

The Communication Model of Centralized Quantum Internet - Pseudocode

A PSEUDOCODE OF PROTOCOLS

Algorithm: Entanglement Preparation Control

```

input :Path
output:entanglement preparation result
// Path: completed communication path
//  $T_{csm}$ : central state matrix
//  $T_{lsm}$ : local state matrix
// LC: local domain controller
1 for  $LC \in Path$  do
2   announce to prepare w-state entanglement;
3   if  $reply_{preparation} == "success"$  then
4     update( $T_{csm}, T_{lsm}$ );
5     start entanglement distribution process;
6   else if  $retry \leq retry_{max}$  then
7     update( $T_{csm}, T_{lsm}, LinkState$ );
8      $retry++$ ;
9     restart entanglement preparation process;
10  else
11    update( $T_{csm}, T_{lsm}, DeviceState, "maintain"$ );
12    if not intra-domain-communication then
13      restart entanglement routing process;
14    else announce communication failed;

```

Algorithm: Resource Check & Reservation

```

input :  $Path_{middle}$ 
output: Path
// rm: memory name of repeater
// cm: memory name of local domain controller
//  $Path_{middle}$ : middle-state path given by entanglement routing
1 for  $device \in Path_{middle}$  do
2    $dstate = \text{CheckDeviceState}(T_{csm}, device);$ 
3    $mstate, rm/cm = \text{FindMem}(T_{csm}, device);$ 
4   if  $dstate == \text{"normal"} \ \&\& \ mstate == \text{"idle"}$  then
5      $reply = \text{ReserveResource}(device, rm/cm);$ 
6     if  $reply == \text{true}$  then
7        $\text{update}(T_{csm}, T_{lsm}, \text{MemoryState});$ 
8        $Path \leftarrow \text{updatepath}(device, rm/cm);$ 
9     else
10       $result_{reservation} = \text{"failed"};$ 
11      break;
12   else
13      $result_{reservation} = \text{"failed"};$ 
14     break;
15 if  $result_{reservation} \neq \text{"failed"}$  then
16   return Path;
17 else
18    $\text{recover } T_{csm}, T_{lsm};$ 
19   return null;

```

Algorithm: Entanglement Distribution Control

```

input :  $Path$ 
output: entanglement distribution result
//  $t_d$ : entanglement distribution timer
1 for  $repeater/user \in Path$  do
2   if  $reply_{distribution} == "success"$  then
3      $\_update(T_{csm}, T_{lsm}, LinkState);$ 
4   else
5      $\_update(T_{csm}, T_{lsm}, LinkState);$ 
6      $result_{distribution} = "false"$ 
7 if  $result_{distribution} != "false" \ \&\& \ t_d \leq t_{max}$  then
8    $\_update(T_{csm}, T_{lsm});$ 
9   if not intra-domain-communication then
10     $\_start \text{ entanglement swapping process};$ 
11   else
12     $\_start \text{ quantum teleportation process};$ 
13 else if  $retry \leq retry_{max}$  then
14    $\_reset \ t_d \text{ and memories};$ 
15    $retry ++;$ 
16    $\_retry \text{ entanglement preparation \& distribution};$ 
17 else
18   for failed repeaters/users do
19      $\_update(T_{csm}, T_{lsm}, DeviceState, "maintain");$ 
20   if not intra-domain-communication then
21      $\_restart \text{ entanglement routing process};$ 
22   else  $\_announce \text{ communication failed};$ 

```

Algorithm: Centralized Entanglement Routing

```

input :  $U_{src}, U_{dst}$ 
output: optimal  $Path_{middle}$ 
//  $T_{dsp}$ : domain shortest path table
//  $T_{der}$ : domain edge repeater table
//  $N$ : recursion number
//  $Path_{pr}$ : paths of previous round
//  $Path_{cr}$ : paths of current round
1  $D_{src}, D_{dst} = \text{FindDomain}(T_{csm}, U_{src}, U_{dst});$ 
2  $Path_{array} = \text{FindPaths}(T_{dsp}, T_{der}, D_{src}, D_{dst});$ 
3  $Path_{pr} \leftarrow Path_{array};$ 
4 while  $N > 0$  do
5   for  $path \in Path_{pr}$  do
6     for  $repeater \in path$  do
7       nodes =  $\text{FindReplaceNodes}(T_{csm});$ 
8       if nodes is not none then
9          $Path_{cr} \leftarrow \text{AddNewPath};$ 
10    if  $Path_{cr}$  is none then
11      break
12     $Path_{array} \leftarrow Path_{cr};$ 
13     $\text{Clear}(Path_{pr});$ 
14     $Path_{pr} \leftarrow Path_{cr};$ 
15     $N = N - 1;$ 
16  $\text{EliminateInvalidPath}(Path_{array});$ 
17 for  $path \in Path_{array}$  do
18   for  $repeater \in path$  do
19      $\text{SwappingSuccessRate} = \text{Result}(T_{csm});$ 
20      $\text{LinkState} = \text{Result}(T_{csm});$ 
21      $\text{Score}_R = \text{SwappingSuccessRate} \times 0.3 + \text{LinkState} \times 0.7;$ 
22      $\text{Score}_P = \text{Score}_P + \text{Score}_R;$ 
23    $\text{Score}_P = \frac{\text{Score}_P}{\text{hops}};$ 
24    $\text{Score}_{array} \leftarrow \text{Score}_P;$ 
25  $\text{sortpaths} = \text{Sort}(\text{Score}_{array}, Path_{array}, \text{hops});$ 
26 while true do
27    $Path_{middle} = \text{sortpaths.pop}(0);$ 
28    $\text{Path} = \text{ResourceCheck\&Reserve}(Path_{middle});$ 
29   if  $\text{Path} \neq \text{null}$  then
30     start entanglement preparation process;
31     break
32   else if  $\text{sortpaths} == \text{null}$  then
33     announce no path found;
34     announce communication failed;
35     break

```

Algorithm: Entanglement Swapping Control

```

input :Path
output:entanglement swapping result
//  $t_{st}$ : swapping & teleportation timer
1 for  $repeater \in Path$  do
2   announce to perform entanglement swapping;
3   if  $reply_{swapping} == "success" \ \&\& \ t_{st} < t_{max}$  then
4     | update( $T_{csm}, T_{lsm}, SwappingSuccessRate$ );
5   else
6     | update( $T_{csm}, T_{lsm}, SwappingSuccessRate$ );
7     |  $result_{swapping} = "false"$ ;
8     | break
9 if  $result_{swapping} != "false"$  then
10  | update( $T_{csm}, T_{lsm}$ );
11  | start quantum teleportation process;
12 else if  $retry \leq retry_{max}$  then
13  | reset  $t_{st}$  and memories;
14  |  $retry++$ ;
15  | retry entanglement preparation & distribution;
16 else
17  | for failed repeater do
18  | | update( $T_{csm}, T_{lsm}, DeviceState, "maintain"$ );
19  | restart entanglement routing process;

```

Algorithm: End-End Communication Request

```

input :  $U_{src}, request_{user}$ 
output: communication request result
// t: communication request timer
1  $U_{src}$  send  $request_{user}$  to local domain controller;
  // in the local domain controller
2 if  $request_{user}.U_{dst} \in T_{lsm}$  then
  // intra-domain-communication
  3 forward  $request_{user}$  to  $U_{dst}$ ;
  4 if  $reply_{user} == "accept"$  &&  $t \leq t_{max}$  then
  5    $mstate, cm_1, cm_2 = \text{FindMem}(T_{lsm}, LC)$ ;
  6    $um_{src} = request_{user}.um$ ;
  7    $um_{dst} = reply_{user}.um$ ;
  8   if  $mstate == "idle"$  then
  9     ReserveResource( $um/cm$ );
 10     update( $T_{csm}, T_{lsm}, \text{MemoryState}$ );
 11     Path =  $U_{src}[LC(cm_1, cm_2), um_{src}] \rightarrow$ 
         $U_{dst}[LC(cm_1, cm_2), um_{dst}]$ ;
 12     start entanglement preparation process;
 13   else announce communication failed;
14 else announce communication failed;
15 else
  // inter-domain-communication
16 forward  $request_{user}$  to central controller;
  // in the central controller
17 if  $request_{user}.U_{dst} \in T_{csm}$  then
18 forward  $request_{user}$  to  $U_{dst}$ ;
19 if  $reply_{user} == "accept"$  &&  $t \leq t_{max}$  then
20    $um_{src} = request_{user}.um$ ;
21    $um_{dst} = reply_{user}.um$ ;
22   ReserveResource( $um$ );
23   update( $T_{csm}, T_{lsm}, \text{MemoryState}$ );
24   start entanglement routing process;
25 else announce communication failed;
26 else announce  $U_{dst}$  not found;

```

Algorithm: Quantum Teleportation Control

```

input :  $U_{src}, U_{dst}$ 
output: quantum teleportation result
//  $result_{BSM}$ : bell measurement result
1 if  $t_{st} \leq t_{max}$  then
2   announce  $U_{src}$  to exec Bell State Measurement;
3   if  $reply_{BSM} == "success"$  then
4      $U_{src}$  forward  $Result_{BSM}$  to  $U_{dst}$ ;
5      $U_{dst}$  do qubit correction with  $Result_{BSM}$ ;
6     if  $reply_{correction} == "success"$  then
7       announce communication success;
8       update( $T_{csm}, T_{lsm}$ );
9       reset  $T_{csm}, T_{lsm}$ , and all related devices;
10    else announce communication failed;
11  else announce communicate failed;
12 else if  $retry \leq retry_{max}$  then
13   reset  $t_{st}$  and memories;
14    $retry++$ ;
15   retry entanglement preparation & distribution;
16 else restart entanglement routing process;

```
