کلاس topologybuilder

- این تابع یک فایل پیکربندی با فرمت JSON را بارگذاری می‌کند که توپولوژی شبکه را تعریف می‌کند.
- اگر فایل باز نشود یا یافت نشود، یک خطای زمان اجرا پرتاب می‌شود.
- سپس پیکربندی به تابع `buildTopology` ارسال می‌شود تا توپولوژی ساخته شود.

2- ساخت توپولوژی کامل(`buildTopology`)

- این تابع برای هر سیستم خودمختار (AS) در پیکربندی فایل، حلقه می‌زند (`config["Autonomous_systems"]`).
- برای هر AS:
  - روتر DHCP مربوط به AS با استفاده از تابع `getDhcpRouterForAs` بازیابی می‌شود.
  - سپس توابع مختلفی برای ساخت روترها، PCs، مدیریت روترهای خراب، و اتصال PCs به روترها و دروازه‌های AS فراخوانی می‌شود.

3- دریافت روتر DHCP (`getDhcpRouterForAs`)

- این تابع یک روتر پیش‌فرض را برای استفاده به عنوان سرور DHCP برای یک AS خاص برمی‌گرداند.
- روتر DHCP بر اساس `asId` انتخاب می‌شود. به عنوان مثال، برای AS1 ، روتر 5 به عنوان روتر DHCP انتخاب می‌شود.

4 - ساخت روترها و PCs (`buildRoutersAndPCs`)

- این تابع روترها و PCs را براساس پیکربندی برای AS خاص می‌سازد.
- به هر روتر و PC یک آدرس MAC منحصر به فرد اختصاص داده می‌شود که از کلاس `GenerateMACAddress` از طریق (`generateUniqueMAC`) استفاده می‌کند.
- سپس به روترها و PCs آدرس IP اختصاص داده می‌شود که از روتر DHCP درخواست می‌شود.
- روترها و PCs به فهرست‌های مربوطه (`routers`) و (`pcs`) اضافه می‌شوند.

5 - ساخت روترها برای یک AS خاص(`buildRouters`)

- این تابع روترها را برای یک AS خاص می‌سازد و به آن‌ها آدرس IP اختصاص می‌دهد.
- هر روتر ساخته شده و به توپولوژی اضافه می‌شود، و آدرس IP آن از طریق DHCP به آن تخصیص داده می‌شود.

6 - ساخت PCs برای یک AS خاص(`buildPCs`)

- این تابع PCs را برای AS می‌سازد و به آن‌ها آدرس IP اختصاص می‌دهد که از سرور DHCP گرفته می‌شود.
- هر PC ساخته شده و به توپولوژی اضافه می‌شود و آدرس IP آن از طریق DHCP سرور به آن داده می‌شود.

7 - مدیریت روترهای خراب(`handleBrokenRouters`)

- این تابع روترهای خراب را از توپولوژی حذف می‌کند.
- پیکربندی بررسی می‌شود تا هر روتر خراب در فهرست موجود باشد و سپس آن روتر از نقشه‌ی `routers`حذف می‌شود.

8 -ساخت توپولوژی(`buildASTopology`) AS

- این تابع توپولوژی شبکه را در یک AS خاص بر اساس نوع توپولوژی (`topology_type`) که می‌تواند "Mesh" یا "RingStar"باشد) می‌سازد.
  - **توپولوژی :Mesh** هر روتر به تمامی روترهای دیگر در AS متصل می‌شود.
  - **توپولوژی :RingStar** روترها به صورت دایره‌ای به یکدیگر متصل می‌شوند.
- برای هر اتصال، روترها با استفاده از تابع `connectRouter`به هم متصل می‌شوند.

9 - اتصال PCs به روترها(`connectPCsToRouters`)

- این تابع PCs را به روترهای خود (دروازه‌ها) متصل می‌کند.
- هر روتر و PC از طریق تابع `connectUser`به هم متصل می‌شوند.

10 - اتصال دروازه‌های AS ها(`connectAsGateways`)

- این تابع دروازه‌های سیستم‌های خودمختار مختلف را به هم متصل می‌کند.
- پیکربندی برای هر جفت دروازه‌ی متصل به یکدیگر بررسی می‌شود و روترها بر اساس آن با استفاده از تابع `connectRouter`به هم متصل می‌شوند.

11 - چاپ توپولوژی(`printTopology`)

- این تابع توپولوژی کامل شبکه را چاپ می‌کند که شامل اطلاعاتی از روترها، روترهای متصل به آن‌ها و PCs متصل به آن‌ها است.
- برای هرAS ، روترها و دستگاه‌های متصل به آن‌ها) روترها و (PCs چاپ می‌شود.

خلاصه:

- **ساخت روترها و PCs:** روترها و PCs برای هر AS ساخته می‌شوند و آدرس‌های MAC و IP منحصر به فرد به آن‌ها اختصاص داده می‌شود.
- **اتصال‌ها:** روترها و PCs بر اساس توپولوژی مشخص Mesh) یا RingStar) به یکدیگر متصل می‌شوند.
- **DHCP:** روتر DHCP آدرس‌های IP را به روترها و PCs اختصاص می‌دهد.
- **پایداری سیستم:** روترهای خراب از توپولوژی حذف می‌شوند.
- **اتصال دروازه‌ها:** دروازه‌های AS های مختلف به یکدیگر متصل می‌شوند.
- **ثبت فعالیت‌ها:** سیستم فرآیندهای ساخت روترها، PCsو اتصالات را ثبت می‌کند و در صورت بروز خطاها یا هشدارها)مانند یافت نشدن روتر یا(PC ، آن‌ها را گزارش می‌کند.

این سیستم اطمینان حاصل می‌کند که هر روتر و PC در شبکه دارای یک شناسه منحصر به فرد) آدرس (MAC است و به درستی به شبکه متصل می‌شود مطابق با توپولوژی تعریف‌شده در پیکربندی.

کلاس DHPCServer

کلاس `DHCP` که در اینجا تعریف شده است، مسئول مدیریت تخصیص آدرس‌های IP به گره‌ها در شبکه است. این کلاس برای تخصیص IP از یک بازه خاص استفاده می‌کند و از آنجایی که آدرس‌های IP می‌توانند در طول زمان به گره‌ها اختصاص یابند، عملکردهای مختلفی برای تخصیص، ذخیره‌سازی و مدیریت آدرس‌ها وجود دارد. در اینجا یک گزارش از کد کلاس آورده شده است:

1.متغیرهای کلاس

- `ipCounter`: یک متغیر استاتیک که تعداد آدرس‌های IP اختصاص داده شده را نگه می‌دارد. این متغیر شمارنده‌ای برای تعیین آدرس بعدی در بازه IP است.
- `ipRange`: بازه آدرس‌های IP که DHCP از آن استفاده می‌کند.
- `asID`: شناسه سیستم خودمختار (AS) که این DHCP مربوط به آن است.
- `allocatedIPs`: یک نقشه که آدرس‌های IP تخصیص داده شده را برای هر گره ذخیره می‌کند. کلید نقشه شناسه گره است و مقدار آن آدرس IP تخصیص داده شده به آن گره است.
- `macAddressTable`: یک نقشه دیگر که آدرس‌های MAC گره‌ها را ذخیره می‌کند.

2.سازنده(`constructor`)

- سازنده کلاس `DHCP` که بازه IP و شناسه AS را به عنوان ورودی می‌گیرد و آن‌ها را به متغیرهای مربوطه نسبت می‌دهد.

3.تابع `generateIPAddress`

- این تابع یک آدرس IP جدید بر اساس بازه مشخص شده تولید می‌کند.
- شمارنده (`ipCounter`) IP افزایش می‌یابد و آدرس جدید با استفاده از آن تولید می‌شود.
- اگر `ipCounter`بیشتر از 254 شود) که نشان‌دهنده اتمام بازه IP است(، یک استثنا پرتاب می‌شود.

4.تابع `handleDiscover`

- این تابع برای مدیریت مرحله DISCOVER در پروتکل DHCP است.
- بررسی می‌کند که آیا به گره‌ای که شناسه‌اش `nodeId`است، قبلاً IP اختصاص داده شده است یا خیر.
- اگر IP اختصاص داده نشده باشد، یک IP جدید برای گره تخصیص داده می‌شود و آدرس MAC آن در جدول MAC ذخیره می‌شود.
- در نهایت آدرس IP تخصیص داده شده به گره برگشت داده می‌شود.

5.تابع `saveAllocationsToFile`

- این تابع تخصیصات) DHCP آدرس‌های IP و آدرس‌های (MAC را به یک فایل متنی (`dhcp_allocations.txt`) ذخیره می‌کند.
- اگر هیچ تخصیصی وجود نداشته باشد، پیامی مبنی بر عدم تخصیص IP نمایش داده می‌شود.
- اگر تخصیصات وجود داشته باشد، آن‌ها به فایل ذخیره می‌شوند و یک پیام تأیید برای ذخیره‌سازی نمایش داده می‌شود.

6.تابع `assignIP`

- این تابع به گره‌ای با شناسه `nodeID`یک آدرس IP تخصیص می‌دهد.
- اگر این گره قبلاً آدرس IP دریافت کرده باشد، IPقبلی باز می‌گردد.
- در غیر این صورت، یک آدرس IP جدید از بازه تولید می‌شود و به گره تخصیص داده می‌شود.
- اگر تخصیص IP برای این بازه تمام شده باشد، یک استثنا پرتاب می‌شود.
- سپس آدرس IP به گره اختصاص داده می‌شود و شمارنده IP برای تخصیص IP به گره بعدی افزایش می‌یابد.

7.مدیریت آدرس‌های IP وMAC

- این کلاس آدرس‌های IP و MAC گره‌ها را در دو نقشه مجزا ذخیره می‌کند `allocatedIPs` :برای آدرس‌های IP و `macAddressTable`برای آدرس‌های.MAC
- هنگامی که یک گره برای اولین بار IP دریافت می‌کند، آدرس MAC آن نیز ثبت می‌شود.

8.سایر ویژگی‌ها

- **محدودیت‌های بازه IP**: بازه IP محدود به 254 آدرس است و در صورت رسیدن به این حد، آدرس‌های جدید نمی‌توانند تخصیص یابند.
- **فایل ذخیره‌سازی**: تمام تخصیص‌های IP به صورت مرتب در یک فایل متنی ذخیره می‌شوند، که شامل شناسهAS ، شناسه گره، آدرس IP و آدرس MAC می‌شود.

خلاصه:

کلاس DHCPبه طور کلی برای تخصیص آدرس‌های IP به گره‌ها در یک شبکه طراحی شده است. این کلاس فرآیند تخصیص آدرس، ذخیره‌سازی تخصیص‌ها و مدیریت آدرس‌های IP و MAC را انجام می‌دهد. همچنین، برای جلوگیری از تخصیص IP بیشتر از ظرفیت بازه، محدودیتی برای تعدادIP ها تعیین شده است. فایل تخصیصات DHCP نیز به طور خودکار ذخیره می‌شود تا گزارش‌هایی از تخصیص‌ها در اختیار قرار گیرد.

کلاس RIP

کلاس RIPکه در کد تعریف شده است، یک پیاده‌سازی از پروتکل مسیریابی RIP (Routing Information Protocol) برای یک روتر است. این کلاس شامل توابعی است که برای شناسایی همسایه‌ها، ارسال به‌روزرسانی‌های مسیریابی، پردازش به‌روزرسانی‌ها، و مدیریت جدول مسیریابی روتر طراحی شده‌اند. در اینجا گزارشی از عملکرد این کلاس آورده شده است:

1- متغیرهای کلاس

- routerاشاره‌گری به شیء کلاس Routerکه روتر را نمایش می‌دهد.
- protocolCompleteمتغیر بولی که نشان می‌دهد آیا پروتکل RIP تکمیل شده است یا خیر.
- neighborsیک نقشه که شناسه همسایه‌ها (IP) آدرس‌ها (و هزینه آن‌ها را ذخیره می‌کند.
- routingTableیک نقشه که جدول مسیریابی را ذخیره می‌کند. هر ورودی در این جدول شامل اطلاعاتی نظیر شبکه مقصد، ماسک زیرشبکه، گیت‌وی، هزینه و پروتکل مسیریابی است.
- routingMutexیک mutex برای همگام‌سازی دسترسی به جدول مسیریابی در محیط‌های چند نخ(multi-threaded).

2- سازنده(constructor)

- سازنده کلاس RIPیک اشاره‌گر به شیء Routerدریافت می‌کند که برای ارسال و دریافت بسته‌ها از آن استفاده می‌شود.

- متغیر protocolCompleteبه مقدار اولیه falseتنظیم می‌شود.

3- تابع identifyNeighbors

- این تابع همسایه‌ها را شناسایی می‌کند. برای هر پورت در روتر، بررسی می‌شود که آیا پورت قادر به ارسال پیغام hello است یا خیر.
- اگر پورت به یک گره متصل باشد، اطلاعات همسایه شامل شناسه گره (ID) و هزینه آن در نقشه neighborsذخیره می‌شود.
- سپس یک ورودی جدید به جدول مسیریابی اضافه می‌شود که شامل اطلاعات شبکه مقصد، ماسک زیرشبکه، گیت‌وی بعدی، هزینه و پروتکل RIP است.

4- تابع sendRoutingUpdates

- این تابع برای ارسال به‌روزرسانی‌های مسیریابی به همسایه‌ها طراحی شده است.
- برای هر همسایه موجود در نقشه neighbors، یک بسته با اطلاعات جدول مسیریابی ایجاد شده و به آن همسایه ارسال می‌شود.
- داده‌های جدول مسیریابی به صورت رشته‌ای سریالیزه می‌شود و درون بسته قرار می‌گیرد.

5- تابع processRoutingUpdates

- این تابع برای پردازش به‌روزرسانی‌های مسیریابی دریافت‌شده از همسایه‌ها طراحی شده است.
- ابتدا داده‌های جدول مسیریابی از بسته دریافتی سریالیزه و پردازش می‌شود.
- سپس برای هر مقصد موجود در جدول دریافتی، هزینه جدید محاسبه می‌شود. اگر هزینه جدید از هزینه موجود کمتر باشد یا مقصد جدیدی باشد، جدول مسیریابی به‌روزرسانی می‌شود.

6- تابع serializeRoutingTable

- این تابع جدول مسیریابی را به یک رشته سریالیزه تبدیل می‌کند.
- اطلاعات مربوط به هر ورودی جدول مسیریابی (شبکه مقصد، ماسک زیرشبکه، گیت‌وی، هزینه و پروتکل) در قالب یک رشته جداشده با ویرگول و علامت‌گذاری‌شده با نقطه‌ویرگول ذخیره می‌شود.

7- تابع deserializeRoutingTable

- این تابع یک رشته سریالیزه‌شده را دریافت کرده و آن را به یک جدول مسیریابی معادل تبدیل می‌کند.

- داده‌ها با استفاده از علامت نقطه‌ویرگول به بخش‌های مختلف تقسیم شده و سپس هر بخش به صورت جداگانه پردازش می‌شود تا اطلاعات مربوط به شبکه مقصد، ماسک زیرشبکه، گیت‌وی و هزینه استخراج شود.

8- تابع `initialize`

- این تابع برای شروع فرآیند RIP طراحی شده است.
- ابتدا همسایه‌ها شناسایی می‌شوند، سپس به‌روزرسانی‌های مسیریابی ارسال می‌شود.
- بعد از آن، تابع `processUpdates`برای پردازش به‌روزرسانی‌ها فراخوانی می‌شود.

9- تابع `processUpdates`

- این تابع برای پردازش به‌روزرسانی‌های مسیریابی که از همسایه‌ها دریافت می‌شود به صورت پیوسته اجرا می‌شود.
- یک تایمر برای محدود کردن مدت زمان اجرای پروتکل در نظر گرفته شده است. پس از 60 ثانیه، پروتکل به اتمام می‌رسد و متغیر `protocolComplete`به `true`تغییر می‌یابد.
- در این مدت، بسته‌ها از بافر روتر دریافت و در صورتی که نوع بسته کنترل باشد(Control packet) ، به‌روزرسانی‌های مسیریابی پردازش می‌شود.

10- تابع `displayRoutingTable`

- این تابع جدول مسیریابی را نمایش می‌دهد.
- برای جلوگیری از مشکلات همگام‌سازی در محیط‌های چند نخ، دسترسی به جدول مسیریابی توسط `mutex`محافظت می‌شود.

11- تابع `isProtocolComplete`

- این تابع بررسی می‌کند که آیا پروتکل RIP تکمیل شده است یا خیر.
- اگر مدت زمان لازم برای اتمام پروتکل سپری شده باشد، این تابع مقدار `true`را باز می‌گرداند.

جمع‌بندی:

کلاس `RIP`مسئول اجرای پروتکل مسیریابی RIP است. این کلاس همسایه‌ها را شناسایی کرده، به‌روزرسانی‌های مسیریابی را ارسال و پردازش می‌کند، و جدول مسیریابی روتر را مدیریت می‌کند. همچنین قابلیت سریالیزه و دزریالیزه کردن داده‌های جدول مسیریابی را داراست و می‌تواند به‌روزرسانی‌ها را به صورت دوره‌ای پردازش کند. این پروتکل به صورت خودکار به‌روزرسانی‌های مسیریابی را به همسایه‌ها ارسال کرده و جدول مسیریابی را بر اساس هزینه‌ها و مسیرهای جدید به‌روز می‌کند.

کلاس autonomoussystem

کلاس AutonomousSystem یک مدل از یک سیستم خودمختار در شبکه است. این کلاس برای مدیریت مجموعه‌ای از روترها، کامپیوترهاPC) ها (و اتصال به دیگر سیستم‌های خودمختار استفاده می‌شود. در اینجا گزارشی از عملکرد این کلاس آورده شده است:

1- متغیرهای کلاس

- id: شناسه سیستم خودمختار (AS) که به هر سیستم تخصیص داده می‌شود.
- routers: یک بردار از اشاره‌گرهای هوشمند (shared pointers) به روترها که متعلق به این سیستم خودمختار هستند.
- pcs: یک بردار از اشاره‌گرهای هوشمند به کامپیوترها که متعلق به این سیستم خودمختار هستند.
- gateway: اشاره‌گری به روتر که به عنوان دروازه (Gateway) برای این سیستم خودمختار عمل می‌کند.
- connectedAS: یک بردار از اشاره‌گرهای هوشمند به سیستم‌های خودمختار متصل که با این سیستم ارتباط دارند.

2- سازنده(constructor)

- سازنده کلاس AutonomousSystem یک شناسه (id) دریافت می‌کند که به سیستم خودمختار اختصاص داده می‌شود.

3- توابع اصلی کلاس

- **تابع getId:**
  - این تابع شناسه سیستم خودمختار را برمی‌گرداند.
- **تابع addRouter:**
  - این تابع یک اشاره‌گر به یک روتر دریافت کرده و آن را به لیست routers اضافه می‌کند.
  - همچنین پیامی به کنسول می‌فرستد که نشان‌دهنده اضافه شدن روتر به سیستم خودمختار است.
- **تابع addPC:**
  - این تابع یک اشاره‌گر به یک کامپیوتر (PC) دریافت کرده و آن را به لیست pcs اضافه می‌کند.
  - یک پیام به کنسول ارسال می‌کند که نشان‌دهنده اضافه شدن کامپیوتر به سیستم خودمختار است.
- **تابع getRouter:**
  - این تابع یک شناسه روتر (id) دریافت می‌کند و تلاش می‌کند روتر مربوطه را از لیست routers پیدا کند و برگرداند.
  - اگر روتر با شناسه مورد نظر پیدا نشود، مقدار nullptr برمی‌گرداند.
- **تابع getPC:**
  - مشابه تابع getRouter است، اما این تابع به دنبال یک کامپیوتر با شناسه داده‌شده در لیست pcs می‌گردد و آن را برمی‌گرداند.
- **تابع setGateway:**
  - این تابع یک روتر را به عنوان دروازه (gateway) برای سیستم خودمختار تعیین می‌کند.
  - یک پیام به کنسول می‌فرستد که نشان‌دهنده تعیین روتر به عنوان دروازه است.
- **تابع getGateway:**
  - این تابع اشاره‌گر به روتر دروازه (gateway) سیستم خودمختار را برمی‌گرداند.
- **تابع connectToAS:**
  - این تابع یک سیستم خودمختار دیگر را به سیستم خودمختار کنونی متصل می‌کند.
  - یک پیام به کنسول می‌فرستد که نشان‌دهنده اتصال دو سیستم خودمختار به یکدیگر است.
- **تابع initializeDHCP:**
  - این تابع یک سرور DHCP را بر روی روتر با شناسه داده‌شده فعال می‌کند.

- ابتدا تلاش می‌شود روتر مربوطه از لیست روترها پیدا شود. اگر روتر پیدا شود، سرور DHCP بر روی آن روتر فعال می‌شود و پیامی به کنسول می‌فرستد. در غیر این صورت، خطای مربوطه به کنسول ارسال می‌شود.

- **تابع `getRouters`:**
  - این تابع یک بردار از اشاره‌گرهای روترهای موجود در سیستم خودمختار را برمی‌گرداند.

4- خلاصه عملکرد کلاس:

کلاس `AutonomousSystem`برای مدیریت اجزای مختلف شبکه در یک سیستم خودمختار مانند روترها، کامپیوترها و دروازه‌ها استفاده می‌شود. این کلاس همچنین از قابلیت اتصال به سایر سیستم‌های خودمختار پشتیبانی می‌کند و می‌تواند با استفاده از DHCP سرورهای مربوطه را مدیریت کند.

عملکرد این کلاس شامل موارد زیر است:

- اضافه کردن و مدیریت روترها و کامپیوترها.
- شناسایی و دریافت روترها و کامپیوترها بر اساس شناسه.
- تنظیم و دریافت دروازه (gateway) سیستم خودمختار.
- اتصال سیستم‌های خودمختار به یکدیگر.
- راه‌اندازی سرور DHCP بر روی یک روتر.
- مدیریت و دسترسی به لیست روترها.

این کلاس به عنوان بخشی از یک سیستم پیچیده‌تر برای مدیریت شبکه‌ها و اتصال بین سیستم‌های خودمختار عمل می‌کند و می‌تواند در برنامه‌های مدیریت شبکه استفاده شود.

کلاس OSPF

####مقدمه پروتکل OSPF (Open Shortest Path First) برای یافتن کوتاه‌ترین مسیر بین نودهای شبکه و مدیریت توپولوژی طراحی شده است.
در این شبیه‌ساز، عملکرد OSPF با استفاده از الگوریتم دایجسترا ارزیابی و نتایج آن تحلیل شده است
--- .نحوه عملکرد پروتکل .1ایجاد توپولوژی OSPF - :(Link-State Database) ابتدا توپولوژی شبکه را با ذخیره تمامی لینک‌ها و وزن‌های مرتبط در یک پایگاه داده تشکیل می‌دهد. این داده‌ها شامل اطلاعات مربوط به نودها و ارتباطات آن‌ها است.

2- محاسبه کوتاه‌ترین مسیرها - :پروتکل از الگوریتم دایجسترا برای محاسبه کوتاه‌ترین مسیرها از یک نود مبدا به تمامی نودهای دیگر استفاده می‌کند. این محاسبات بر اساس وزن لینک‌ها انجام می‌شود که می‌تواند نشان‌دهنده هزینه، تأخیر، یا پهنای باند باشد.

3- -.-به‌روزرسانی جدول مسیریابی - :نتایج الگوریتم به‌صورت جدول مسیریابی ذخیره می‌شوند. این جدول شامل مسیر بهینه برای رسیدن به هر مقصد است.

4- .ارسال بسته‌ها - :بسته‌ها با استفاده از اطلاعات جدول مسیریابی ارسال می‌شوند. در طول فرآیند، مسیرهای دقیق و وضعیت بسته‌ها ثبت می‌شوند.

5- .تحلیل عملکرد - :تأخیر ارسال، نرخ از دست رفتن بسته‌ها، و استفاده از پهنای باند محاسبه می‌شوند.