گزارش پروژه – CA3 فاز دوم

کلاس**ApplicationContext :**

کلاس **ApplicationContext**منطق اصلی پیکربندی و مدیریت برنامه را در خود جای داده است که برای تسهیل راه‌اندازی و رفتار در زمان اجرای یک محیط شبیه‌سازی طراحی شده است.

متدهای عمومی:

1. **loadConfiguration**
   - **تعریف**: bool loadConfiguration(const QString &filePath)
   - **توضیح** :یک فایل پیکربندی JSON را تجزیه کرده و تنظیمات شبیه‌سازی و سیستم‌های خودمختار را مقداردهی اولیه می‌کند.
   - **عملکرد** :
     - خواندن و اعتبارسنجی ورودی JSON از یک فایل.
     - استخراج پارامترهایی مانند :
       - simulation_duration:مدت شبیه‌سازی.
       - cycle_duration:مدت چرخه.
       - TTL:زمان زنده ماندن.
       - packets_per_simulation:تعداد بسته‌ها در شبیه‌سازی.
       - statistical_distribution:توزیع آماری.
     - ساخت و مقداردهی اولیه اشیاء **AutonomousSystem**از پیکربندی.
     - در صورت موفقیت‌آمیز بودن، مقدار trueرا برمی‌گرداند و در غیر این صورت .false
   - **موردهای خاص مدیریت‌شده** :
     - مسیر فایل نادرست یا غیرقابل دسترسی.
     - فرمتی نادرست یا گم‌شده در.JSON
2. **asGateways**
   - **تعریف**: QList<int> asGateways(int asId) const
   - **توضیح** :فهرستی از دروازه‌های **AS**برای یک سیستم خودمختار مشخص (asId) را برمی‌گرداند.

- **عملکرد :**
  - از طریق لیست سیستم‌های خودمختار (**m_autonomousSystems**) پیمایش می‌کند.
  - **asId**داده‌شده را جستجو کرده و دروازه‌های مربوطه را برمی‌گرداند.
  - در صورت عدم یافتن **asId**، فهرستی خالی بازمی‌گرداند.

3. **userGateways**
   - **تعریف**QList<int> userGateways(int asId) const :
   - **توضیح** :فهرستی از دروازه‌های کاربری برای یک سیستم خودمختار مشخص (**asId**) را برمی‌گرداند.
   - **عملکرد :**
     - مشابه با **asGateways**عمل می‌کند، اما برای دروازه‌های کاربری.
     - در صورت عدم یافتن **asId**، فهرستی خالی بازمی‌گرداند.

متغیرهای خصوصی:

1. **پارامترهای شبیه‌سازی:**
   - QString m_simulationDuration:مدت زمان شبیه‌سازی.
   - QString m_cycleDuration:مدت زمان چرخه.
   - int m_ttl:زمان زنده ماندن بسته‌ها.
   - int m_packetsPerSimulation:تعداد بسته‌ها در هر شبیه‌سازی.
   - QString m_statisticalDistribution:توزیع آماری.
2. **لیست سیستم‌های خودمختار:**
   - QList<AutonomousSystem> m_autonomousSystems:نگهداری مجموعه‌ای از اشیاء **AutonomousSystem**که نمایانگر محیط‌های **AS**در شبیه‌سازی هستند.

وابستگی‌های کلیدی:

- **کلاس‌های فریم‌ورک :Qt**
  - QObjectبرای عملکردهای پایه‌ای کلاس.
  - QString, QListو *QJsonبرای مدیریت داده‌ها و پردازش.
  - QFileبرای عملیات مربوط به فایل.

- qDebug()برای اشکال‌زدایی.

- **نوع‌های سفارشی** :

- **AutonomousSystem**، **Gateway**، :**ASConnection**کلاس‌ها یا ساختارهای کمکی که به‌صورت خارجی برای مدل‌سازی سیستم‌های خودمختار، دروازه‌ها و ارتباطات تعریف شده‌اند.

کلاس Port :

کلاس **Port**نمایانگر یک پورت شبکه است که مسئول ارسال و دریافت بسته‌ها (Packets) می‌باشد. این کلاس همچنین قابلیت‌های مدیریتی مختلفی برای تنظیمات مربوط به پورت مانند آدرس IP روتر و شمارش بسته‌ها را فراهم می‌کند.

متدهای عمومی:

1. **سازنده**:
   - **تعریف**: Port(QObject *parent = nullptr)
   - **توضیح** :این سازنده یک شیء **Port**جدید ایجاد می‌کند و مقادیر اولیه به‌صورت پیش‌فرض تنظیم می‌شوند :
     - m_numberبرابر با) 0شمارنده پورت.(
     - m_numberOfPacketsSentبرابر با) 0تعداد بسته‌های ارسال‌شده.(
     - m_routerIPبرابر با "127.0.0.1"آدرس IP پیش‌فرض روتر.
2. **sendPacket**:
   - **تعریف**: void sendPacket(const PacketPtr_t &data)
   - **توضیح** :این متد مسئول ارسال بسته داده‌شده است.
   - **عملکرد** :
     - بررسی می‌کند که آیا داده (data) خالی نیست. اگر بسته خالی باشد، پیامی به سطح هشدار (qWarning) ارسال می‌شود.
     - شمارنده تعداد بسته‌های ارسال‌شده افزایش می‌یابد.
     - سیگنال packetSentصادر می‌شود تا اطلاع دهد که بسته ارسال شده است.
3. **receivePacket**:
   - **تعریف**: void receivePacket(const PacketPtr_t &data)
   - **توضیح** :این متد مسئول دریافت بسته داده‌شده است.

- **عملکرد :**
  - بررسی می‌کند که آیا داده خالی نیست. در صورت خالی بودن بسته، پیامی به سطح هشدار ارسال می‌شود.
  - سیگنال packetReceivedصادر می‌شود تا اطلاع دهد که بسته دریافت شده است.

4. **routerIP**:
   - **تعریف**: QString routerIP() const
   - **توضیح** :این متد آدرس IP روتر را بازمی‌گرداند.
   - **عملکرد** :به‌طور ساده مقدار متغیر m_routerIPرا برمی‌گرداند.

کلاسRouter :

کلاس **Router**نمایانگر یک روتر در شبکه است که از کلاس پایه **Node**ارث‌بری می‌کند. این کلاس مسئول مدیریت جدول مسیریابی و تنظیمات پورت‌ها می‌باشد.

متدهای جدول مسیریابی:

1. **addRoutingTableEntry**:
   - این متد ورودی جدیدی را به جدول مسیریابی روتر اضافه می‌کند.
2. **removeRoutingTableEntry**:
   - این متد ورودی‌ای را از جدول مسیریابی حذف می‌کند.
3. **displayRoutingTable**:
   - این متد جدول مسیریابی را در کنسول نمایش می‌دهد.
4. **findBestRoute**:
   - این متد ساده‌ترین منطق مسیریابی را پیاده‌سازی می‌کند.

کلاسPC :

کلاس **PC**نمایانگر یک کامپیوتر در شبکه است که مسئول ارسال و دریافت بسته‌ها، مدیریت آدرس‌های IP و سایر ویژگی‌ها می‌باشد.

متدها:

1. **setOtherIPs**:
   - این متد لیستی از آدرس‌های IP دیگر کامپیوترها یا دستگاه‌ها را به متغیر otherIPsاختصاص می‌دهد.

2. **generateAndSendPackets**:
   - این متد بسته‌هایی را تولید و ارسال می‌کند.
3. **connectToRouter**:
   - این متد کامپیوتر را به یک روتر متصل می‌کند.
4. **processIncomingPacket**:
   - این متد بسته‌ای که از روتر یا دیگر منابع به کامپیوتر ارسال شده، پردازش می‌کند.

کلاس(Test (Main.cpp :

در این بخش، تمامی تست‌ها برای بخش‌های مختلف پروژه نوشته شده است. این تست‌ها به صورت توابع جداگانه در **main.cpp**قرار دارند.

توپولوژی:

1. **testMeshTopology**:تست ایجاد توپولوژی Mesh با تعداد مشخصی روتر و کامپیوتر.
2. **testStarRingTopology**:تست ایجاد توپولوژی Star-Ring با تعداد مشخصی روتر و کامپیوتر.

AutonomousSystem:

1. **testAutonomousSystem**:ایجاد یک سیستم خودمختار(AS) ، افزودن روتر و کامپیوتر به آن، و اتصال AS به یک AS دیگر.

انتقال بسته‌ها:

1. **testPacketTransmission**:تست ارسال و دریافت بسته‌ها از طریق پورت‌ها.

گزارش کلاس Topology builder

هدف کلاس

کلاس `TopologyBuilder` وظیفه ساخت توپولوژی‌های مختلف شبکه شامل توپولوژی‌های مش (Mesh) و حلقه ستاره‌ای (Star-Ring) را بر عهده دارد. این کلاس قابلیت ایجاد روترها و کامپیوترها(PCs) ، اتصال آنها بر اساس توپولوژی مشخص، و مدیریت ساختار شبکه را فراهم می‌کند .

ساختار کلاس

سازنده‌ها و مخرب

- `TopologyBuilder()`
  سازنده پیش‌فرض که وظیفه مقداردهی اولیه ساختار کلاس را دارد.
- `~TopologyBuilder()`
  مخرب پیش‌فرض که برای پاکسازی منابع مورد استفاده قرار می‌گیرد.

توابع عضو

void buildMeshTopology(int numRouters, int numPCs)

این تابع توپولوژی مش را ایجاد می‌کند.

مراحل اجرا:

پاکسازی توپولوژی موجود.

ایجاد numRouters روتر و ذخیره آنها در لیست.routers

اتصال تمام روترها به یکدیگر به صورت کامل.(Full Mesh)

ایجاد numPCs کامپیوتر و اتصال تصادفی آنها به یکی از روترها.

void buildStarRingTopology(int numRouters, int numPCs)

این تابع توپولوژی حلقه ستاره‌ای را ایجاد می‌کند.

مراحل اجرا:

1. پاکسازی توپولوژی موجود.
2. ایجاد numRouters روتر و اتصال آنها به صورت حلقه.
3. اضافه کردن یک روتر مرکزی و اتصال تمام روترها به این مرکز.
4. ایجاد numPCs کامپیوتر و اتصال تصادفی آنها به یکی از روترها.

void resetTopology()

توپولوژی موجود را پاک می‌کند.

عملیات انجام‌شده:

لیست روترها (routers) و کامپیوترها (pcs) را خالی می‌کند.

void connectNodes(std::shared_ptr<Node> node1, std::shared_ptr<Node> node2)

دو گره شبکه (روتر یا کامپیوتر) را به هم متصل می‌کند.

مراحل اجرا:

1. ایجاد دو پورت جدید برای هر گره.
2. اتصال دو پورت به یکدیگر.
3. اضافه کردن پورت‌ها به گره‌های مربوطه.
4. چاپ پیام اتصال در کنسول.

ساختار داده‌ها

1. لیست روترها

std::vector<std::shared_ptr<Router>> routers;

ذخیره تمامی روترهای ایجادشده.

2. لیست کامپیوترها

std::vector<std::shared_ptr<PC>> pcs;

ذخیره تمامی کامپیوترهای ایجادشده.

مزایا و ویژگی‌ها

- **انعطاف‌پذیری** :امکان ساخت توپولوژی‌های مختلف شبکه.
- **استفاده از حافظه هوشمند** :**(`std::shared_ptr`)** مدیریت خودکار منابع و جلوگیری از مشکلات مدیریت حافظه.
- **مدیریت ساده توپولوژی** :با استفاده از تابع `resetTopology`می‌توان توپولوژی موجود را پاکسازی کرد و توپولوژی جدیدی ساخت.
- **ساختار ماژولار** :توابع مجزا برای وظایف مختلف، از جمله ساخت توپولوژی و اتصال گره‌ها.

کلاس TopologyBuilder یک ابزار قدرتمند و ماژولار برای ساخت توپولوژی‌های شبکه است. با استفاده از این کلاس می‌توان توپولوژی‌های پایه‌ای شبکه را به راحتی پیاده‌سازی و مدیریت کرد.

کلاس topology controller

کلاس `TopologyController` مسئول مدیریت و کنترل توپولوژی شبکه است. این کلاس امکان افزودن، حذف، اتصال و مدیریت نودها (مانند روترها و کامپیوترها) را فراهم می‌کند. در ادامه، تحلیل عملکرد و قابلیت‌های این کلاس آورده شده است:

خصوصیات کلی:

1. **وابستگی به `TopologyBuilder`:**
   - این کلاس از یک شیء `TopologyBuilder` استفاده می‌کند که شامل ساختار توپولوژی شبکه است.

- قبل از اجرای بیشتر عملیات‌ها، بررسی می‌شود که آیا شیء `builder` مقداردهی شده است یا خیر.

سازنده‌ها و مخرب:

1. **سازنده پیش‌فرض:**
   - کلاس با مقداردهی `builder`به `nullptr`آغاز می‌شود.
2. **مخرب:**
   - مخرب استاندارد تعریف شده است اما عملیات خاصی انجام نمی‌دهد.

عملکردها:

1. setBuilder

برای تنظیم شیء TopologyBuilder استفاده می‌شود.

وابستگی اصلی کلاس به builder از طریق این تابع کنترل می‌شود.

2. addRouter

یک روتر جدید ایجاد کرده و به لیست روترهای توپولوژی اضافه می‌کند.

اگر builder مقداردهی نشده باشد، پیغام خطا چاپ می‌شود.

3. addPC

یک PC جدید به لیست کامپیوترهای توپولوژی اضافه می‌کند.

مشابهaddRouter ، بررسی می‌کند که آیا builder مقداردهی شده است.

4. removeNode

نودی با شناسه مشخص (ID) را از توپولوژی حذف می‌کند.

نود می‌تواند روتر یا کامپیوتر باشد.

از توابع استاندارد STL مثل remove_if برای حذف نودها استفاده می‌کند.

5. updateConnection

ارتباط بین دو نود مشخص را به‌روزرسانی یا ایجاد می‌کند.

ابتدا وجود هر دو نود بررسی می‌شود، سپس از متد connectNodes در TopologyBuilder برای اتصال آن‌ها استفاده می‌شود.

6. getRouterCount و getPCCount

به ترتیب تعداد روترها و کامپیوترها را بازمی‌گردانند.

اگر builder مقداردهی نشده باشد، مقدار 0 بازمی‌گردد.

7. isConnected

بررسی می‌کند که آیا دو نود مشخص به هم متصل هستند یا خیر.

با بررسی پورت‌های نود اول، ارتباط با نود دوم را تایید می‌کند.

8. printTopology

جزئیات توپولوژی فعلی شبکه را چاپ می‌کند.

شامل اطلاعاتی درباره نودها و اتصالات آن‌ها است.

9. findNodeById

نودی با شناسه مشخص را جستجو کرده و بازمی‌گرداند.

جستجو در لیست روترها و کامپیوترها انجام می‌شود.

نقاط قوت:

1. **ساختار منظم و خوانا:**
   - متدها به خوبی تفکیک شده‌اند و وظایف مشخصی دارند.
2. **بررسی‌های ایمنی:**
   - در بیشتر متدها، وجود `builder` بررسی می‌شود تا از بروز خطا جلوگیری شود.
3. **انعطاف‌پذیری:**

- این کلاس از `shared_ptr` برای مدیریت نودها استفاده می‌کند، که مدیریت حافظه را آسان‌تر می‌کند.

نقاط ضعف یا بهبودهای پیشنهادی:

مقداردهی پیش‌فرض:builder

اگر builder مقداردهی نشده باشد، بسیاری از متدها اجرا نمی‌شوند. بهتر است خطاهای مربوط به این وضعیت از طریق استثناها (exceptions) مدیریت شود.

اطلاعات بیشتر در خروجی خطا:

در متدهایی که پیام خطا چاپ می‌کنند، می‌توان اطلاعات بیشتری برای کمک به اشکال‌زدایی ارائه داد.

عدم بررسی اتصال‌های تکراری:

در updateConnection بررسی نمی‌شود که آیا اتصال از قبل وجود دارد یا خیر.

بهینه‌سازی جستجوی نودها:

findNodeById در لیست روترها و کامپیوترها به صورت خطی جستجو می‌کند. استفاده از یک ساختار داده مثل unordered_map می‌تواند سرعت جستجو را بهبود بخشد.