Simulador de Redes Quânticas Baseado em Camadas

Nome do projeto:
QuantumNet

Apresentado por:
David Tavares

Apresentado por:
Polyana Moraes

Agenda

01 Fundamentos da Computação Quântica

02 Desenvolvimento do Código

03 Funcionamento das Camadas

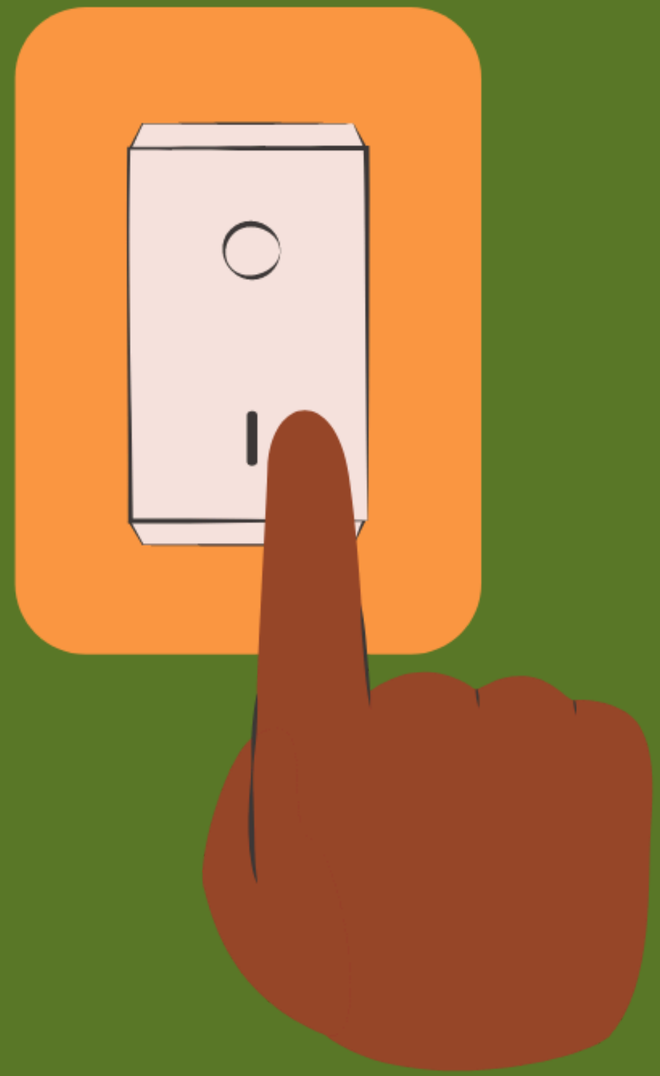
04 Métricas de Desempenho

05 Resultados da Simulação

06 Vantagens e Melhorias Futuras

Fundamentos das Redes Quânticas

- Internet Clássica vs Internet Quântica
- Superposição e Entrelaçamento
- Arquitetura Modular do Simulador



Desenvolvimento do Código

- Linguagem e Bibliotecas Utilizadas
  - Python
  - Networkx e Matplotlib
- Divisão do Simulador
  - Componentes
  - Camadas
  - Objetos

Camadas

Onde cada método e função é construído

Camada Física

Criação de qubits, medição de fidelidade e protocolos de emaranhamento

Camada de Enlace

Execução do emaranhamento e purificação para garantir a qualidade dos qubits

Camada de Rede

Encontra rotas eficientes e realiza entanglement swapping entre hosts

Camada de Transporte

Gerencia a transmissão confiável de qubits usando o protocolo de teletransporte

Camada de Aplicação

Implementa protocolos como a Distribuição Quântica de Chaves (QKD)

Camada Física

- Importação das dependências
- Instanciação da rede
- Configuração da topologia
- Ativação do sistema de log

Guia da Camada Física

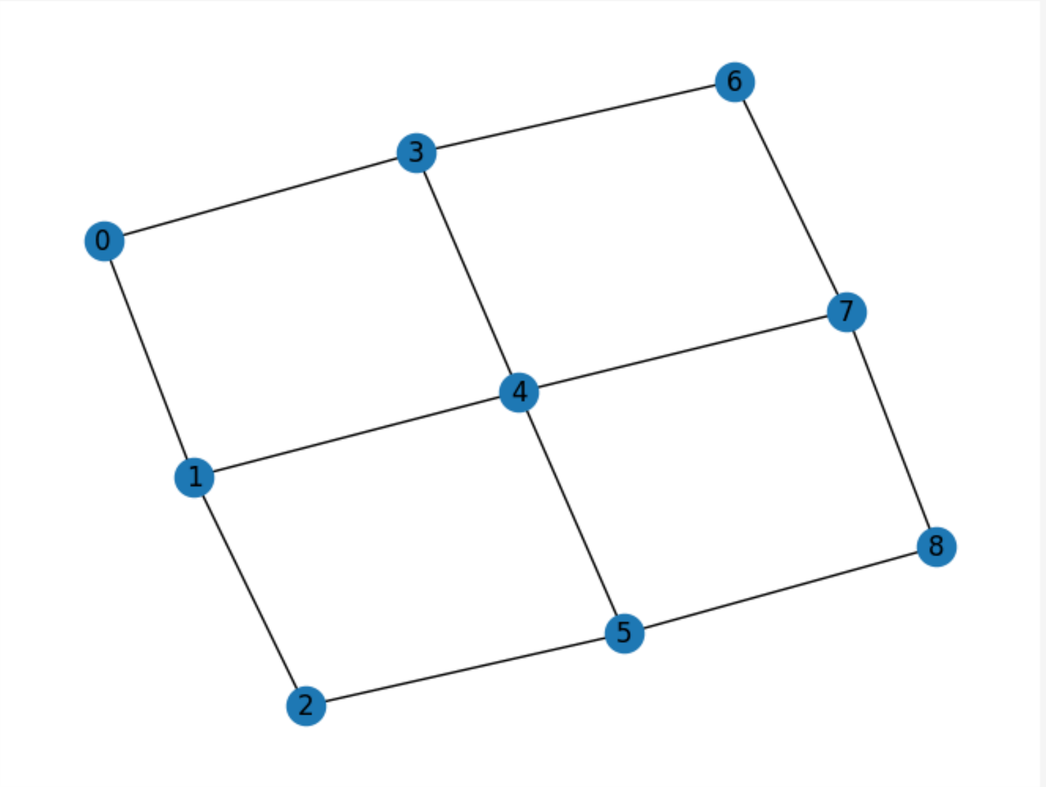
O documento tem o objetivo de demostrar o funcionamento da camada física, além de suas funções e ferramentas.

[1]:
```
from quantumnet.components import Network, Host
from quantumnet.objects import Qubit, Logger
import random
```

Será definida a topologia de Grade na rede.

[2]:
```
rede = Network()
rede.set_ready_topology("Grade", 3,3)
rede.draw()
Logger.activate(Logger)
```

```
Hosts inicializados
Canais inicializados
Pares EPRs adicionados
```



Camada Física

- Seleção aleatória de uma aresta
- Alocação de qubits aos hosts
- Impressão da memória dos hosts

Selecionando aleatoriamente uma aresta do grafo da rede e deseguinando quem será Alice e Bob.

[3]:
```
arestas = list(rede.edges)
aresta_aleatorias = random.sample(arestas, 1)[0]
x,y = aresta_aleatorias

alice = rede.get_host(x)
bob = rede.get_host(y)
```

Adicionando qubits aos hosts.

[4]:
```
h1 = Qubit(1)
h2 = Qubit(2)

alice.add_qubit(h1)
bob.add_qubit(h2)
```

```
2024-09-17 00:42:41,195: Qubit 1 adicionado à memória do Host 0.
2024-09-17 00:42:41,197: Qubit 2 adicionado à memória do Host 1.
```

Acessando a memória dos Hosts.

[5]:
```
print(alice.memory)
print(bob.memory)
```

```
[<quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA090>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA0D0
>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA110>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA15
0>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA190>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA2
10>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA250>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA
290>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA2D0>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECC
A1D0>, <quantumnet.objects.qubit.Qubit object at 0x0000018E9DDC9850>]
[<quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA310>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA350
>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA390>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA3D
0>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA410>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA4
90>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA4D0>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA
510>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECCA550>, <quantumnet.objects.qubit.Qubit object at 0x0000018E8ECC
A450>, <quantumnet.objects.qubit.Qubit object at 0x0000018E9F036550>]
```

Camada Física

- Medição da fidelidade de um qubit
- Execução do protocolo de entrelaçamento
- Reavaliação da fidelidade
- Correção em caso de falha no entrelaçamento

Medindo a fidelidade de um qubit.

In [8]:
```
#Pode se medir a fidelidade e acessar de somente um host
rede.physical.fidelity_measurement_only_one(h1)
```

```
2024-09-17 00:42:41,273: A fidelidade do qubit Qubit 1 with state 0 é 0.9232708011510475
```

Out[8]: 0.9232708011510475

Protocolo de criação de entrelaçamento entre dois qubits.

In [9]:
```
rede.physical.entanglement_creation_heralding_protocol(alice,bob)
```

```
2024-09-17 00:42:41,293: Timeslot 1: Par epr criado com fidelidade 0.44242918766576184
2024-09-17 00:42:41,295: Timeslot 2: O protocolo de criação de emaranhamento foi bem sucedido, mas com fidelidade baixa.
```

Out[9]: False

Reavaliando a fidelidade de um qubit para comparar suas variações após a passagem de um timeslot.

In [10]:
```
#Pode se medir a fidelidade e acessar de somente um host
rede.physical.fidelity_measurement_only_one(h1)
```

```
2024-09-17 00:42:41,318: A fidelidade do qubit Qubit 1 with state 0 é 0.9140380931395371
```

Out[10]: 0.9140380931395371

Realiza-se o primeiro protocolo, on demand, que através da fidelidade inicial e da probabilidade de sucesso do on demand determina o sucesso do ECHP.

In [11]:
```
rede.physical.echp_on_demand(2,5)
```

```
2024-09-17 00:42:41,341: A fidelidade do qubit Qubit 29 with state 0 é 0.6114927782397923
2024-09-17 00:42:41,342: A fidelidade do qubit Qubit 59 with state 0 é 0.21210339030375075
2024-09-17 00:42:41,343: Timeslot 3: A probabilidade de sucesso do ECHP falhou.
```

Out[11]: False

Realiza-se, então, o segundo protocolo, on replay, que através da fidelidade inicial e da probabilidade de sucesso do on replay determina o sucesso do ECHP.

In [12]:
```
rede.physical.echp_on_replay(2,5)
```

Camada Enlace

Definindo os hosts alice e bob nos nós 1 e 2 respectivamente, e definindo os qubits que serão enviados.

In [3]:
```
alice = rede.get_host(1)
bob = rede.get_host(2)
qubit1 = Qubit(1)
qubit2 = Qubit(2)
```

Ao enviar uma solicitação(request) para a rede, se a fidelidade dos qubits for superior a 0.5, a rede aceita a conexão e envia a chave. Caso contrário, uma nova tentativa de entrelaçamento é realizada. Se a tentativa falhar novamente, a rede tenta purificar o entrelaçamento para recuperar a conexão.

In [4]:
```
rede.linklayer.request(1,2)
```

```
2024-09-17 00:43:07,815: Timeslot 1: Tentativa de emaranhamento entre 1 e 2.
2024-09-17 00:43:07,816: Timeslot 2: Par epr criado com fidelidade 0.1420104118319821
2024-09-17 00:43:07,817: Timeslot 3: O protocolo de criação de emaranhamento foi bem sucedido, mas com fidelidade baixa.
2024-09-17 00:43:07,819: Timeslot 3: Entrelaçamento falhou entre 1 e 2 na tentativa 1.
2024-09-17 00:43:07,820: Timeslot 4: Tentativa de emaranhamento entre 1 e 2.
2024-09-17 00:43:07,821: Timeslot 5: Par epr criado com fidelidade 0.01038252761282216
2024-09-17 00:43:07,822: Timeslot 6: O protocolo de criação de emaranhamento foi bem sucedido, mas com fidelidade baixa.
2024-09-17 00:43:07,823: Timeslot 6: Entrelaçamento falhou entre 1 e 2 na tentativa 2.
2024-09-17 00:43:07,824: A purificação utilizada foi tipo 1.
2024-09-17 00:43:07,825: Timeslot 7: Purificação falhou no canal (1, 2) devido a baixa fidelidade após purificação.
```

Camada de Rede

- Busca de rota válida
- Entanglement swapping

Definindo o host 6 como Alice e o host 1 como Bob.

In [3]:
```
alice = rede.get_host(6)
bob = rede.get_host(1)
```

Nesse sentido, primeiramente precisa-se escolher a melhor rota e seus critérios são dados pela rota com menor caminho. Para verificar isso usaremos a função short_route_valid, que irá entregar uma rota válida entre Alice e Bob. Essa rota válida deve ter pelo menos 1 par EPR e se nó tem 2 qubits.

In [4]:
```
rede.networklayer.short_route_valid(8,0)
```

```
2024-09-17 00:43:49,429: Timeslot 1: Buscando rota válida entre 8 e 0.
2024-09-17 00:43:49,432: Rota válida encontrada: [8, 5, 2, 1, 0]
```

Out[4]: [8, 5, 2, 1, 0]

Dessa forma, vamos realizar o entanglement swapping entre Alice e Bob, o que permite criar um par de qubits emaranhados entre dois nós que não têm uma conexão direta de emaranhamento, utilizando pares intermediários.

In [5]:
```
rede.networklayer.entanglement_swapping(8,0)
```

```
2024-09-17 00:43:49,447: Timeslot 2: Buscando rota válida entre 8 e 0.
2024-09-17 00:43:49,449: Rota válida encontrada: [8, 5, 2, 1, 0]
2024-09-17 00:43:49,450: Timeslot 3: Realizando Entanglement Swapping.
2024-09-17 00:43:49,450: Par EPR <quantumnet.objects.epr.Epr object at 0x0000025C36AE2C10> adicionado ao canal (8, 2).
2024-09-17 00:43:49,452: Par EPR <quantumnet.objects.epr.Epr object at 0x0000025C36B0E250> removido do canal (8, 5).
2024-09-17 00:43:49,453: Par EPR <quantumnet.objects.epr.Epr object at 0x0000025C36B0D510> removido do canal (5, 2).
2024-09-17 00:43:49,455: Timeslot 4: Realizando Entanglement Swapping.
2024-09-17 00:43:49,456: Par EPR <quantumnet.objects.epr.Epr object at 0x0000025C36B0D510> adicionado ao canal (8, 1).
2024-09-17 00:43:49,456: Par EPR <quantumnet.objects.epr.Epr object at 0x0000025C36AE2C10> removido do canal (8, 2).
2024-09-17 00:43:49,457: Par EPR <quantumnet.objects.epr.Epr object at 0x0000025C36B0D250> removido do canal (2, 1).
2024-09-17 00:43:49,457: Timeslot 5: Realizando Entanglement Swapping.
2024-09-17 00:43:49,459: Par EPR <quantumnet.objects.epr.Epr object at 0x0000025C36B0D250> adicionado ao canal (8, 0).
2024-09-17 00:43:49,460: Par EPR <quantumnet.objects.epr.Epr object at 0x0000025C36B0D510> removido do canal (8, 1).
2024-09-17 00:43:49,462: Par EPR <quantumnet.objects.epr.Epr object at 0x0000025C36B0CCD0> removido do canal (1, 0).
2024-09-17 00:43:49,463: Timeslot 6: Realizando Entanglement Swapping.
2024-09-17 00:43:49,465: Entanglement Swapping concluído com sucesso entre 8 e 0
```

Out[5]: True

Camada de Transporte

- Busca de rota
- Criação de qubit
- Teletransporte de qubit

A função a seguir é chamada para que seja realizada a requesição de uma rota para que haja a transmissão de n qubits necessários. Ela vai executar toda a camada de transporte.

In [6]:
```
rede.transportlayer.run_transport_layer(0,8,11)
```

```
2024-09-17 00:44:24,179: Número insuficiente de qubits na memória de Alice (Host 0). Criando mais 1 qubits para completar os 11 necessários.
2024-09-17 00:44:24,180: Timeslot antes da criação do qubit: 1
2024-09-17 00:44:24,182: Qubit 90 adicionado à memória do Host 0.
2024-09-17 00:44:24,183: Qubit 90 criado com fidelidade inicial 0.7620490470903759 e adicionado à memória do Host 0.
2024-09-17 00:44:24,184: Qubit criado para Alice (Host 0) no timeslot: 2
2024-09-17 00:44:24,185: Tentativa 1 de transmissão de qubits entre 0 e 8.
2024-09-17 00:44:24,185: Timeslot 3: Buscando rota válida entre 0 e 8.
2024-09-17 00:44:24,188: Rota válida encontrada: [0, 3, 6, 7, 8]
2024-09-17 00:44:24,189: Teletransporte de qubit de 0 para 8 na rota [0, 3, 6, 7, 8] foi bem-sucedido com fidelidade final de 0.2363687492145497.
2024-09-17 00:44:24,190: Timeslot 4: Buscando rota válida entre 0 e 8.
2024-09-17 00:44:24,191: Rota válida encontrada: [0, 3, 6, 7, 8]
2024-09-17 00:44:24,192: Teletransporte de qubit de 0 para 8 na rota [0, 3, 6, 7, 8] foi bem-sucedido com fidelidade final de 0.3671543825919449.
2024-09-17 00:44:24,193: Timeslot 5: Buscando rota válida entre 0 e 8.
2024-09-17 00:44:24,194: Rota válida encontrada: [0, 3, 6, 7, 8]
2024-09-17 00:44:24,195: Teletransporte de qubit de 0 para 8 na rota [0, 3, 6, 7, 8] foi bem-sucedido com fidelidade final de 0.01887270001414548.
2024-09-17 00:44:24,197: Timeslot 6: Buscando rota válida entre 0 e 8.
2024-09-17 00:44:24,198: Rota válida encontrada: [0, 3, 6, 7, 8]
2024-09-17 00:44:24,199: Teletransporte de qubit de 0 para 8 na rota [0, 3, 6, 7, 8] foi bem-sucedido com fidelidade final de 0.022769993875717015.
2024-09-17 00:44:24,200: Timeslot 7: Buscando rota válida entre 0 e 8.
2024-09-17 00:44:24,201: Rota válida encontrada: [0, 3, 6, 7, 8]
2024-09-17 00:44:24,201: Teletransporte de qubit de 0 para 8 na rota [0, 3, 6, 7, 8] foi bem-sucedido com fidelidade final de 0.028604161990649753.
2024-09-17 00:44:24,202: Timeslot 8: Buscando rota válida entre 0 e 8.
2024-09-17 00:44:24,203: Rota válida encontrada: [0, 3, 6, 7, 8]
2024-09-17 00:44:24,204: Teletransporte de qubit de 0 para 8 na rota [0, 3, 6, 7, 8] foi bem-sucedido com fidelidade final de 0.07328641964481597.
2024-09-17 00:44:24,204: Timeslot 9: Buscando rota válida entre 0 e 8.
2024-09-17 00:44:24,205: Rota válida encontrada: [0, 3, 6, 7, 8]
2024-09-17 00:44:24,206: Teletransporte de qubit de 0 para 8 na rota [0, 3, 6, 7, 8] foi bem-sucedido com fidelidade final de 0.13541722476705548.
2024-09-17 00:44:24,206: Timeslot 10: Buscando rota válida entre 0 e 8.
2024-09-17 00:44:24,207: Rota válida encontrada: [0, 3, 6, 7, 8]
2024-09-17 00:44:24,207: Teletransporte de qubit de 0 para 8 na rota [0, 3, 6, 7, 8] foi bem-sucedido com fidelidade final de 0.039725430982378566.
2024-09-17 00:44:24,208: Timeslot 11: Buscando rota válida entre 0 e 8.
2024-09-17 00:44:24,208: Rota válida encontrada: [0, 3, 6, 7, 8]
2024-09-17 00:44:24,209: Teletransporte de qubit de 0 para 8 na rota [0, 3, 6, 7, 8] foi bem-sucedido com fidelidade final de 0.017002676473893048.
2024-09-17 00:44:24,209: Timeslot 12: Buscando rota válida entre 0 e 8.
2024-09-17 00:44:24,212: Rota válida encontrada: [0, 3, 6, 7, 8]
2024-09-17 00:44:24,214: Teletransporte de qubit de 0 para 8 na rota [0, 3, 6, 7, 8] foi bem-sucedido com fidelidade final de 0.010750088606970633.
2024-09-17 00:44:24,214: Timeslot 13: Buscando rota válida entre 0 e 8.
2024-09-17 00:44:24,215: Rota válida encontrada: [0, 3, 6, 7, 8]
2024-09-17 00:44:24,216: Teletransporte de qubit de 0 para 8 na rota [0, 3, 6, 7, 8] foi bem-sucedido com fidelidade final de 0.060785941936519455.
2024-09-17 00:44:24,217: Transmissão e teletransporte de 11 qubits entre 0 e 8 concluídos com sucesso. Timeslot: 13
```

Out[6]: True

Camada de aplicação

A camada de aplicação coordena e utiliza todas as camadas da rede

Por meio da chamada da função qks_e91_protocol, passando os argumentos do ID de Alice, ID de Bob e o número de chaves que é pedido nessa requisição. Houve a transmissão das chaves.

In [4]:
```
rede.application_layer.run_app("QKD_E91",3,8,5)
```

```
2024-09-20 11:19:44,291: Transmissão e teletransporte de 5 qubits entre 3 e 8 concluídos com sucesso. Timeslot: 51
2024-09-20 11:19:44,294: Timeslot incrementado após transmissão: 52
2024-09-20 11:19:44,298: Chaves obtidas até agora: [0, 0, 1, 1, 1]
2024-09-20 11:19:44,303: Protocolo E91 bem-sucedido. Chave final compartilhada: [0, 0, 1, 1, 1]
```

Métricas

Pode-se setar as métricas, escolher seu nome e o jeito que quer a saída

Pode-se gerar um arquivo em csv, print ou variável para setar métricas escolhidas pelo próprio usuário.

In [15]:
```
rede.get_metrics(file_name="teste.csv")
```

```
# Dicionário com todas as métricas possíveis
available_metrics = {
    "Timeslot Total": self.get_timeslot(),
    "EPRs Usados": self.get_total_useds_eprs(),
    "Qubits Usados": self.get_total_useds_qubits(),
    "Fidelidade na Camada de Transporte": self.transportlayer.avg_fidelity_on_transportlayer(),
    "Fidelidade na Camada de Enlace": self.linklayer.avg_fidelity_on_linklayer(),
    "Média de Rotas": self.networklayer.get_avg_size_routes()
```

Conclusão

Flexibilidade e Eficiência

oferece uma solução modular em camadas

Aplicações e Resultados

facilidade no teste de topologias e protocolos

Futuro Promissor

melhorias como a adição de uma interface gráfica e maior otimização

Obrigado!

Código Github