان داده می‌شود. رفتار چندپخشی<sup>۲</sup> سیاست‌ها با  $N + N$  نمایش داده می‌شود، در حالی که ترتیب اعمال سیاست‌ها (برای اعمال شدن بر روی یک بسته) با  $N \cdot N$  نشان داده می‌شود. اعمال مکرر یک سیاست به صورت  $N^*$  کدگذاری می‌شود.

$\perp$  نشان‌دهنده یک سیاست خالی است که هیچ رفتاری ندارد. عملگر ترکیب ترتیبی، که با  $N; D$  نشان داده می‌شود، مشخص می‌کند زمانی که سیاست نتکت به بسته جاری به‌طور موفقیت‌آمیز اعمال شود، بسته می‌تواند به مرحله بعدی ارسال شود و بسته جدیدی برای پردازش بر اساس بقیه سیاست  $D$  فراخوانی شود. ارتباط در داینتکت، که با  $x!N; D$  و  $x?N; D$  کدگذاری شده است، شامل دو مرحله است. در مرحله اول، ارسال و دریافت سیاست‌های نتکت از طریق کانال  $x$  به ترتیب با  $x!N$  و  $x?N$  نشان داده می‌شود. دومین مرحله این است که به محض اینکه پیام‌های ارسالی

<sup>۱</sup> Non-deterministic Choice

<sup>۲</sup> Multicast



یا دریافتی با موفقیت ارتباط برقرار کردند، بسته جدیدی فراخوانی شده و بر اساس  $D$  پردازش می‌شود. ترکیب موازی دو سیاست داینکتک (برای فعال‌سازی همگام‌سازی) با  $D || D$  نشان داده می‌شود. انتخاب غیرقطعی بین دو سیاست نیز به صورت  $D \oplus D$  می‌باشد. در نهایت، می‌توان از متغیرهای بازگشتی  $X$  در مشخصه سیاست‌های داینکتک استفاده کرد، که هر متغیر بازگشتی باید معادله تعریف‌کننده یکتایی به صورت  $X \triangleq D$  داشته باشد.

معنای داینکتک نیز به صورت صوری تعریف شده است و رفتارهای شبکه را بر اساس سیاست‌های تعریف‌شده توصیف می‌کند. این زبان قادر است تا تاریخچه بسته‌ها و تغییرات آن‌ها در شبکه را پیگیری کند و این امکان را به کاربران می‌دهد که رفتارهای مختلف شبکه را با دقت بالا تحلیل کنند. از این رو، این زبان به عنوان یک ابزار قوی برای مدل‌سازی و تحلیل شبکه‌های پیچیده و پویا مطرح می‌شود.

در نهایت، داینکتک به دلیل استفاده از اصول جبری و قابلیت‌های گسترده‌اش در مدیریت شبکه‌های پویا، یکی از زبان‌های مهم در زمینه شبکه‌های نرم‌افزار محور به شمار می‌آید. این زبان نه تنها توانایی مدل‌سازی تغییرات پویا را دارد، بلکه به کنترلرها امکان می‌دهد که از سیاست‌های پیچیده‌تری برای مدیریت جریان‌های شبکه استفاده کنند و خطاهای احتمالی را به حداقل برسانند.

### ۳-۴-۲ - ویژگی‌های ایمنی

در شبکه‌های نرم‌افزار محور، بررسی ویژگی‌های ایمنی برای اطمینان از صحت عملکرد شبکه و جلوگیری از بروز خطاها یا مشکلات اساسی ضروری است. این ویژگی‌ها تضمین می‌کنند که شبکه به درستی وظایف خود را انجام داده و بسته‌های داده طبق قوانین و سیاست‌های تعریف‌شده هدایت می‌شوند. دو مورد از مهم‌ترین ویژگی‌های ایمنی در داینکتک که به طور خاص در ارزیابی و تضمین صحت شبکه مورد توجه قرار می‌گیرند، دسترسی‌پذیری<sup>۱</sup> و در مسیر پذیری<sup>۲</sup> هستند. این دو ویژگی به منظور اطمینان از عملکرد صحیح شبکه در هر لحظه و جلوگیری از بروز ناهنجاری‌ها در جریان داده‌ها به کار می‌روند.

#### دسترسی‌پذیری

دسترسی‌پذیری به معنای بررسی این موضوع است که آیا یک نقطه خروجی از یک نقطه ورودی در شبکه قابل دسترسی است یا خیر. در داینکتک، این ویژگی با استفاده از مفاهیم موجود در نتکت بررسی می‌شود. برای اینکه یک

<sup>۱</sup> reachability

<sup>۲</sup> waypointing

نقطه خروجی در یک شبکه از یک نقطه ورودی دسترسی‌پذیر باشد، باید معادله نتکت برقرار باشد که نشان می‌دهد هیچ بسته‌ای از نقطه ورودی نمی‌تواند به نقطه خروجی برسد مگر اینکه قوانین و سیاست‌های شبکه به درستی اعمال شده باشند.

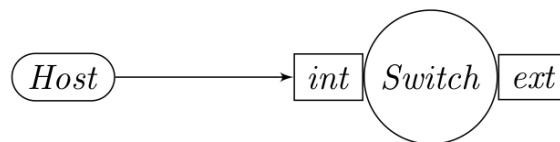
### در مسیرپذیری

این ویژگی بررسی می‌کند که آیا یک نقطه میانی بین نقاط ورودی و خروجی وجود دارد که تمام بسته‌ها از طریق آن عبور کنند. این ویژگی به منظور اطمینان از عبور بسته‌ها از مسیرهای مشخص شده و کنترل ترافیک شبکه به کار می‌رود. برای اینکه یک نقطه به عنوان نقطه میانی در نظر گرفته شود، باید همه بسته‌ها از نقطه ورودی به نقطه خروجی از آن عبور کنند و معادلات نتکت مربوطه برای بررسی این ویژگی استفاده می‌شوند.

## ۳-۵- مثال‌ها

### ۳-۵-۱- دیوار آتش حالت‌مند

برای درک بهتر مفاهیم داینتکت از مثال دیوار آتش حالت‌مند استفاده می‌کنیم. دیوار آتش حالت‌مند به عنوان یک مکانیزم امنیتی، به صورت پویا جدول جریان خود را بر اساس رویدادهای شبکه به‌روزرسانی می‌کند. حالت ساده این شبکه را می‌توانید در شکل (۳-۲) ببینید. در ابتدا، دیوار آتش به گونه‌ای پیکربندی شده است که تمامی بسته‌های ورودی از شبکه خارجی به شبکه داخلی را مسدود کند. اما زمانی که یک درخواست معتبر از شبکه داخلی به یک گره خارجی ارسال می‌شود، دیوار آتش جدول جریان خود را به‌روزرسانی می‌کند تا پاسخ‌های گره خارجی به شبکه داخلی مجاز شوند. این رفتار پویا باعث می‌شود که دیوار آتش علاوه بر تأمین امنیت، انعطاف‌پذیری لازم را برای پاسخ به درخواست‌های شبکه در زمان واقعی فراهم کند.



شکل (۳-۲): دیوار آتش حالت‌مند

در شکل (۳-۳)، توصیف داینتکت برای دیوار آتش در مثال (۳-۲) ارائه شده است. در این توصیف، از کانال پیام secConReq برای باز کردن اتصال و از secConEnd برای بستن آن استفاده شده است. رفتار سوئیچ با استفاده از دو عبارت Switch و 'Switch مدل شده است.

$$Host \triangleq secConReq!1; Host \oplus \\ secConEnd!1; Host$$

$$Switch \triangleq ((port = int) \cdot (port \leftarrow ext)); Switch \oplus \\ ((port = ext) \cdot 0); Switch \oplus \\ secConReq?1; Switch'$$

$$Switch' \triangleq ((port = int) \cdot (port \leftarrow ext)); Switch' \oplus \\ ((port = ext) \cdot (port \leftarrow int)); Switch' \oplus \\ secConEnd?1; Switch$$

$$Init \triangleq Host || Switch$$

شکل (۳-۳): دیوارآتش حالت‌مند

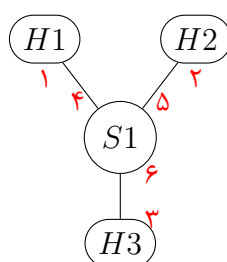
### ۳-۵-۲- مثال دوم: تک سوئیچ

در این پژوهش برای توضیح مفاهیم و روش‌های مطرح‌شده، از مثال شکل (۳-۴) استفاده می‌کنیم. در این مثال از یک توپولوژی ساده شبکه استفاده می‌کنیم که به عنوان یک نقطه مرجع در طول تحلیل و مدل‌سازی مورد استفاده قرار می‌دهیم. این شبکه شامل یک سوئیچ ( $S1$ ) است که به سه میزبان مجزا متصل می‌شود. میزبان ۱ ( $H1$ ) به پورت ۴ سوئیچ  $S1$ ، میزبان ۲ ( $H2$ ) به پورت ۵، و میزبان ۳ ( $H3$ ) به پورت ۶ متصل شده‌اند. در این مثال، فرض بر این است که جدول جریان<sup>۱</sup> سوئیچ  $S1$  از قبل با قانونی تنظیم شده که نحوه ارسال بسته‌ها را تعیین می‌کند. به‌طور خاص، اگر سوئیچ بسته‌ای را از پورت ۴ دریافت کند، مجاز است آن بسته را به پورت ۶ ارسال کند. این قانون از پیش تعریف‌شده در جدول جریان سوئیچ را می‌توان به زبان رسمی داینتکت به صورت عبارت ۳-۱ نمایش داد.

$$X_{S1} \triangleq pt = 4 \cdot pt \leftarrow 6 \quad (۱-۳)$$

در عبارت ۳-۱،  $X_{S1}$  نشانگر جدول جریان سوئیچ  $S1$  است و  $pt = 4$  یک پیش شرط برای بررسی بسته از

<sup>۱</sup> Flow Table



شکل (۳-۴): مثال با یک سوئیچ

پورت ۴ است و  $6 \leftarrow pt$  نشان‌دهنده ارسال بسته به پورت ۶ است. در این مثال، که در آن سوئیچ  $S1$  ابتدا برای ارسال بسته‌ها از پورت ۴ به پورت ۶ تنظیم شده است، ممکن است تغییرات پویا در جدول جریان رخ دهد. برای مثال، اگر شرایط شبکه تغییر کند یا سیاست جدیدی معرفی شود، مانند ارسال بسته‌ها از پورت ۵ به پورت ۶، سوئیچ باید جدول جریان خود را به‌روزرسانی کند تا این نیازهای مسیریابی جدید را پوشش دهد. در چنین مواردی، مدل داینتکت اهمیت زیادی در ارزیابی صحت این به‌روزرسانی‌های پویا دارد. این مدل تضمین می‌کند که جدول جریان جدید به همراه سیاست جدید همچنان رفتار مورد انتظار شبکه را حفظ می‌کند و از بروز مشکلات ناشی از تنظیمات نادرست جلوگیری می‌شود.

## فصل ۴: بررسی فایل لاگ و استخراج توپولوژی

---

## ۴-۱- مقدمه

تا این لحظه، ما به بررسی مفاهیم پایه‌ای مرتبط با شبکه‌های نرم‌افزار محور و ابزارهای رسمی مدل‌سازی مانند نتکت و داینکت پرداخته‌ایم. این مفاهیم به ما درک بهتری از نحوه توصیف و تحلیل رفتار شبکه‌های نرم‌افزار محور ارائه می‌دهند. همچنین، به بررسی ویژگی‌های ایمنی مانند دسترسی‌پذیری و در مسیر پذیری پرداختیم که نقش کلیدی در تضمین صحت و پایداری شبکه‌ها دارند.

در این فصل، تمرکز خود را به بخش عملی و تحلیل داده‌های واقعی شبکه معطوف می‌کنیم. به طور خاص، به بررسی لاگ فایل‌های پروتکل اُپن‌فلو می‌پردازیم که شامل اطلاعات ارزشمندی از تعاملات بین سوئیچ‌ها و کنترلرها است. با تحلیل این لاگ فایل‌ها می‌توان اطلاعات مهمی مانند تغییرات در جریان داده‌ها، پیکربندی<sup>۱</sup> سوئیچ‌ها و وضعیت توپولوژی شبکه را استخراج کرد.

در نهایت به جزئیات الگوریتم مورد استفاده برای استخراج توپولوژی از لاگ فایل‌های اُپن‌فلو خواهیم پرداخت. این الگوریتم قادر است از داده‌های موجود در لاگ فایل‌ها، توپولوژی دقیق شبکه و چگونگی اتصال گره‌های مختلف را استخراج کند. همچنین در فصل ۵ مشاهده می‌کنید که استخراج توپولوژی برای تحلیل و توصیف شبکه و همچنین پیش‌بینی خطا در شبکه، اهمیت زیادی دارد.

## ۴-۲- لاگ اُپن‌فلو

در این بخش، به بررسی لاگ فایل اُپن‌فلو می‌پردازیم. همانطور که در بخش ۲-۲- مشاهده کردید، اُپن‌فلو یکی از اولین و پرکاربردترین پروتکل‌ها در شبکه‌های نرم‌افزار محور است که به عنوان یک ای‌پی‌آی جنوبی عمل می‌کند. در معماری شبکه‌های نرم‌افزار محور، ای‌پی‌آی جنوبی رابطی است که بین لایه کنترل و لایه داده‌ای شبکه ارتباط برقرار می‌کند و به کنترلر اجازه می‌دهد تا با دستگاه‌های شبکه مانند سوئیچ‌ها و روترها تعامل داشته باشد. به بیان دیگر، اُپن‌فلو واسطی برای ارتباط بین کنترلر و تجهیزات شبکه است که به کنترلر اجازه می‌دهد جداول جریان را به صورت پویا تغییر دهد و مدیریت ترافیک شبکه را از راه دور انجام دهد. این پروتکل به کنترلر این امکان را می‌دهد تا سیاست‌های مسیریابی و امنیتی شبکه را اعمال کرده و بر اساس نیازهای تغییرپذیر شبکه، رفتار دستگاه‌های فیزیکی را تنظیم کند (Gember-Jacobson et al., 2014; McKeown et al., 2008).

فایل‌های لاگ، جزئیات دقیقی از تعاملات بین سوئیچ‌ها و کنترلر را ثبت می‌کنند و به‌ویژه تغییرات پویا در جداول جریان را که در شبکه‌های نرم‌افزار محور رخ می‌دهند، مستند می‌سازند. با توجه به ماهیت این تغییرات پویا و ارتباط بین لایه کنترل و داده‌ای، فرض شده است که فایل لاگ ما فقط شامل بسته‌های اُپن‌فلو است. با تحلیل این فایل لاگ،

<sup>۱</sup> Reconfiguration

می‌توانیم رفتار شبکه را استنتاج کنیم و به درک بهتری از ساختار و الگوهای عملیاتی آن برسیم، در ادامه به ساختار و محتوای فایل‌های لاگ آپن‌فلو می‌پردازیم و توضیح می‌دهیم که چگونه رویدادهای شبکه در این فایل‌ها ثبت می‌شوند.

از میان انواع مختلف بسته‌های آپن‌فلو، پیام‌های Packet\_In، Flow\_Mod و Packet\_Out تمرکز اصلی تحلیل ما هستند. این پیام‌ها به عنوان پل ارتباطی بین سوئیچ‌ها و کنترلر عمل می‌کنند و امکان مدیریت ترافیک شبکه را از طریق منطق متمرکز کنترلر فراهم می‌آورند (The Open Networking Foundation, 2012).

#### ۴-۲-۱- پیام‌های Packet\_In

این پیام‌ها یکی از انواع اصلی بسته‌های آپن‌فلو هستند که از سوئیچ به کنترلر ارسال می‌شوند وقتی که سوئیچ بسته‌ای دریافت می‌کند که با هیچ‌یک از ورودی‌های جدول جریان آن مطابقت ندارد یا زمانی که بسته به دلیل تطابق با ورودی‌های خاص به کنترلر ارسال می‌شود. این پیام‌ها حاوی بخشی از بسته دریافتی و اطلاعات متادیتا مانند پورت ورودی و جزئیات دیگر هستند. پیام‌های Packet\_In به کنترلر اجازه می‌دهند تا درباره بسته‌هایی که در جدول جریان سوئیچ مشخص نشده‌اند، تصمیم‌گیری کند.

در مثال (۴-۳)، پیام Packet\_In بین سوئیچ (S1) و کنترلر (C) در محیط شبکه تعامل برقرار می‌کند. زمانی که سوئیچ بسته‌ای را بر روی پورت ۵ دریافت می‌کند اما ورودی متناظر در جدول جریان خود ندارد، یک پیام Packet\_In تولید کرده و آن را برای پردازش بیشتر به کنترلر ارسال می‌کند. این عمل باعث می‌شود کنترلر بسته را تحلیل کرده و احتمالاً جدول جریان را با قوانین جدید به‌روزرسانی کند. فایل لاگ متناظر با این پیام در شکل (۴-۱) ارائه شده است.

```
Type: OFPT_PACKET_IN (10)
switch: S1
controller: C
In port: 5
Reason: No matching flow (table-miss flow entry)
eth.dst: aa:a4:91:a5:71:28
eth.src: 7e:c8:93:0d:7c:71
ip.src: 10.0.0.2
ip.dst: 10.0.0.3
```

شکل (۴-۱): نمونه لاگ فایل Packet\_In برای مثال (۴-۳)

#### ۴-۲-۲- پیام‌های Flow\_Mod

این پیام‌ها توسط کنترلر به سوئیچ ارسال می‌شوند و به تغییراتی که در جدول جریان سوئیچ لازم است پاسخ می‌دهند. پیام‌های Flow\_Mod به کنترلر اجازه می‌دهند تا ورودی‌های جدول جریان را به‌صورت پویا تغییر دهد و نحوه پردازش بسته‌های آینده را تعیین کند.

در مثال (۴-۳)، پیام Flow\_Mod فرآیند به‌روزرسانی پویا جدول جریان سوئیچ (S1) توسط کنترلر (C) را تسهیل می‌کند. پس از تحلیل پیام Packet\_In، کنترلر ممکن است نیاز به نصب یک قانون جدید جریان را تشخیص دهد. در این حالت، کنترلر یک پیام Flow\_Mod به سوئیچ ارسال می‌کند و به آن دستور می‌دهد که بسته‌های دریافت‌شده بر روی پورت ۵ را هدایت کند. فایل لاگ متناظر با این پیام Flow\_Mod در شکل (۴-۲) ارائه شده است.

```
Type: OFPT_FLOW_MOD (14)
switch: S1
controller: C
In port: 5
openflow.eth_src: 7e:c8:93:0d:7c:71
openflow.eth_dst: aa:a4:91:a5:71:28
openflow.ofp_match.source_addr: 10.0.0.2
openflow.ofp_match.dest_addr: 10.0.0.3
```

شکل (۴-۲): نمونه لاگ فایل Flow\_Mod برای مثال (۴-۳)

#### ۴-۲-۳- پیام‌های Packet\_Out

این پیام‌ها توسط کنترلر به سوئیچ ارسال می‌شوند تا به سوئیچ دستور دهند یک بسته خاص را از یک پورت تعیین‌شده ارسال کند. این پیام‌ها امکان کنترل دقیق بر هدایت بسته‌ها را به کنترلر می‌دهند.

```
Type: OFPT_PACKET_OUT (13)
switch: S1
controller: C
In port: 5
Actions type: Output to switch port (0)
Output port: 6
```

شکل (۴-۳): نمونه لاگ فایل Packet\_Out برای مثال (۴-۳)

در مثال (۴-۳)، پس از دریافت پیام Packet\_In و پردازش بسته، کنترلر ممکن است تصمیم بگیرد بسته را فوراً ارسال کند. این کار را با ارسال یک پیام Packet\_Out به سوئیچ انجام می‌دهد و اقدام خروجی مانند ارسال بسته



از پورت ۵ به پورت ۶ را مشخص می‌کند. فایل لاگ متناظر با این پیام Packet\_Out در شکل (۴-۳) ارائه شده است

### ۴-۳- استخراج توپولوژی

برای استخراج توپولوژی شبکه از فایل‌های لاگ آپن‌فلو، ما از یک رویکرد سیستماتیک استفاده می‌کنیم که پیکربندی‌های پویا ثبت شده در این لاگ‌ها را تحلیل می‌کند. با فرض اینکه فایل لاگ فقط شامل پیام‌های آپن‌فلو مانند Packet\_In، Flow\_Mod، Packet\_Out باشد، تحلیل ما کاملاً بر اساس تعاملات بین کنترلر و سوئیچ‌ها است. از آنجا که هیچ دانش قبلی از رفتار شبکه در نظر گرفته نمی‌شود، این روش به ما اجازه می‌دهد تا توپولوژی شبکه را بر اساس ارتباطات و به‌روزرسانی‌های جریان که توسط کنترلر به صورت لحظه‌ای انجام می‌شود، استنتاج کنیم. همچنین، این پیام‌های آپن‌فلو شامل جزئیات حیاتی مانند میزبان مبدأ، میزبان مقصد و نام سوئیچ هستند که همگی برای نقشه‌برداری ساختار شبکه ضروری‌اند. با ذخیره هر دستگاه و ارتباطات مربوطه به عنوان گره‌ها و یال‌ها در یک گراف جهت‌دار، می‌توانیم روابط و مسیرهای بین عناصر مختلف شبکه را به‌طور دقیق نمایش دهیم. سپس، شماره‌های پورت به یال‌های خروجی هر گره اختصاص داده می‌شوند که نشان‌دهنده رابط‌های فیزیکی است که دستگاه‌ها از طریق آنها به هم متصل شده‌اند.

در الگوریتم استخراج توپولوژی شکل (۴-۴)، مراحل استخراج توپولوژی شبکه از فایل‌های لاگ با ساخت یک گراف جهت‌دار شرح داده شده است. برای توضیح جزئی‌تر، این الگوریتم به بررسی هر پیام Packet\_In در فایل لاگ می‌پردازد. برای هر پیام Packet\_In، بررسی می‌شود که آیا میزبان مبدأ، میزبان مقصد یا سوئیچ موجود در بسته قبلاً به عنوان گره‌ای در گراف G نمایان شده است یا خیر. اگر هر یک از این عناصر در گراف وجود نداشته باشد، الگوریتم گره متناظر را به گراف اضافه می‌کند. سپس، بر اساس تعاملات ثبت‌شده در پیام‌های Packet\_In، یال‌های مورد نیاز گراف تعریف می‌شوند. این یال‌ها ارتباطات بین میزبان‌ها و سوئیچ‌ها را نشان می‌دهند و چگونگی جریان ترافیک در شبکه را بازنمایی می‌کنند. با اضافه کردن سیستماتیک گره‌ها و یال‌ها، الگوریتم به تدریج یک توپولوژی جامع ایجاد می‌کند که ساختار و الگوهای عملیاتی شبکه را بر اساس فایل لاگ منعکس می‌کند. این رویکرد تضمین می‌کند که هر عنصر مرتبط با ارسال بسته‌ها به طور دقیق در گراف نمایان می‌شود و یک دید واضح از توپولوژی شبکه به دست می‌دهد.

در الگوریتم تخصیص پورت شکل (۴-۵)، فرآیند اختصاص دادن شماره پورت به یال‌های خروجی هر گره سوئیچ در گراف شبکه G شرح داده شده است. ابتدا، یک دیکشنری خالی تعریف می‌شود تا نگاشت بین هر جفت گره‌های متصل (یعنی یال‌های گراف) و شماره پورت‌های مربوطه ذخیره شود. علاوه بر این، یک متغیر p با مقدار ۱ مقداردهی اولیه می‌شود که برای اختصاص تدریجی شماره پورت استفاده خواهد شد. برای هر گره سوئیچ در گراف G، الگوریتم از میان گره‌های همسایه آن (یعنی گره‌هایی که مستقیماً به آن از طریق یک یال متصل هستند) تکرار می‌کند. برای هر همسایه، یک تابل به شکل (گره، همسایه) ایجاد می‌شود تا یال بین این دو گره را نمایش دهد. سپس این تابل به شماره

پورت در دیکشنری اختصاص داده می‌شود. پس از اختصاص یک شماره پورت به یال جاری،  $p$  به اندازه ۱ افزایش می‌یابد و اطمینان حاصل می‌شود که هر همسایه بعدی یک شماره پورت منحصر به فرد دارد. این فرآیند ادامه می‌یابد تا همه همسایگان تمام گره‌های سوئیچ دارای شماره پورت شوند و تخصیص پورت برای کل شبکه کامل شود.

با ایجاد گراف توپولوژی و تخصیص پورت‌ها، می‌توانیم توپولوژی شبکه را به صورت یک رشته نمایشی تعریف کنیم، که مطابق با تعریف ارائه‌شده در چارچوب نتکت است (Anderson et al., 2014). در مثال ما در شکل (۳-۴)، توپولوژی شبکه را می‌توان به طور مؤثر در چارچوب داینکت مدل‌سازی کرد. بر اساس فایل‌های لاگ ارائه‌شده و رفتار شبکه مرتبط (۱-۴)، (۲-۴) و (۳-۴)، توپولوژی برای این مثال به طور رسمی همان‌طور که در عبارت ۴-۱ نشان داده شده است، تعریف می‌شود.

$$\begin{aligned}
 T = & (pt = 1 . pt \leftarrow 4) + (pt = 4 . pt \leftarrow 1) \\
 & (pt = 2 . pt \leftarrow 5) + (pt = 5 . pt \leftarrow 2) \\
 & (pt = 6 . pt \leftarrow 3) + (pt = 3 . pt \leftarrow 6)
 \end{aligned}
 \tag{۱-۴}$$

**Algorithm 1** Topology Extraction**Require:** *Log File L*


---

```

1:  $G \leftarrow$  Empty directed graph  $G$ 
2:  $Prev\_SrcHost \leftarrow \emptyset$ 
3:  $Prev\_DstHost \leftarrow \emptyset$ 
4:  $Prev\_SwitchName \leftarrow \emptyset$ 
5:  $Flag \leftarrow False$ 
6: for each  $Packet\_In$  message( $m$ ) in the  $L$  do
7:    $SrcHost \leftarrow$  Source host in the  $m$ 
8:    $DstHost \leftarrow$  Destination host in the  $m$ 
9:    $SwitchName \leftarrow$  switch name in the  $m$ 
10:  if  $SrcHost$  is not in  $G$  then
11:    Add node  $SrcHost$  to  $G$ 
12:  if  $DstHost$  is not in  $G$  then
13:    Add node  $DstHost$  to  $G$ 
14:  if  $SwitchName$  is not in  $G$  then
15:    Add node  $SwitchName$  to  $G$ 
16:  if  $Prev\_DstHost == DstHost$  and  $Flag$  then
17:    Add edge  $Prev\_SwitchName \rightarrow Prev\_DstHost$  to  $G$ 
18:     $Flag = False$ 
19:  if  $Prev\_SrcHost != SrcHost$  then
20:    Add edge  $SrcHost \rightarrow SwitchName$  to  $G$ 
21:     $Flag = True$ 
22:  else if  $Prev\_SwitchName == SwitchName$  then
23:    Add edge  $Prev\_SwitchName \rightarrow SwitchName$  to  $G$ 
24:     $Prev\_SrcHost \leftarrow SrcHost$ 
25:     $Prev\_DstHost \leftarrow DstHost$ 
26:     $Prev\_SwitchName \leftarrow SwitchName$ 
27: return Topology  $G$ 

```

---

شکل (۴-۴): الگوریتم استخراج توپولوژی از لاگ فایل

**Algorithm 2** Port Allocation**Require:** Topology  $G$ 


---

```

1:  $PortMap \leftarrow$  Empty dictionary
2:  $p = 1$ 
3: for node in  $G$  do
4:   if node.type == "switch" then
5:     for n in node.neighbors do
6:        $PortMap[(node,n)] \leftarrow p$ 
7:        $p \leftarrow p+1$ 
8: return  $PortMap$ 

```

---

شکل (۴-۵): الگوریتم تخصیص پورت

## فصل ۵: استخراج قوانین DyNetKAT

---

## ۵-۱- مقدمه

هدف از این فصل، توضیح چگونگی استخراج و تعریف قوانین و روابط داینتکت از فایل‌های لاگ آپن‌فلو است. این قوانین شامل جریان‌های داده‌ای، به‌روزرسانی جداول جریان و تعاملات بین سوئیچ‌ها و کنترلر می‌شود.

در این فصل، ابتدا به نمای کلی توصیف شبکه‌های نرم‌افزار محور توسط داینتکت می‌پردازیم، سپس الگوریتم‌هایی را که برای استخراج قوانین داینتکت از این لاگ‌ها به کار گرفته می‌شوند، معرفی خواهیم کرد. این الگوریتم‌ها توسط آقای دکتر حسین حجت<sup>۱</sup> و خانم دکتر جورجیانا کلتیس<sup>۲</sup> نوشته شده‌اند و نقطه عطف این پژوهش به حساب می‌آیند. این الگوریتم‌ها به ما کمک می‌کند تا تصویری دقیق و جامع از رفتار شبکه را به‌صورت رسمی در چارچوب داینتکت ارائه دهیم.

## ۵-۲- نمای کلی توصیف شبکه

در داینتکت، رفتارهای شبکه نرم‌افزار محور از طریق عبارات ترکیبی که شامل جداول جریان، توپولوژی‌ها و اقدامات کنترلر هستند، ثبت می‌شوند. رفتار کلی شبکه توسط یک عبارت ترکیبی که ارتباطات بین سوئیچ‌ها و کنترلر را شامل می‌شود، توصیف می‌شود. این چارچوب اطمینان حاصل می‌کند که تغییرات در وضعیت شبکه، مانند به‌روزرسانی‌های جداول جریان، تنها برای بسته‌های جدید قابل مشاهده هستند و یکپارچگی بسته‌های در جریان حفظ می‌شود. این قابلیت برای مدل‌سازی و تأیید رفتارهای لحظه‌ای شبکه‌های نرم‌افزار محور بسیار حیاتی است. رفتار کلی شبکه به صورت عبارت ۵-۱ بیان می‌شود.

$$SDN \triangleq D_{X_1, \dots, X_m} \parallel C, \quad (۵-۱)$$

که در آن  $D_{X_1, \dots, X_m}$  عملیات لایه داده را نشان می‌دهد که شامل  $m$  سوئیچ است و در قالب جداول جریان مربوطه  $X_1, \dots, X_m$  کدگذاری شده‌اند. همچنین،  $C$  به کنترلر اشاره دارد. این ساختار موازی، ارتباط همزمان بین لایه داده و کنترلر را بازتاب می‌دهد و اطمینان می‌دهد که به‌روزرسانی‌های وضعیت شبکه، مانند تغییرات در جداول جریان، به‌صورت منسجم اجرا و اعمال می‌شوند. همچنین، عبارت لایه داده به صورت عبارت ۵-۲ نوشته می‌شود.

<sup>۱</sup> Dr. Hossein Hojjat

<sup>۲</sup> Dr. Georgiana Caltais

$$D_{X_1, \dots, X_m} \triangleq ((X_1 + \dots + X_m) \cdot T)^*; D_{X_1, \dots, X_m} \oplus \sum_{X'_i \in FT}^{\oplus} m_i ? X'_i; D_{X_1, \dots, X'_i, \dots, X_m} \quad (2-5)$$

عبارت ۲-۵، نشان‌دهنده عملیات کلی لایه داده و تعامل آن با توپولوژی شبکه  $T$  است. در این حالت سوئیچ‌ها در ارتباط با توپولوژی رفتار خود را تکرار می‌کنند و پیام‌های شبکه را در طول شبکه انتقال می‌دهند. رفتار یک سوئیچ (جدول جریان  $X_i$ ) می‌تواند با دریافت یک پیام از کنترلر در کانال  $m_i$  می‌تواند به  $X'_i$  تغییر کند و در این صورت رفتار لایه داده به  $D_{X_1, \dots, X'_i, \dots, X_m}$  به‌روزرسانی خواهد شد.

### ۳-۵ - قوانین استخراج

در این بخش، یک روش‌شناسی که توسط آقای دکتر حسین حجت و خانم دکتر جورجیانا کلتیس طراحی شده‌اند، برای استخراج قوانین توصیفی داینکتک از مجموعه داده‌های لاگ شبکه نرم‌افزار محور، با تمرکز بر پیام‌های `Packet_In`، `Flow_Mod` و `Packet_Out` ارائه می‌دهیم. با تحلیل سیستماتیک پیام‌های شبکه، می‌توانیم مشخصات داینکتک را که تعامل بین لایه داده و کنترلر را به‌طور دقیق مدل می‌کند، مطابق با عبارات ۱-۵ و ۲-۵ بسازیم. ابتدا، چند فرض کلیدی برای روش‌شناسی بیان می‌کنیم. اول، فایل لاگ آپن‌فلو با استفاده از ابزار وایرشارک<sup>۱</sup> استخراج می‌شود که به‌عنوان ورودی ابزار ما عمل می‌کند. دوم، در مرحله پیش‌پردازش، توپولوژی شبکه مطابق با روشی که در الگوریتم (۴-۴) شرح داده شد، استخراج می‌شود. همچنین فرض بر این است که در ابتدا هیچ قانونی در جداول جریان روی سوئیچ‌ها نصب نشده و رفتار کنترلر در ابتدا برابر با  $\perp$  است. علاوه بر این، فرض می‌شود که لایه کنترل و لایه داده به‌طور هم‌زمان ارتباط برقرار می‌کنند. در نتیجه، جمع سمت راست  $\oplus$  در عبارت ۲-۵، جدول جریان سوئیچ  $X_i$  را تنها زمانی به‌روز می‌کند که کنترلر از طریق یک پیام ارسال‌شده در کانال  $m_i$  آن را آغاز کند.

همان‌طور که در بخش ۲-۴ توضیح داده شد، سه نوع اصلی از پیام‌ها بین سوئیچ‌ها و کنترلر مبادله می‌شوند. در ادامه، کدگذاری‌های داینکتک مربوط به هر یک از این نوع پیام‌ها را تعریف خواهیم کرد.

### ۳-۵-۱ - قوانین پیام `Packet_In(sw, mid, omids)`

این پیام با شناسه `mid` نشان می‌دهد که سوئیچ `sw` پیام‌هایی در کانال‌های `omids` را به کنترلر ارسال کرده است. کدگذاری قوانین داینکتک مربوط به این پیام به صورت عبارت ۳-۵ است.

<sup>۱</sup> Wireshark

$$D_{X_1, \dots, X_m} \oplus = \sum_{omid \in omids}^{\oplus} omid!1; D_{X_1, \dots, X_m} \quad (۳-۵)$$

عبارت ۳-۵ بیانگر آن است که سوئیچ به درستی با کنترلر ارتباط برقرار کرده است و به آن در کانال *omid* پیام ارسال کرده است.

### ۵-۳-۲ - قوانین پیام $\text{Flow\_Mod}(sw, mid', omids')$

این پیام نشان می‌دهد که کنترلر پیام با نشانه  $mid'$  را از سوئیچ  $sw$  دریافت کرده است و پیام‌هایی با نشانه‌های  $omids'$  ارسال کرده است. کدگذاری قوانین داینکتک مربوط به این پیام به صورت عبارت ۴-۵ می‌باشد.

$$C \oplus = mid'?1; \left( \sum_{omid_{sw}^i \in omids'}^{\oplus} omid_{sw}^i!ftmid_{sw}^i; C \right) \quad (۴-۵)$$

در عبارت ۴-۵،  $ftmid_{sw}^i$  توسط کنترلر محاسبه می‌شود و تضمین می‌کند که ارتباط مابین کنترلر و لایه داده برقرار شده است و میتواند تغییرات پویا در جداول جریان سوئیچ اعمال شود.

### ۵-۳-۳ - قوانین پیام $\text{Packet\_Out}(sw, mid, mtype, ops)$

این پیام نشان می‌دهد که سوئیچ  $sw$  پیام با نشانه  $mid$  و از نوع  $mtype$  را دریافت و پردازش کرده است. کدگذاری قوانین داینکتک مربوط به این پیام به صورت عبارت ۵-۵ می‌باشد.

$$D_{X_1, \dots, X_i, \dots, X_m} \oplus = mid?X_i'; D_{X_1, \dots, X_i', \dots, X_m} \quad (۵-۵)$$

عبارت ۵-۵ بیانگر تغییر جدول جریان  $X_i$  به  $X_i' = ops$  است و مشخص می‌کند پیکربندی جدید به صورت پویا بر روی لایه داده اعمال شده است.

## ۴-۵- مثال

همانطور که قبلاً اشاره کردیم، برای نمایش اصول و روش‌های مطرح شده در این پژوهش، از یک توپولوژی ساده شبکه به عنوان مثال اجرایی در بخش ۳-۵-۲ استفاده می‌کنیم. همچنین فرض کنید که شکل‌های (۱-۴)، (۲-۴) و (۳-۴)، به عنوان مرجع لاگ فایل برای بررسی و تحلیل است.

رفتار شبکه نرم‌افزارمحور به‌عنوان ترکیب موازی لایه داده  $D_{X_{S1}}$  و کنترلر  $C$  توصیف می‌شود که با معادله ۵-۶ بیان شده است. فرض می‌کنیم که قانونی از پیش تعریف‌شده، همان‌طور که در عبارت ۳-۱ مشخص شده است، در سوئیچ  $S1$  وجود دارد  $(6 \leftarrow pt \cdot pt = 4 \cdot X_{S1})$ . علاوه بر این، رفتار کنترلر هنوز مشخص نیست و به‌صورت  $C \triangleq \perp$  تعریف می‌شود. توپولوژی  $T$ ، همان‌طور که در عبارت ۴-۱ تعریف شده است، طبق روش شرح داده‌شده در بخش ۴-۳-۳ استخراج شده است.

$$SDN \triangleq D_{X_{S1}} \parallel C \quad (۵-۶)$$

$$D_{X_{S1}} \triangleq ((X_{S1}) \cdot T)^*; D_{X_{S1}}$$

همانطور که در عبارت ۵-۷ قابل مشاهده است، هنگامی که یک پیام `Packet_In` مانند شکل (۴-۱) مشاهده می‌شود، لایه داده با افزودن یک عبارت طبق قانون ۵-۳ به‌روزرسانی می‌شود.

$$D_{X_{S1}} \triangleq ((X_{S1}) \cdot T)^*; D_{X_{S1}} \oplus ch!1; D_{X_{S1}} \quad (۵-۷)$$

سپس، همان‌طور که در معادله ۵-۸ نشان داده شده است، کنترلر  $C$  اصطلاحات خود را بر اساس قانون ۵-۴ و پیام (۴-۲) به‌روزرسانی می‌کند.

$$C \triangleq (ch?1; (ch!(pt = 5 \cdot pt \leftarrow 6); C)) \quad (۵-۸)$$

در نهایت، پس از پیام `Packet_Out`، شکل (۴-۳)، لایه داده و جدول جریان سوئیچ طبق قانون ۵-۵ به‌روزرسانی می‌شود، و به عبارت ۵-۹ منجر خواهد شد. این عبارت به درستی نشان می‌دهد که یک پیکربندی پویا<sup>۱</sup> توسط کنترلر به

<sup>۱</sup> Dynamic Reconfiguration



درستی در لایه داده اعمال می‌شود و باعث اضافی شدن جریان جدید ( $X'_{S1} \triangleq pt = 5.pt \leftarrow 6$ ) در سوئیچ  $X_{S1}$  خواهد شد.

$$X'_{S1} \triangleq (pt = 5.pt \leftarrow 6)$$

$$\begin{aligned} D_{X_{S1}} &\triangleq ((X_{S1}) \cdot T)^*; D_{X_{S1}} \oplus \\ &\quad ch!1; D_{X_{S1}} \oplus \\ &\quad ch'?X'_{S1}; D_{X'_{S1}} \end{aligned} \quad (۹-۵)$$

$$\begin{aligned} D_{X'_{S1}} &\triangleq ((X'_{S1}) \cdot T)^*; D_{X'_{S1}} \oplus \\ &\quad ch!1; D_{X'_{S1}} \oplus \\ &\quad ch'?X'_{S1}; D_{X'_{S1}} \end{aligned}$$

در نتیجه، این مثال به صورت گام‌به‌گام فرآیند استخراج عبارات داینکت، به صورت سیستماتیک از روی لاگ فایل و رویدادهای شبکه را نشان داد و رفتار شبکه به درستی در داینکت مدل کرد.

## فصل ۶: پیاده‌سازی و آزمایش‌ها

---

## ۱-۶- پیاده‌سازی

در این فصل، به پیاده‌سازی ابزار و آزمایش‌های طراحی شده به هدف پاسخ‌گویی به سوالات تحقیق می‌پردازیم. این ابزار، FPSDN، با استفاده از زبان برنامه‌نویسی پایتون پیاده‌سازی شده است. به صورت کلی، ورودی این ابزار لاگ فایل شبکه‌های نرم‌افزار محور است و قادر است داده‌های لاگ را پردازش کند، توپولوژی شبکه را استخراج کند و رفتار شبکه را در قالب قوانین رسمی داینکتک مدل‌سازی نماید. در نهایت، ابزار قادر است برای ویژگی امنیتی تعریف شده، وجود ترتیب خاصی از آپدیت‌های سویچ‌ها که منجر به خطا می‌شود را بررسی کند و در صورت امکان وجود خطا، آن را پیش‌بینی می‌کند. در ادامه، مراحل کلیدی پیاده‌سازی توضیح داده می‌شود.

### ۱-۱-۶- ورودی ابزار

ورودی ابزار، یک فایل لاگ واقعی شبکه است که شامل داده‌های ترافیک ضبط‌شده در پروتکل اُپن‌فلو می‌باشد. برای

این فایل‌ها باید دارای فرمت pcapng باشند. ابزار با فرض ساختار استاندارد این فایل‌ها، اطلاعات مورد نیاز را برای مراحل بعدی پردازش استخراج می‌کند.

### ۲-۱-۶- مرحله پیش‌پردازش

در این مرحله، ابزار با استفاده از بهینه‌سازی و فیلترگذاری، بسته‌های غیرضروری و تکراری را حذف کرده و فقط اطلاعات مهم را برای تحلیل انتخاب می‌کند. این فرآیند نه تنها حجم داده‌ها را کاهش می‌دهد بلکه امکان تحلیل سریع‌تر و دقیق‌تر را فراهم می‌کند. همچنین، در این مرحله رویدادهای شبکه گروه‌بندی می‌شوند؛ هر رویداد مجموعه‌ای از بسته‌ها را شامل می‌شود که یک مسیر مشخص را در شبکه نشان می‌دهند. این دسته‌بندی به ابزار کمک می‌کند تا رفتار جریان داده‌ها را بهتر شناسایی کرده و برای مدل‌سازی استفاده کند.

### ۶-۱-۳- مرحله یادگیری توپولوژی شبکه

برای شناسایی توپولوژی شبکه، بسته‌های لاگ بررسی شده‌اند و با استفاده از الگوریتم (۴-۴)، گره‌ها (سوئیچ‌ها و میزبان‌ها) و ارتباطات بین آن‌ها را شناسایی می‌شوند و توپولوژی شبکه در قالب گراف جهت‌دار مدل‌سازی می‌شود. این گراف با استفاده از کتابخانه NetworkX ایجاد شده و قابلیت نمایش بصری را نیز دارد. در این مرحله، تمام گره‌ها به همراه نوع (سوئیچ یا میزبان) و مشخصات ارتباطات ذخیره می‌شوند که به‌طور دقیق ساختار و توپولوژی شبکه را شبیه‌سازی می‌کند.

### ۶-۱-۴- مرحله یادگیری مدل رفتاری شبکه

در این مرحله قوانین مطرح شده توسط آقای دکتر حجت و خانم دکتر جورجیانا کلتیس در بخش ۵ پیاده‌سازی شده است. در این مرحله، تمامی تعاملات و ارتباطات شبکه در قالب مدل رسمی داینکت ارائه می‌شوند. در این مرحله تمامی پیکربندی‌ها و به‌روزرسانی رفتار سوئیچ‌ها نیز به دست می‌آیند.

### ۶-۱-۵- مرحله بررسی پیکربندی‌های شبکه و ایجاد ویژگی‌های ایمنی مورد نیاز

در این مرحله، تنظیمات شبکه به زبان داینکت نوشته می‌شوند و ویژگی‌های امنیتی (بخش ۳-۴-۲-) مورد نیاز شبکه، مانند جلوگیری از دسترسی غیرمجاز تعریف می‌گردند. این ویژگی‌ها بر اساس نیازهای خاص شبکه و سناریوهای محتمل خطا طراحی می‌شوند. پس از ایجاد مدل رفتاری شبکه، تمامی حالت‌های پیکربندی‌های موجود و به‌روزرسانی‌های انجام‌شده در سوئیچ‌ها با ترتیب‌های متفاوت، به صورت خودکار بررسی می‌شوند.

### ۶-۱-۶- مرحله بررسی ویژگی‌های ایمنی

در این مرحله مدل رفتاری شبکه و ویژگی‌های امنیتی تعریف‌شده به ابزار داینکت ارسال می‌شوند تا بررسی دقیقی از صحت و عملکرد این ویژگی‌ها انجام شود. ابزار با اجرای تحلیل‌های دقیق و شبیه‌سازی‌های مرتبط، نتایج ویژگی‌های امنیتی را استخراج می‌کند. این مرحله به شناسایی نقض‌های احتمالی و ارزیابی کارایی ویژگی‌های امنیتی تعریف‌شده کمک می‌کند.

## ۶-۱-۷- خروجی ابزار

خروجی ابزار شامل مجموعه‌ای از اطلاعات کلیدی است که شامل رویدادهای شبکه، توپولوژی شبکه، مدل رفتاری شبکه، و نتایج بررسی ویژگی‌های امنیتی می‌باشد. این خروجی‌ها به صورت فایل‌های JSON ذخیره می‌شوند و برای استفاده در تحلیل‌های بیشتر یا بهبود پیکربندی‌های شبکه کاربرد دارند. همچنین، ابزار امکان ارائه نمایش‌های گرافیکی از توپولوژی و زمان‌های پردازش را فراهم می‌کند تا کاربران بتوانند تحلیل جامعی از شبکه خود به دست آورند.

## ۶-۲- آزمایش‌ها

برای انجام آزمایش‌های این پژوهش، به لاگ فایل‌های واقعی شبکه‌های نرم‌افزارمحور نیاز بود. با این حال، به دلیل عدم دسترسی به چنین لاگ فایل‌هایی از محیط‌های واقعی آپن‌فلو، تصمیم گرفته شد که شبکه‌های نرم‌افزارمحور شبیه‌سازی شوند. این شبیه‌سازی امکان بررسی دقیق‌تر سناریوها و استخراج داده‌های مورد نیاز را فراهم کرد.

در این پژوهش، از شبیه‌ساز مینینت<sup>۱</sup> برای ایجاد شبکه‌های نرم‌افزارمحور استفاده شد. مینینت ابزاری قدرتمند و رایج برای شبیه‌سازی شبکه‌های نرم‌افزار محور است که امکان تعریف توپولوژی‌های مختلف، افزودن سوئیچ‌ها، میزبان‌ها و ایجاد ترافیک بین گره‌ها را به صورت مجازی فراهم می‌کند. در این شبیه‌سازی‌ها، شبکه‌های ایجاد شده به کنترلر پاکس<sup>۲</sup> متصل شدند. پاکس یک کنترلر شبکه‌های نرم‌افزار محور متن‌باز و مبتنی بر پایتون است که قابلیت مدیریت و پیکربندی سوئیچ‌ها و جریان‌های داده را دارد.

برای ضبط تعاملات بین کنترلر و سوئیچ‌ها و استخراج لاگ فایل‌های شبکه، از ابزار وایرشارک<sup>۳</sup> استفاده شد. وایرشارک یک ابزار تحلیل پروتکل شبکه است که امکان مشاهده و ضبط بسته‌های داده را در زمان واقعی فراهم می‌کند. در این مرحله، داده‌های ترافیک شبکه که شامل پیام‌های آپن‌فلو بودند، ضبط و در قالب فایل‌های pcapng ذخیره شدند. این فایل‌ها به عنوان ورودی ابزار تحلیل رفتار شبکه استفاده شدند.

تمام مراحل شبیه‌سازی شبکه، اتصال به کنترلر، و استخراج لاگ فایل‌ها به طور دقیق مستندسازی شده و به همراه لاگ فایل‌های شبکه در مخزن گیت‌هاب به آدرس زیر در دسترس قرار دارند:

<https://github.com/mghobakhlou/FPSDN>

کاربران می‌توانند با مراجعه به این مخزن، پیاده‌سازی‌ها را مشاهده و برای ایجاد سناریوهای مشابه از آن استفاده

<sup>۱</sup> Mininet

<sup>۲</sup> POX

<sup>۳</sup> Wireshark

کنند.

با استفاده از این روش، لاگ فایل مربوط به هشت سناریوی مختلف از توپولوژی‌های گوناگون شبکه استخراج شد. این توپولوژی‌ها شامل ساختارهای رایج در شبکه‌های نرم‌افزار محور هستند که در شرایط مختلف آزمایش شده‌اند. در تمامی این آزمایش‌ها شبکه در ابتدا در وضعیت امن قرار دارد و بعد از پیکربندی‌ها و به‌روزرسانی‌های جدید نیز همانطور که در هر آزمایش اشاره خواهد شد، در وضعیت امن می‌باشد. در تعریف آزمایش‌ها، نوع ویژگی را به صورت  $Path(X,Y)$  تعریف می‌کنیم که به معنی قابل دسترسی بودن دستگاه  $Y$  از دستگاه  $X$  است و  $Add\_Rule(X, Y, Z)$  به معنی ایجاد یک به‌روزرسانی در سوییچ  $X$  است که باعث اتصال دستگاه  $Y$  به  $Z$  خواهد شد. همچنین! به معنای نقیض است، یعنی،  $!Path(X,Y)$  به معنی قابل دسترسی نبودن دستگاه  $Y$  از دستگاه  $X$  است.

جزئیات این سناریوها به شرح زیر است:

### ۶-۲-۱- آزمایش Star

در این آزمایش، توپولوژی ستاره‌ای (شکل (۶-۱)) با یک سوئیچ در مرکز و ۶ میزبان در اطراف آن پیاده‌سازی شده است. سناریو به این صورت است که در ابتدا میزبان ۱ فقط قادر به ارسال پیام به میزبان ۲ است و با ۴ به‌روزرسانی در جدول جریان سوئیچ ۷، میزبان ۱ قادر به ارسال پیام به مابقی میزبان‌ها خواهد بود. جزئیات این آزمایش به شرح زیر است:

وضعیت امن اولیه:

$Path(H1,H2)$

وضعیت امن ثانویه:

$Path(H1,H3)$

$Path(H1,H4)$

$Path(H1,H5)$

$Path(H1,H6)$

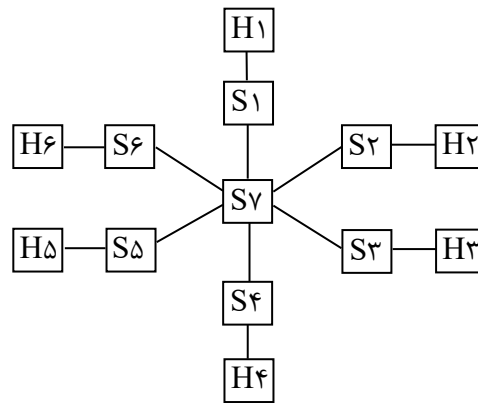
به‌روزرسانی‌های سوییچ‌ها به ترتیب:

$Add\_Rule(S7, S1, S3)$

$Add\_Rule(S7, S1, S4)$

$Add\_Rule(S7, S1, S5)$

$Add\_Rule(S7, S1, S6)$



شکل (۶-۱): توپولوژی آزمایش Star

### ۶-۲-۲- آزمایش Mesh

در این آزمایش، توپولوژی مش (شکل (۶-۲)) با ۶ سوئیچ مجزا و ۶ میزبان مجزا و چسبیده به هر یک از سوئیچ‌ها پیاده‌سازی شده است. این سناریو به این صورت که در ابتدا فقط میزبان شماره ۱ از سوئیچ شماره ۱ توانایی ارسال پیام به میزبان شماره ۲ از سوئیچ شماره ۲ را دارد و با ۴ به روزرسانی جدید در جدول جریان سوئیچ شماره ۱، میزبان ۱ قادر به ارسال پیام به مابقی میزبان‌ها را خواهد داشت. جزئیات این آزمایش به شرح زیر است:

وضعیت امن اولیه:

$\text{Path}(H1, H2)$

وضعیت امن ثانویه:

$\text{Path}(H1, H3)$

$\text{Path}(H1, H4)$

$\text{Path}(H1, H5)$

$\text{Path}(H1, H6)$

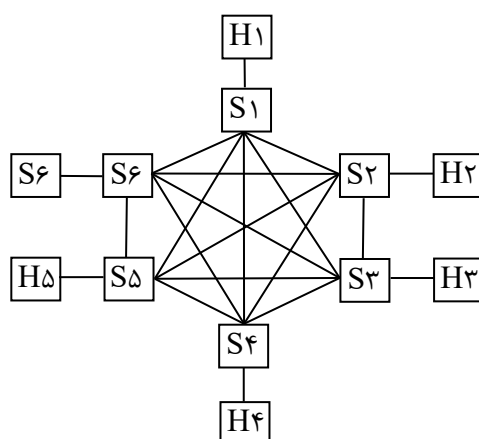
به روزرسانی‌های سوئیچ‌ها به ترتیب:

$\text{Add\_Rule}(S1, H1, S3)$

$\text{Add\_Rule}(S1, H1, S4)$

$\text{Add\_Rule}(S1, H1, S5)$

$\text{Add\_Rule}(S1, H1, S6)$



شکل (۶-۲): توپولوژی آزمایش Mesh

### ۶-۲-۳ - آزمایش Ring

در این آزمایش، توپولوژی رینگ (شکل (۶-۳)) با ۶ سوئیچ و ۶ میزبان متصل به هر کدام از سوئیچ‌ها پیاده‌سازی شده است. این سناریو به این صورت است که در ابتدا میزبان شماره ۱ قادر به ارسال پیام به میزبان شماره ۵ از سوئیچ شماره ۵ دارد و با ۲ به‌روزرسانی جدید در جداول جریان سوئیچ‌ها، میزبان ۳ توانایی ارسال پیام به میزبان ۶ را خواهد داشت. جزئیات این آزمایش به شرح زیر است:

وضعیت امن اولیه:

$\text{Path}(H1, H5)$

وضعیت امن ثانویه:

$\text{Path}(H3, H6)$

به روزرسانی‌های سوئیچ‌ها به ترتیب:

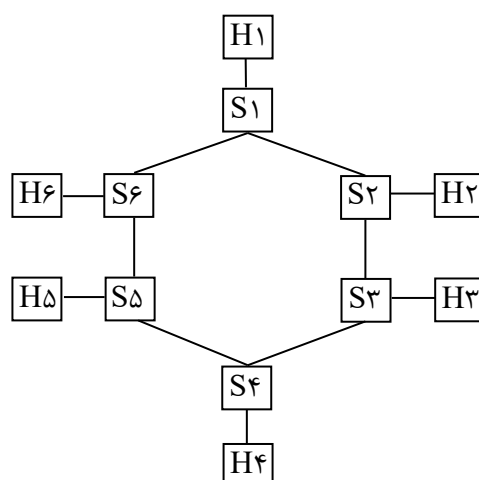
$\text{Add\_Rule}(S3, H3, S4)$

$\text{Add\_Rule}(S5, S4, S6)$

### ۶-۲-۴ - آزمایش Black Hole

در این آزمایش، توپولوژی خطی (شکل (۶-۴)) با ۱۰ سوئیچ و ۱۰ میزبان متناظر با هر سوئیچ پیاده‌سازی شده است. در این سناریو، در ابتدا میزبان شماره ۱ قادر به ارسال پیام به میزبان شماره ۴ دارد و با ۲ به‌روزرسانی جدید در جداول جریان سوئیچ‌ها، میزبان ۲ قادر به ارسال پیام به میزبان ۳ را خواهد داشت. جزئیات این آزمایش به شرح زیر است:





شکل (۶-۳): توپولوژی آزمایش Ring

وضعیت امن اولیه:

Path(H1,H4)

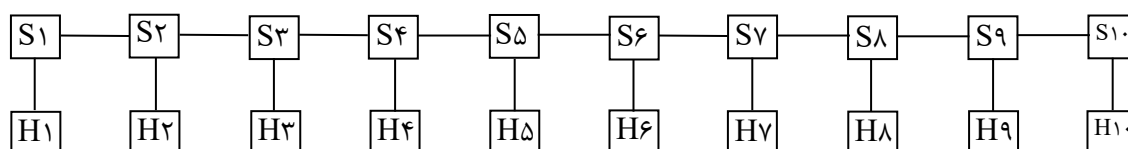
وضعیت امن ثانویه:

Path(H2,H3)

به روزرسانی‌های سوئیچ‌ها به ترتیب:

Add\_Rule(S2, S1, H2)

Add\_Rule(S3, S2, H3)



شکل (۶-۴): توپولوژی خطی

## ۶-۲-۵- آزمایش Race condition

در این آزمایش (شکل (۶-۵))، از ۶ سوئیچ و ۴ میزبان استفاده شده است و توپولوژی به کار رفته در آن از مقاله دایننتکت (Caltais et al., 2021) استفاده شده است. در این سناریو در ابتدا میزبان ۱ قادر به ارسال پیام به میزبان ۳ دارد و با ۲ به‌روزرسانی جدید در جداول جریان سوئیچ‌ها میزبان ۲ قادر به ارسال پیام به میزبان ۴ را خواهد داشت.

جزئیات این آزمایش به شرح زیر است:

وضعیت امن اولیه:

Path(H1,H3)

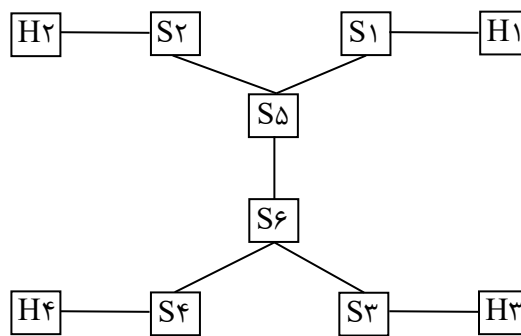
وضعیت امن ثانویه:

Path(H2,H4)

به روزرسانی‌های سویچ‌ها به ترتیب:

Add\_Rule(S5, S2, S6)

Add\_Rule(S6, S5, S4)



شکل (۵-۶): توپولوژی آزمایش Race Condition

## ۶-۲-۶ آزمایش Isolation

در این آزمایش، از توپولوژی خطی (شکل (۴-۶)) با ۶ سوئیچ و ۶ میزبان متناظر استفاده شده است. در این حالت فرض کنید که به صورت خطی هر ۲ سوئیچ و ۲ میزبان متناظر با آن را یک گروه می‌نامیم. در این سناریو، میزبان ۱ (از گروه ۱) قادر به ارسال پیام به میزبان شماره ۴ (از گروه ۲) دارد و بعد از ۲ به‌روزرسانی در جداول جریان سوئیچ‌ها، میزبان شماره ۳ (از گروه ۲) قادر به ارسال پیام به میزبان شماره ۶ (از گروه ۳) را خواهد داشت. جزئیات این آزمایش به شرح زیر است:

وضعیت امن اولیه:

Path(H1,H4)

وضعیت امن ثانویه:

Path(H3,H6)

به روزرسانی‌های سویچ‌ها به ترتیب:

Add\_Rule(S3, H3, S4)

```

graph TD
    H1[H1] --- S1[S1]
    H2[H2] --- S1
    S1 --- S2[S2]
    S2 --- S3[S3]
    S3 --- S4[S4]
    S4 --- S5[S5]
    S5 --- S6[S6]
    H3[H3] --- S3
    H4[H4] --- S4
    H5[H5] --- S5
    H5[H5] --- S6
  
```

شکل (۶-۶): تویولوژی آزمایش Isolation

در این آزمایش، توپولوژی خطی (شکل (۴-۶)) با ۱۰ سوئیچ و ۱۰ میزبان متناظر با هر سوئیچ پیاده‌سازی شده است. در این سناریو، در ابتدا میزبان شماره ۷ قادر به ارسال پیام به میزبان شماره ۱۰ دارد و با ۳ به‌روزرسانی جدید در جداول جریان سوئیچ‌ها، میزبان ۶ قادر به ارسال پیام به میزبان ۱۰ و میزبان شماره ۴ قادر به ارسال پیام به میزبان شماره ۸ را خواهد داشت. جزئیات این آزمایش به شرح زیر است:

Path(H4,H8)

Add Rule(S8, S7, H8)

## ۶-۲-۸- آزمایش Faulty Fattree

در این آزمایش، توپولوژی شکل (۶-۷) پیاده‌سازی شده است. در این آزمایش، در ابتدا میزبان شماره ۲ قادر به ارسال پیام به میزبان شماره ۸ را دارد و با ۳ به‌روزرسانی در جداول جریان سوئیچ‌ها، میزبان شماره ۵ به میزبان شماره ۷ و میزبان شماره ۱ به میزبان شماره ۸ می‌تواند پیام ارسال کند. جزئیات این آزمایش به شرح زیر است:

وضعیت امن اولیه:

Path(H2,H8)

وضعیت امن ثانویه:

Path(H5,H7)

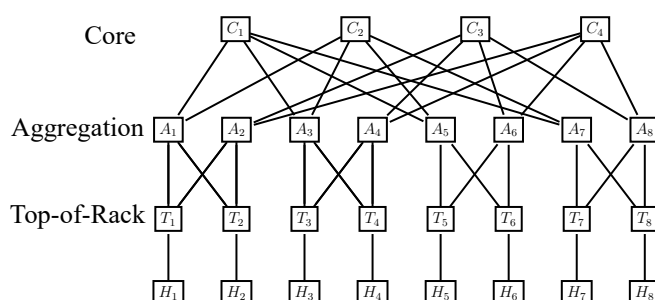
Path(H1,H8)

به روزرسانی‌های سوئیچ‌ها به ترتیب:

Add\_Rule(T1, H1, A1)

Add\_Rule(C1, A5, A7)

Add\_Rule(A7, C1, T7)



شکل (۶-۷): توپولوژی آزمایش FatTree

## ۳-۶- نصب و راه‌اندازی

شما می‌توانید ابزار را از لینک زیر دانلود و استفاده کنید.

<https://github.com/mghobakhlou/FPSDN>

این مخزن شامل تمام اسکریپت‌ها، فایل‌های داده و مستندات اضافی است که به شما کمک می‌کند محیط را تنظیم کرده، فایل‌های لاگ را پیش‌پردازش کنید و قوانین داینتکت را استخراج کنید و نتایج مورد بحث در این گزارش را تولید و تایید کنید.

همانطور که در مخزن گیت‌هاب نوشته شده است، بهترین و راحت‌ترین روش نصب ابزار بر روی سیستم‌عامل اوبونتو<sup>۱</sup> با ورژن حداکثری ۲۲ است. با این حال برای راحتی کاربران، ابزار با استفاده از داکر<sup>۲</sup> مجازی سازی شده است و می‌توانید مطابق با دستورات ارائه شده در مخزن گیت‌هاب، ابزار را بر روی هر سیستم‌عامل دلخواه با استفاده از داکر نصب و استفاده کنید.

---

<sup>۱</sup> Ubuntu

<sup>۲</sup> Docker

## فصل ۷: پیش‌بینی خطا و نتایج

---

## ۱-۷ ویژگی‌های ایمنی آزمایش‌ها

در این فصل به بررسی نتایج آزمایش‌های ارائه شده در فصل ۶ می‌پردازیم. همانطور که بیان شد، ورودی ابزار لاگ فایل شبکه‌ای است که از یک وضعیت امن با تعداد به روزرسانی جدید به وضعیت امن دیگر می‌رود. ترکیب و ترتیب هر کدام از این به‌روزرسانی‌ها می‌تواند وضعیت شبکه را به حالت نامطلوبی ببرد و منجر به خطا شود. در حال حاضر ابزار ما، به صورت کاملاً خودکار قادر به بررسی و پیدا کردن ترکیب و ترتیبی از به‌روزرسانی‌های شبکه که باعث خطا در شبکه می‌شود، را دارد.

### • آزمایش Star:

وضعیت امن اولیه: دسترسی‌پذیری میزبان شماره ۲ از میزبان شماره ۱ ( $Path(H1, H2)$ ).  
وضعیت امن ثانویه: دسترسی‌پذیری مابقی میزبان‌ها از میزبان ۱ بعد از به‌روزرسانی‌ها ( $Path(H1, H3)$  ،  $Path(H1, H4)$  ،  $Path(H1, H5)$  ،  $Path(H1, H6)$ ).  
تعریف خطا در این آزمایش (پیش‌بینی شده توسط چند به‌روزرسانی پویا):  
این آزمایش به هدف بررسی درستی ابزار، حالات امن اولیه و ثانویه ایجاد شده است و خطایی در این آزمایش تعریف نشده است.

### • آزمایش Mesh:

وضعیت امن اولیه: دسترسی‌پذیری میزبان شماره ۲ از میزبان شماره ۱ ( $Path(H1, H2)$ ).  
وضعیت امن ثانویه بعد از به‌روزرسانی‌ها: دسترسی‌پذیری مابقی میزبان‌ها بعد از به‌روزرسانی‌ها ( $Path(H1, H3)$  ،  $Path(H1, H4)$  ،  $Path(H1, H5)$  و  $Path(H1, H6)$ ).  
تعریف ویژگی‌های ایمنی در این آزمایش (پیش‌بینی شده توسط چند به‌روزرسانی پویا):  
این آزمایش به هدف بررسی درستی ابزار، حالات امن اولیه و ثانویه ایجاد شده است و خطایی در این آزمایش تعریف نشده است.

### • آزمایش Ring:

وضعیت امن اولیه: دسترسی‌پذیری میزبان شماره ۵ از میزبان شماره ۱ ( $Path(H1, H5)$ ).  
وضعیت امن ثانویه بعد از به‌روزرسانی‌ها: دسترسی‌پذیری میزبان شماره ۶ از میزبان شماره ۳ ( $Path(H3, H6)$ ).  
تعریف ویژگی‌های ایمنی در این آزمایش (خطای پیش‌بینی شده توسط چند به‌روزرسانی پویا):  
عدم دسترسی‌پذیری میزبان شماره ۵ از میزبان شماره ۳ ( $Path(H3, H5)$ ).

### • آزمایش Black Hole:

وضعیت امن اولیه: دسترسی‌پذیری میزبان شماره ۴ از میزبان شماره ۱ ( $Path(H1, H4)$ ).  
 وضعیت امن ثانویه بعد از به‌روزرسانی‌ها: دسترسی‌پذیری میزبان شماره ۳ از میزبان شماره ۲ ( $Path(H2, H3)$ ).  
 تعریف ویژگی(های) ایمنی در این آزمایش (خطای پیش‌بینی شده توسط چند به‌روزرسانی پویا):  
 عدم دسترسی‌پذیری میزبان شماره ۳ از میزبان شماره ۱ ( $!Path(H1, H3)$ ). - عدم دسترسی‌پذیری میزبان شماره ۴ از میزبان شماره ۲ ( $!Path(H2, H4)$ ).

### • آزمایش Race condition:

وضعیت امن اولیه: دسترسی‌پذیری میزبان شماره ۳ از میزبان شماره ۱ ( $Path(H1, H3)$ ).  
 وضعیت امن ثانویه بعد از به‌روزرسانی‌ها: دسترسی‌پذیری میزبان شماره ۴ از میزبان شماره ۲ ( $Path(H2, H4)$ ).  
 تعریف ویژگی(های) ایمنی در این آزمایش (خطای پیش‌بینی شده توسط چند به‌روزرسانی پویا):  
 عدم دسترسی‌پذیری میزبان شماره ۴ از میزبان شماره ۱ ( $!Path(H1, H4)$ ) و عدم دسترسی‌پذیری میزبان شماره ۳ از میزبان شماره ۲ ( $!Path(H2, H3)$ ).

### • آزمایش Isolation:

وضعیت امن اولیه: دسترسی‌پذیری میزبان شماره ۴ از میزبان شماره ۱ ( $Path(H1, H4)$ ).  
 وضعیت امن ثانویه بعد از به‌روزرسانی‌ها: دسترسی‌پذیری میزبان شماره ۶ از میزبان شماره ۳ ( $Path(H3, H6)$ ).  
 تعریف ویژگی(های) ایمنی در این آزمایش (خطای پیش‌بینی شده توسط چند به‌روزرسانی پویا):  
 عدم دسترسی‌پذیری میزبان شماره ۶ از میزبان شماره ۱ ( $!Path(H1, H6)$ ).

### • آزمایش Faulty Linear:

وضعیت امن اولیه: دسترسی‌پذیری میزبان شماره ۱۰ از میزبان شماره ۱ ( $Path(H1, H10)$ ).  
 وضعیت امن ثانویه بعد از به‌روزرسانی‌ها: دسترسی‌پذیری میزبان شماره ۱۰ از میزبان شماره ۶ ( $Path(H6, H10)$ ).  
 تعریف ویژگی(های) ایمنی در این آزمایش (خطای پیش‌بینی شده توسط چند به‌روزرسانی پویا):  
 عدم دسترسی‌پذیری میزبان شماره ۱۰ از میزبان شماره ۴ ( $!Path(H4, H10)$ ) و عدم دسترسی‌پذیری میزبان شماره ۸ از میزبان شماره ۶ ( $!Path(H6, H8)$ ).

### • آزمایش Faulty Fattree:



وضعیت امن اولیه: دسترسی‌پذیری میزبان شماره ۸ از میزبان شماره ۲ ( $\text{Path}(\text{H2}, \text{H8})$ ).

وضعیت امن ثانویه بعد از به‌روزرسانی‌ها: دسترسی‌پذیری میزبان شماره ۷ از میزبان شماره ۵ ( $\text{Path}(\text{H5}, \text{H7})$ ) و دسترسی‌پذیری میزبان شماره ۸ از میزبان شماره ۱ ( $\text{Path}(\text{H1}, \text{H8})$ ).

تعریف ویژگی(های) ایمنی در این آزمایش (خطای پیش‌بینی شده توسط چند به‌روزرسانی پویا):

عدم دسترسی‌پذیری میزبان شماره ۷ از میزبان شماره ۱ ( $\text{Path}(\text{H1}, \text{H7})$ )! و عدم دسترسی‌پذیری میزبان شماره ۸ از میزبان شماره ۵ ( $\text{Path}(\text{H5}, \text{H8})$ )! و عدم دسترسی‌پذیری میزبان شماره ۷ از میزبان شماره ۲ ( $\text{Path}(\text{H2}, \text{H7})$ )!.

## ۷-۲- نتایج

در این پژوهش، ۸ آزمایش تعریف شده در بخش ۶-۲- و ویژگی‌های امنیتی تعریف شده در بخش ۷-۱- بررسی شدند. برای دانلود نتایج آزمایش‌های می‌توانید فایل `result_with_detail.xlsx` در مخزن گیت‌هاب ابزار مراجعه کنید. در جدول (۷-۱) به صورت خلاصه می‌توانید نتایج را مشاهده کنید. در ادامه به بررسی نتایج برای هر آزمایش می‌پردازیم:

### • آزمایش Star:

در این آزمایش به طور خودکار، مجموعاً ۱۷ ترتیب مختلف از به‌روزرسانی‌های جداول سوییچ‌ها آزمایش شد که برای ۵ ویژگی امنیتی بررسی شدند. وضعیت امن اولیه یعنی دسترسی‌پذیری میزبان شماره ۲ از میزبان شماره ۱ به ازای تمامی حالات به درستی برقرار بود. وضعیت امن ثانویه یعنی دسترسی‌پذیری مابقی میزبان‌های ۳، ۴، ۵ و ۶ از میزبان ۱ بعد از به‌روزرسانی‌ها نیز به درستی برقرار بود. این آزمایش به هدف بررسی درستی ابزار، حالات امن اولیه و ثانویه ایجاد شده بود و به درستی خطایی در این آزمایش نتیجه گرفته نشد.

### • آزمایش Mesh:

در این آزمایش به طور خودکار، مجموعاً ۵ ترتیب مختلف از به‌روزرسانی‌های جداول سوییچ‌ها آزمایش شد که برای ۵ ویژگی امنیتی بررسی شدند. وضعیت امن اولیه یعنی دسترسی‌پذیری میزبان شماره ۲ از میزبان شماره ۱ برقرار شد. وضعیت امن ثانویه بعد از به‌روزرسانی‌ها یعنی دسترسی‌پذیری مابقی میزبان‌های ۳، ۴، ۵ و ۶ بعد از به‌روزرسانی‌ها نیز به درستی برقرار بودند. این آزمایش به هدف بررسی درستی ابزار، حالات امن اولیه و ثانویه ایجاد شده بود و به درستی خطایی در این آزمایش نتیجه گرفته نشد.

جدول (۷-۱): خلاصه نتایج آزمایش‌ها

نام آزمایش	تعداد پکت قبل از پیش‌پردازش	تعداد پکت بعد از پیش‌پردازش	زمان پیش‌پردازش (ثانیه)	زمان استخراج قوانین (میلی ثانیه)	نوع ویژگی	زمان بررسی ویژگی توسط داینتکت (ثانیه)	نتیجه
Ring	975	36	2.53	0.61	Path(H1,H5)	0.33	Satisfied
					Path(H3,H6)	0.91	Satisfied
					!Path(H3,H5)	0.36	Violated
Mesh	1767	40	4.42	1.22	Path(H1,H2)	0.38	Satisfied
					Path(H1,H3)	0.44	Satisfied
					Path(H1,H4)	0.43	Satisfied
					Path(H1,H5)	0.44	Satisfied
					Path(H1,H6)	0.42	Satisfied
Race Condition	529	32	1.41	0.58	Path(H1,H3)	0.34	Satisfied
					Path(H2,H4)	0.94	Satisfied
					!Path(H1,H4)	0.35	Violated
					!Path(H2,H3)	0.34	Violated
Isolation	2347	32	6.16	0.66	Path(H1,H4)	0.34	Satisfied
					Path(H3,H6)	0.94	Satisfied
					!Path(H1,H6)	0.34	Violated
Blackhole	1893	24	5.19	0.56	Path(H1,H4)	0.34	Satisfied
					!Path(H2,H3) (before_rcfg)	0.33	Satisfied
					Path(H2,H3) (after_rcfgs)	0.82	Satisfied
					!Path(H2,H4)	0.33	Violated
					!Path(H1,H3)	0.33	Violated
Fattree Faulty	11713	60	32.99	1.04	Path(H2,H8)	0.5	Satisfied
					Path(H5,H7)	23.33	Satisfied
					Path(H1,H8)	0.71	Satisfied
					!Path(H1,H7)	23.47	Violated
					!Path(H5,H8)	0.55	Violated
					!Path(H2,H7)	0.64	Violated
Star	1567	72	4.01	2.48	Path(H1,H2)	0.69	Satisfied
					Path(H1,H3)	1.05	Satisfied
					Path(H1,H4)	1.02	Satisfied
					Path(H1,H5)	1.13	Satisfied
					Path(H1,H6)	1.02	Satisfied
Linear Faulty	1660	56	4.23	1.21	Path(H7,H10)	0.5	Satisfied
					Path(H6,H10)	0.54	Satisfied
					!Path(H4,H10)	23.58	Violated
					!Path(H6,H8)	23.63	Violated
Sum	22451	352	60.94	8.36			
Average	2806.37	44	7.61	1.04			

### • آزمایش Ring:

در این آزمایش به طور خودکار، مجموعاً ۵ ترتیب مختلف از به‌روزرسانی‌های جداول سوئیچ‌ها آزمایش شد که برای ۳ ویژگی امنیتی بررسی شدند. وضعیت امن اولیه یعنی دسترسی‌پذیری میزبان شماره ۵ از میزبان شماره ۱ در وضعیت ابتدایی شبکه به درستی برقرار بود. همچنین وضعیت امن ثانویه بعد از به‌روزرسانی‌ها یعنی دسترسی‌پذیری میزبان شماره ۶ از میزبان شماره ۳ نیز برقرار بود. در این آزمایش خطا به صورت عدم دسترسی‌پذیری میزبان شماره ۵ از میزبان شماره ۳ تعریف شده بود که نتیجه ویژگی نشان می‌دهد که یک ترتیب از به‌روزرسانی وجود دارد که موجب نقض این ویژگی می‌شود. در نتیجه توانستیم این خطا را با این ترتیب خاص از به‌روزرسانی‌ها در جداول جریان سوئیچ‌ها پیش‌بینی کنیم.

### • آزمایش Black Hole:

در این آزمایش به طور خودکار، مجموعاً ۵ ترتیب مختلف از به‌روزرسانی‌های جداول سویچ‌ها آزمایش شد که برای ۵ ویژگی امنیتی بررسی شدند. وضعیت امن اولیه یعنی دسترسی‌پذیری میزبان شماره ۴ از میزبان شماره ۱، بدون به‌روزرسانی در جداول جریان سویچ‌ها برقرار شد و عدم دسترسی‌پذیری میزبان شماره ۳ از میزبان شماره ۲ قبل به‌روزرسانی‌ها نیز به درستی برقرار بود. وضعیت امن ثانویه بعد از به‌روزرسانی‌ها یعنی دسترسی‌پذیری میزبان شماره ۳ از میزبان شماره ۲ بعد از به‌روزرسانی در جداول جریان به درستی برقرار شد. در این آزمایش پیش‌بینی خطا به معنی پیدا کردن ترتیبی از به‌روزرسانی‌ها در جداول جریان‌ها برای نقض کردن ویژگی‌های عدم دسترسی‌پذیری میزبان شماره ۳ از میزبان شماره ۱ و عدم دسترسی‌پذیری میزبان شماره ۴ از میزبان شماره ۲ بود که مشاهده شد که ترتیبی از به‌روزرسانی‌ها وجود دارند که باعث نقض شدن این ویژگی‌ها می‌شوند.

#### • آزمایش Race condition:

در این آزمایش به طور خودکار، مجموعاً ۵ ترتیب مختلف از به‌روزرسانی‌های جداول سویچ‌ها آزمایش شد که برای ۴ ویژگی امنیتی بررسی شدند. وضعیت امن اولیه یعنی دسترسی‌پذیری میزبان شماره ۳ از میزبان شماره ۱ به درستی برقرار شد و وضعیت امن ثانویه بعد از به‌روزرسانی‌ها یعنی دسترسی‌پذیری میزبان شماره ۴ از میزبان شماره ۲ نیز برقرار شد. در این آزمایش پیش‌بینی خطا به معنی پیدا کردن ترتیبی از به‌روزرسانی‌ها در جداول جریان‌ها برای نقض کردن ویژگی‌های عدم دسترسی‌پذیری میزبان شماره ۴ از میزبان شماره ۱ و عدم دسترسی‌پذیری میزبان شماره ۳ از میزبان شماره ۲ بود که نتایج نشان می‌دهند ترتیبی از به‌روزرسانی‌های جداول جریان وجود دارد که موجب نقض این ویژگی‌های امنیتی می‌شود.

#### • آزمایش Isolation:

در این آزمایش به طور خودکار، مجموعاً ۵ ترتیب مختلف از به‌روزرسانی‌های جداول سویچ‌ها آزمایش شد که برای ۳ ویژگی امنیتی بررسی شدند. وضعیت امن اولیه یعنی دسترسی‌پذیری میزبان شماره ۴ از میزبان شماره ۱ به درستی بدون هیچ به‌روزرسانی برقرار شد. وضعیت امن ثانویه بعد از به‌روزرسانی‌ها یعنی دسترسی‌پذیری میزبان شماره ۶ از میزبان شماره ۳ نیز با وجود ترتیبی از به‌روزرسانی‌ها برقرار شد. در این آزمایش پیش‌بینی خطا به معنی پیدا کردن ترتیبی از به‌روزرسانی‌ها در جداول جریان‌ها برای نقض کردن ویژگی عدم دسترسی‌پذیری میزبان شماره ۶ از میزبان شماره ۱ بود که آزمایش‌ها نشان دادند ترتیبی از به‌روزرسانی‌ها وجود دارد که باعث نقض این ویژگی می‌شود.

#### • آزمایش Faulty Linear:

در این آزمایش به طور خودکار، مجموعاً ۱۰ ترتیب مختلف از به‌روزرسانی‌های جداول سویچ‌ها آزمایش شد که برای ۴ ویژگی امنیتی بررسی شدند. وضعیت امن اولیه یعنی دسترسی‌پذیری میزبان شماره ۱۰ از میزبان شماره ۱ به درستی برقرار شد و وضعیت امن ثانویه بعد از به‌روزرسانی‌ها یعنی دسترسی‌پذیری میزبان شماره

۱۰ از میزبان شماره ۶ به درستی بعد از به روزرسانی یک سوئیچ برقرار شد. در این آزمایش پیش‌بینی خطا به معنی پیدا کردن ترتیبی از به روزرسانی‌ها در جداول جریان‌ها برای نقض کردن ویژگی‌های عدم دسترسی‌پذیری میزبان شماره ۱۰ از میزبان شماره ۴ و عدم دسترسی‌پذیری میزبان شماره ۸ از میزبان شماره ۶ بود. آزمایش نشان داد ترتیبی از به‌روزرسانی‌ها در جداول سوئیچ‌ها وجود دارد که باعث نقض این ویژگی‌های امنیتی می‌شود.

### • آزمایش Faulty Fattree:

در این آزمایش به طور خودکار، مجموعاً ۱۰ ترتیب مختلف از به‌روزرسانی‌های جداول سوئیچ‌ها آزمایش شد که برای ۶ ویژگی امنیتی بررسی شدند. وضعیت امن اولیه یعنی دسترسی‌پذیری میزبان شماره ۸ از میزبان شماره ۲ بدون به‌روزرسانی در جداول جریان سوئیچ‌ها برقرار شد. نتایج آزمایش‌ها نشان داد ترتیبی از به روزرسانی‌ها در جداول جریان سوئیچ‌ها وجود دارد که وضعیت امن ثانویه بعد از به‌روزرسانی‌ها یعنی دسترسی‌پذیری میزبان شماره ۷ از میزبان شماره ۵ و دسترسی‌پذیری میزبان شماره ۸ از میزبان شماره ۱ برقرار شوند. در این آزمایش پیش‌بینی خطا به معنی پیدا کردن ترتیبی از به روزرسانی‌ها در جداول جریان‌ها برای نقض کردن ویژگی‌های عدم دسترسی‌پذیری میزبان شماره ۷ از میزبان شماره ۱، عدم دسترسی‌پذیری میزبان شماره ۸ از میزبان شماره ۵ و عدم دسترسی‌پذیری میزبان شماره ۷ از میزبان شماره ۲ بود. آزمایش‌ها نشان می‌دهند ترتیبی از به‌روزرسانی‌ها وجود دارد که این ۳ ویژگی امنیتی نیز نقض شوند.

همانطور که در جدول نتایج در مخزن کد قابل مشاهده است، در مجموع ۸ آزمایش و مجموعاً ۳۵ ویژگی (ویژگی وضعیت امن اولیه، ویژگی‌های) وضعیت امن ثانویه، ویژگی‌های پیش‌بینی خطا) مورد بررسی قرار گرفتند. مجموع بسته‌های این ۸ آزمایش قبل پیش‌پردازش ۲۲۴۵۱ و بعد از پیش‌پردازش ۳۵۲ بود و این به معنی کاهش 98.43 درصدی در حجم بسته‌ها برای یادگیری رفتار شبکه‌های نرم‌افزار محور است. از نظر زمانی برای این ۸ آزمایش، در مجموع زمان پیش پردازش 60.94 ثانیه، زمان استخراج رفتار شبکه‌نرم افزار محور 8.36 میلی ثانیه گزارش شد. همچنین در آزمایش مجموعاً ۲۸۵ ترتیب متفاوت از به‌روزرسانی‌ها برای ۳۵ ویژگی امنیتی بررسی شدند که نتایج آن در توضیحات بالا و جدول (۷-۱) قابل مشاهده است. در مجموع بررسی این ۲۸۵ ترتیب متفاوت نیز 761.20 ثانیه زمان برد.

تنها حالت‌هایی که برنامه ما قادر به پاسخگویی آن نبود در آزمایش star بود و داینتکت قادر به بررسی ۱۰ حالت ترتیب به‌روزرسانی‌ها در جداول جریان‌ها نبود. تمام این تحلیل‌ها با حداکثر دو به‌روزرسانی (پیکربندی) در شبکه انجام شده است. این محدودیت ناشی از زیرساخت زبان داینتکت است که در بررسی ویژگی‌های با تعداد بیشتری از پیکربندی‌ها محدودیت محاسباتی دارد. در صورتی که تعداد به‌روزرسانی‌ها افزایش یابد، زمان محاسبات به حدی زیاد می‌شود که اجرای برنامه به دلیل زمان طولانی متوقف می‌شود. این نشان‌دهنده اهمیت بهینه‌سازی الگوریتم‌ها و روش‌های بررسی در آینده برای تحلیل سناریوهای پیچیده‌تر است. سناریوهای مختلف آورده شده است.

قابل ذکر است این آزمایشات بر روی یک کامپیوتر با سیستم‌عامل Ubuntu 22.04 و پردازنده Intel i5-4210U و ۴ گیگابایت حافظه رم<sup>۱</sup> انجام شده است. این تنظیمات سخت‌افزاری یک محیط میان‌رده را نشان می‌دهد و در نتیجه نتایج به‌دست‌آمده قابل تکرار در محیط‌های مشابه هستند. برای تکرار این نتایج، می‌توانید از دستورالعمل‌های دقیق موجود در مخزن گیت‌هاب ابزار استفاده کنید.

با توجه به تحلیل‌های ارائه‌شده، ابزار به‌خوبی توانسته است ویژگی‌های ایمنی شبکه را بررسی کند و خطاهای احتمالی ناشی از پیکربندی نادرست را شناسایی نماید. نتایج نشان‌دهنده کارایی بالای ابزار در یادگیری رفتار و تحلیل شبکه‌های نرم‌افزار محور و همچنین دقت آن در پیش‌بینی وضعیت شبکه در شرایط مختلف است. این آزمایش‌ها نشان می‌دهند که ابزار نه‌تنها برای تحلیل شبکه‌های ساده، بلکه برای شبکه‌های پیچیده با توپولوژی‌های پیشرفته نیز عملکرد موافی داشته است.

### ۷-۳- تحلیل سوالات تحقیق

- آیا می‌توان ابزاری برای یادگیری رفتار شبکه‌های نرم‌افزار محور از داده‌های لاگ شبکه توسعه داد؟  
بله، در این تحقیق، ابزاری با استفاده از تحلیل دقیق داده‌های لاگ شبکه توسعه داده شده است که قادر است رفتار شبکه‌های نرم‌افزار محور را مدل‌سازی کند و به درستی وضعیت شبکه را نشان‌دهد.
- آیا ابزار می‌تواند در شبیه‌سازی سناریوهای واقعی، مانند توپولوژی‌های پیچیده عملکرد مناسبی داشته باشد و نتایج دقیقی ارائه دهد؟ یا وابستگی‌ای به توپولوژی خاصی از شبکه وجود دارد؟  
ابزار توسعه‌یافته در این پژوهش عملکرد مناسبی در شبیه‌سازی سناریوهای واقعی نشان داده است و توانسته است در توپولوژی‌های مختلف به درستی رفتار شبکه را یاد بگیرد. استفاده از روش‌های مبتنی بر تحلیل لاگ‌ها به جای دسترسی مستقیم به شبکه، انعطاف‌پذیری ابزار را افزایش داده و امکان عملکرد در شرایط مختلف توپولوژیکی را فراهم کرده است.
- آیا ابزار توسعه‌یافته در چارچوب این پژوهش می‌تواند در پیش‌بینی خطاهای شبکه‌های نرم‌افزار محور موفق باشد؟  
این ابزار توانسته است خطاهای بالقوه در شبکه‌های نرم‌افزار محور را با دقت بالایی پیش‌بینی کند. این ابزار به‌طور خاص توانسته است ناهنجاری‌ها و رفتارهای غیرمنتظره ناشی از تغییرات پیکربندی و به‌روزرسانی‌ها را شناسایی کند.

<sup>۱</sup> RAM

- چه نوع خطاهایی را می‌توان با استفاده از این ابزار پیش‌بینی کرد؟

این ابزار می‌تواند انواع مختلفی از خطاهای شبکه مانند خطاهای ناشی از تغییرات پویا مانند خطاهای مربوط به به‌روزرسانی همزمان و خطاهای مسیریابی مانند گم شدن بسته‌ها را پیش‌بینی کند.

- مقیاس پذیری ابزار چگونه است؟ حداکثر چه تعداد به‌روزرسانی‌های همزمان در پیکربندی سوییچ‌ها را می‌توان بررسی کرد؟

از نظر سایز توپولوژی و سایز داده لاگ، ابزار مقیاس پذیر است ولی برای بررسی ویژگی‌های امنیتی مربوط به به‌روزرسانی‌های همزمان شبکه فقط می‌توان تا حداکثر دو پیکربندی پویا را بررسی کرد و درحال حاضر امکان بررسی تعداد بیشتر از پیکربندی پویا وجود ندارد.

## فصل ۸: بحث و نتیجه گیری

---

## ۸-۱- مقدمه

در این پژوهش، تمرکز اصلی بر توسعه ابزاری برای پیش‌بینی خطا و مدیریت آن در شبکه‌های نرم‌افزار محور بود. با توجه به اهمیت روزافزون شبکه‌های نرم‌افزار محور در مراکز داده و شبکه‌های پیچیده، اطمینان از عملکرد صحیح این شبکه‌ها و پیش‌بینی و کاهش خطاهای احتمالی بسیار حائز اهمیت است. در این راستا، از زبان دایننتکت که مبتنی بر جبر کلین و روش‌های صوری است، برای مدل‌سازی رفتار شبکه و شناسایی خطاها استفاده شد. برای تحقق این هدف، ابزار FPSDN طراحی و پیاده‌سازی شد که قادر است فایل‌های لاگ واقعی شبکه را پردازش کرده و به‌طور خودکار قوانین و عبارات دایننتکت را استخراج کند.

این پایان‌نامه علاوه بر توسعه ابزار، بر روی ارزیابی کارایی و دقت ابزار در سناریوها با توپولوژی‌های مختلف متمرکز شد که در محیط‌های مراکز داده رایج هستند. از این ابزار برای شبیه‌سازی و تشخیص خطاهای ناشی از پیکربندی‌های پویا استفاده شد، که تأثیر این پیکربندی‌ها بر رفتار شبکه مورد بررسی قرار گرفت.

## ۸-۲- جمع‌بندی

نتایج این پژوهش نشان داد که ابزار FPSDN توانایی پردازش شبکه‌های نرم‌افزار محور را به‌صورت خودکار دارد و قادر است خطاهای احتمالی را به‌طور دقیق تشخیص دهد. با استفاده از فایل‌های لاگ واقعی شبکه و استخراج قوانین دایننتکت، رفتار شبکه به‌صورت رسمی مدل‌سازی شد و تأثیر پیکربندی‌های پویا بر شبکه مورد ارزیابی قرار گرفت. آزمایش‌ها نشان دادند که برخی پیکربندی‌های پویا می‌توانند به خطاهایی منجر شوند که باعث دسترسی‌های غیرمجاز و نقض امنیت شبکه می‌شوند.

## ۸-۳- محدودیت‌ها

این پژوهش اگرچه نتایج امیدوارکننده‌ای ارائه داد، اما محدودیت‌هایی نیز دارد. یکی از محدودیت‌ها، پیچیدگی و زمان‌بر بودن فرآیند پیش‌پردازش داده‌ها در شبکه‌های بزرگ‌تر است. اگرچه FPSDN در شرایط مختلف عملکرد خوبی نشان داد، اما در شبکه‌های بسیار پیچیده با ترافیک سنگین، زمان پردازش به‌طور قابل‌توجهی افزایش می‌یابد. علاوه بر این، هرچند ابزار ما توانایی شبیه‌سازی و تشخیص خطاهای بسیاری را دارد، اما برخی سناریوهای پیچیده‌تر با پیکربندی‌های بیشتر ممکن است به دقت بالاتری نیاز داشته باشند. در نهایت، با وجود موفقیت در پیاده‌سازی FPSDN، این ابزار همچنان نیازمند بهبود در مدیریت پیکربندی‌های چندلایه و سنگین‌تر است که می‌تواند در شبکه‌های بزرگ‌تر و پیچیده‌تر کاربرد داشته باشد.



## ۸-۴- پیشنهادها

با توجه به نتایج حاصل از این پژوهش، چند پیشنهاد برای بهبود عملکرد FPSDN ارائه می‌شود:

- **بهبود بخش پیش‌پردازش:** مرحله پیش‌پردازش داده‌ها، به‌ویژه حذف بسته‌های تکراری و تخصیص پورت‌ها، نقش مهمی در دقت و سرعت پردازش دارد. استفاده از الگوریتم‌های بهینه‌تر برای پیش‌پردازش و موازی‌سازی این عملیات می‌تواند زمان پردازش را بهبود بخشد و عملکرد ابزار را افزایش دهد.
- **توسعه ابزار برای مدل‌سازی بهتر جداول جریان در حال حاضر،** از فیلدهای جداول جریان فقط پورت ورودی و خروجی مدل‌سازی شده است. یکی از کارهای آینده می‌تواند مدل‌سازی بهتر جداول جریان با فیلدهای بیشتر باشد.
- **بهبود ابزار داینکتک مربوط به بررسی ویژگی‌ها در چندین پیکربندی:** بررسی ویژگی‌های مرتبط با چندین پیکربندی در حال حاضر به‌صورت مؤثر انجام می‌شود، اما با بهینه‌سازی کد و موازی‌سازی فرآیندها، زمان لازم برای بررسی این ویژگی‌ها می‌تواند کاهش یابد. همچنین، استفاده از روش‌های پیشرفته‌تر برای بهینه‌سازی می‌تواند دقت و کارایی ابزار را افزایش دهد.

## منابع و مراجع

---

- Al-Fares, M., Loukissas, A., & Vahdat, A. (2008). A scalable, commodity data center network architecture. *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, 63–74. <https://doi.org/10.1145/1402958.1402967>
- Anderson, C. J., Foster, N., Guha, A., Jeannin, J.-B., Kozen, D., Schlesinger, C., & Walker, D. (2014). Netkat: Semantic foundations for networks. *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 113–126. <https://doi.org/10.1145/2535838.2535862>
- Beckett, R., Gupta, A., Mahajan, R., & Walker, D. (2017). A general approach to network configuration verification. *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 155–168. <https://doi.org/10.1145/3098822.3098834>
- Bhardwaj, A., Zhou, Z., & Benson, T. A. (2021). A comprehensive study of bugs in software defined networks. *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 101–115. <https://doi.org/10.1109/DSN48987.2021.00026>
- Caltais, G., Hojjat, H., Mousavi, M. R., & Tunç, H. C. (2021). Dynetkat: An algebra of dynamic networks. *CoRR*, abs/2102.10035. <https://arxiv.org/abs/2102.10035>
- Casado, M., Foster, N., & Guha, A. (2014). Abstractions for software-defined networks. *Commun. ACM*, 57(10), 86–95. <https://doi.org/10.1145/2661061.2661063>
- El-Hassany, A., Miserez, J., Bielik, P., Vanbever, L., & Vechev, M. (2016). Sdnracer: Concurrency analysis for software-defined networks. *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 402–415. <https://doi.org/10.1145/2908080.2908124>
- Elsayed, M. S., Le-Khac, N.-A., & Jurcut, A. D. (2020). Insdn: A novel sdn intrusion dataset. *IEEE Access*, 8, 165263–165284. <https://doi.org/10.1109/ACCESS.2020.3022633>

- Estan, C., & Varghese, G. (2002). New directions in traffic measurement and accounting. *SIGCOMM Comput. Commun. Rev.*, 32(4), 323–336. <https://doi.org/10.1145/964725.633056>
- Gember-Jacobson, A., Viswanathan, R., Prakash, C., Grandl, R., Khalid, J., Das, S., & Akella, A. (2014). Opennf: Enabling innovation in network function control. *SIGCOMM Comput. Commun. Rev.*, 44(4), 163–174. <https://doi.org/10.1145/2740070.2626313>
- Gonzalez, J., Andión, J., Lopez, J., & G., H. A. (2017). Root cause analysis of network failures using machine learning and summarization techniques. *IEEE Communications Magazine*, 55, 126–131. <https://doi.org/10.1109/MCOM.2017.1700066>
- Hamilton., J. (2009). Networking: The last bastion of mainframe computing.
- Hammadi, A., & Mhamdi, L. (2014). A survey on architectures and energy efficiency in data center networks. *Computer Communications*, 40, 1–21. <https://doi.org/https://doi.org/10.1016/j.comcom.2013.11.005>
- Kozen, D. (1997). Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19(3), 427–443. <https://doi.org/10.1145/256167.256195>
- Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1), 14–76. <https://doi.org/10.1109/JPROC.2014.2371999>
- McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., & Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2), 69–74. <https://doi.org/10.1145/1355734.1355746>
- Menaceur, A., Drid, H., & Rahouti, M. (2023). Fault tolerance and failure recovery techniques in software-defined networking: A comprehensive approach. *Journal of Network and Systems Management*, 31. <https://doi.org/10.1007/s10922-023-09772-x>
- Olmezoglu, C. (2022, July). Causal analysis of safety violations in dynetkat. <http://essay.utwente.nl/92646/>

- Scott, C., Wundsam, A., Raghavan, B., Panda, A., Or, A., Lai, J., Huang, E., Liu, Z., El-Hassany, A., Whitlock, S., Acharya, H., Zarifis, K., & Shenker, S. (2014). Troubleshooting blackbox sdn control software with minimal causal sequences. *SIGCOMM Comput. Commun. Rev.*, 44(4), 395–406. <https://doi.org/10.1145/2740070.2626304>
- Serban, C., & Bota, F. (2020). A conceptual framework for software fault prediction using neural networks. In D. Simian & L. F. Stoica (Eds.), *Modelling and development of intelligent systems* (pp. 171–186). Springer International Publishing.
- Sloane, T. (2013). Software-defined networking: The new norm for networks. open networking foundation.
- Smolka, S., Eliopoulos, S., Foster, N., & Guha, A. (2015). A fast compiler for netkat. *SIGPLAN Not.*, 50(9), 328–341. <https://doi.org/10.1145/2858949.2784761>
- The Open Networking Foundation. (2012, June). OpenFlow Switch Specification.
- Yinbo, Y., Li, X., Leng, X., Song, L., Bu, K., Chen, Y., Yang, J., Zhang, L., Cheng, K., & Xiao, X. (2018). Fault management in software-defined networking: A survey. *IEEE Communications Surveys & Tutorials*, PP, 1–1. <https://doi.org/10.1109/COMST.2018.2868922>
- Zeng, H., Kazemian, P., Varghese, G., & McKeown, N. (2012). Automatic test packet generation. *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, 241–252. <https://doi.org/10.1145/2413176.2413205>

## فصل آ: پیوست‌ها

---

## آ-۱- کدها

## آ-۱-۱- توپولوژی FatTree در Mininet

```

1 def myNetwork():
2
3     net = Mininet(topo=None,
4                   build=False,
5                   ipBase='0/8.0.0.10')
6
7     info( '*** Adding controller\n' )
8     c0=net.addController(name='c0',
9                          controller=RemoteController,
10                         ip='1.0.0.127',
11                         protocol='tcp',
12                         port=6633)
13
14     info( '*** Add switches\n')
15     s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
16     s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
17     s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
18     s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
19     s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
20     s6 = net.addSwitch('s6', cls=OVSKernelSwitch)
21     s7 = net.addSwitch('s7', cls=OVSKernelSwitch)
22     s8 = net.addSwitch('s8', cls=OVSKernelSwitch)
23     s9 = net.addSwitch('s9', cls=OVSKernelSwitch)
24     s10 = net.addSwitch('s10', cls=OVSKernelSwitch)
25     s11 = net.addSwitch('s11', cls=OVSKernelSwitch)
26     s12 = net.addSwitch('s12', cls=OVSKernelSwitch)
27     s13 = net.addSwitch('s13', cls=OVSKernelSwitch)
28     s14 = net.addSwitch('s14', cls=OVSKernelSwitch)
29     s15 = net.addSwitch('s15', cls=OVSKernelSwitch)
30     s16 = net.addSwitch('s16', cls=OVSKernelSwitch)
31     s17 = net.addSwitch('s17', cls=OVSKernelSwitch)

```

```

32     s18 = net.addSwitch('s18', cls=OVSKernelSwitch)
33     s19 = net.addSwitch('s19', cls=OVSKernelSwitch)
34     s20 = net.addSwitch('s20', cls=OVSKernelSwitch)
35
36     info( '*** Add hosts\n')
37     h1 = net.addHost('h1', cls=Host, ip='1.0.0.10', defaultRoute=None)
38     h2 = net.addHost('h2', cls=Host, ip='2.0.0.10', defaultRoute=None)
39     h3 = net.addHost('h3', cls=Host, ip='3.0.0.10', defaultRoute=None)
40     h4 = net.addHost('h4', cls=Host, ip='4.0.0.10', defaultRoute=None)
41     h5 = net.addHost('h5', cls=Host, ip='5.0.0.10', defaultRoute=None)
42     h6 = net.addHost('h6', cls=Host, ip='6.0.0.10', defaultRoute=None)
43     h7 = net.addHost('h7', cls=Host, ip='7.0.0.10', defaultRoute=None)
44     h8 = net.addHost('h8', cls=Host, ip='8.0.0.10', defaultRoute=None)
45
46     info( '*** Add links\n')
47     net.addLink(s13, h1)
48     net.addLink(s14, h2)
49     net.addLink(s15, h3)
50     net.addLink(s16, h4)
51     net.addLink(s17, h5)
52     net.addLink(s18, h6)
53     net.addLink(s19, h7)
54     net.addLink(s20, h8)
55     net.addLink(s20, s12)
56     net.addLink(s19, s11)
57     net.addLink(s20, s11)
58     net.addLink(s19, s12)
59     net.addLink(s18, s10)
60     net.addLink(s18, s9)
61     net.addLink(s9, s17)
62     net.addLink(s17, s10)
63     net.addLink(s16, s8)
64     net.addLink(s16, s7)
65     net.addLink(s7, s15)
66     net.addLink(s15, s8)

```



```
67     net.addLink(s6, s14)
68     net.addLink(s14, s5)
69     net.addLink(s5, s13)
70     net.addLink(s13, s6)
71     net.addLink(s5, s1)
72     net.addLink(s5, s2)
73     net.addLink(s6, s3)
74     net.addLink(s6, s4)
75     net.addLink(s7, s1)
76     net.addLink(s7, s2)
77     net.addLink(s8, s3)
78     net.addLink(s8, s4)
79     net.addLink(s9, s1)
80     net.addLink(s9, s2)
81     net.addLink(s10, s3)
82     net.addLink(s10, s4)
83     net.addLink(s11, s1)
84     net.addLink(s11, s2)
85     net.addLink(s12, s3)
86     net.addLink(s12, s4)
87
88     info( '*** Starting network\n')
89     net.build()
90     info( '*** Starting controllers\n')
91     for controller in net.controllers:
92         controller.start()
93
94     info( '*** Starting switches\n')
95     net.get('s1').start([c0])
96     net.get('s2').start([c0])
97     net.get('s3').start([c0])
98     net.get('s4').start([c0])
99     net.get('s5').start([c0])
100    net.get('s6').start([c0])
101    net.get('s7').start([c0])
```

```
102     net.get('s8').start([c0])
103     net.get('s9').start([c0])
104     net.get('s10').start([c0])
105     net.get('s11').start([c0])
106     net.get('s12').start([c0])
107     net.get('s13').start([c0])
108     net.get('s14').start([c0])
109     net.get('s15').start([c0])
110     net.get('s16').start([c0])
111     net.get('s17').start([c0])
112     net.get('s18').start([c0])
113     net.get('s19').start([c0])
114     net.get('s20').start([c0])
115
116     info( '** Post configure switches and hosts\n')
117
118     CLI(net)
119     net.stop()
120
121 if __name__ == '__main__':
122     setLogLevel( 'info' )
123     myNetwork()
```

## **Abstract**

In this research, we study fault prediction in Software-Defined Networks (SDNs) using the DyNetKAT framework. The investigation involves extracting network topology and DynetKAT rules from OpenFlow protocol log files. The proposed tool, FPSDN, facilitates comprehensive analysis of dynamic network behaviors and efficient management of network updates. By utilizing DyNetKAT rules, the tool enables formal verification of network behaviors, including dynamic reconfigurations of switches, with a focus on properties such as reachability. The findings indicate that the tool can accurately identify potential faults and prevent costly disruptions in SDNs. This advancement in fault prediction significantly enhances network reliability, improves performance, and reduces downtime, ensuring optimal operation in dynamic network environments.

**Keywords:** Software Defined Networks, Log File, DyNetKAT, OpenFlow Protocol, Dynamic Reconfigurations, Topology, Switch, Reachability



**KHATAM University**

**Non-governmental, Non-profitable**

**Faculty of Engineering**

**Department of Data Science**

## **Predicting Faults in Software Defined Networks**

A thesis submitted to the Graduate Studies Office

In partial fulfillment of the requirements for

The degree of M.Sc in

Computer Engineering, Data Science

**Supervisor:**

Dr. Hossein Hojjat

**By:**

Mohammadreza Ghobakhlou

September 2024