The Communication Model of Hierarchical 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
for $LC \in Path$ do
    announce to prepare w-state entanglement;
    if $reply_{preparation}$ == "success" then
        update($T_{csm}$, $T_{lsm}$);
        start entanglement distribution process;
    else if $retry <= retry_{max}$ then
        update($T_{csm}$, $T_{lsm}$, LinkState);
        retry ++;
        restart entanglement preparation process;
    else
        update($T_{csm}$, $T_{lsm}$, DeviceState, "maintain");
        if not intra-domain-communication then
            restart entanglement routing process;
        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
for $device \in Path_{middle}$ do
    dstate = CheckDeviceState($T_{csm}$, device);
    mstate, rm/cm = FindMem($T_{csm}$, device);
    if dstate == "normal" && mstate == "idle" then
        reply = ReserveResource(device, rm/cm);
        if reply == true then
            update($T_{csm}$, $T_{lsm}$, MemoryState);
            Path ⟵ updatepath(device, rm/cm);
        else
            $result_{reservation}$ = "failed";
            break;
    else
        $result_{reservation}$ = "failed";
        break;
if $result_{reservation}$ != "failed" then
    return Path;
else
    recover $T_{csm}$, $T_{lsm}$;
    return null;
```

Algorithm: Entanglement Distribution Control

```
input  : $Path$
output : entanglement distribution result
// $t_d$: entanglement distribution timer
for repeater/user ∈ Path do
    if $reply_{distribution}$ == "success" then
        update($T_{csm}$, $T_{lsm}$, LinkState);
    else
        update($T_{csm}$, $T_{lsm}$, LinkState);
        $result_{distribution}$ = "false"
if $result_{distribution}$ != "false" && $t_d$ <= $t_{max}$ then
    update($T_{csm}$, $T_{lsm}$);
    if not intra-domain-communication then
        start entanglement swapping process;
    else
        start quantum teleportation process;
else if retry <= $retry_{max}$ then
    reset $t_d$ and memories;
    retry ++;
    retry entanglement preparation & distribution;
else
    for failed repeaters/users do
        update($T_{csm}$, $T_{lsm}$, DeviceState, "maintain");
    if not intra-domain-communication then
        restart entanglement routing process;
    else announce 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
$D_{src}$, $D_{dst}$ = FindDomain($T_{csm}$, $U_{src}$, $U_{dst}$);
$Path_{array}$ = FindPaths($T_{dsp}$, $T_{der}$, $D_{src}$, $D_{dst}$);
$Path_{pr} \longleftarrow Path_{array}$;
while $N > 0$ do
    for $path \in Path_{pr}$ do
        for $repeater \in path$ do
            nodes = FindReplaceNodes($T_{csm}$);
            if nodes is not none then
                $Path_{cr} \longleftarrow$ AddNewPath;
    if $Path_{cr}$ is none then
        break
    $Path_{array} \longleftarrow Path_{cr}$;
    Clear($Path_{pr}$);
    $Path_{pr} \longleftarrow Path_{cr}$;
    N = N - 1;
EliminateInvalidPath($Path_{array}$);
for $path \in Path_{array}$ do
    for $repeater \in path$ do
        SwappingSuccessRate = Result($T_{csm}$);
        LinkState = Result($T_{csm}$);
        $Score_R$ = SwappingSuccessRate × 0.3 + LinkState × 0.7;
        $Score_P = Score_P + Score_R$;
    $Score_P = \frac{Score_P}{hops}$;
    $Score_{array} \longleftarrow Score_P$;
sortpaths = Sort($Score_{array}$, $Path_{array}$, hops);
while true do
    $Path_{middle}$ = sortpaths.pop(0);
    Path = ResourceCheck&Reserve($Path_{middle}$);
    if Path != null then
        start entanglement preparation process;
        break
    else if sortpaths == null then
        announce no path found;
        announce communication failed;
        break
```

Algorithm: Entanglement Swapping Control

```
input  : Path
output : entanglement swapping result
// $t_{st}$: swapping & teleportation timer
for $repeater \in Path$ do
    announce to perform entanglement swapping;
    if $reply_{swapping}$ == "success" && $t_{st} < t_{max}$ then
        update($T_{csm}$, $T_{lsm}$, SwappingSuccessRate);
    else
        update($T_{csm}$, $T_{lsm}$, SwappingSuccessRate);
        $result_{swapping}$ = "false";
        break
if $result_{swapping}$ != "false" then
    update($T_{csm}$, $T_{lsm}$);
    start quantum teleportation process;
else if $retry <= retry_{max}$ then
    reset $t_{st}$ and memories;
    retry ++;
    retry entanglement preparation & distribution;
else
    for failed repeater do
        update($T_{csm}$, $T_{lsm}$, DeviceState, “maintain”);
    restart entanglement routing process;
```

```
Algorithm: End-End Communication Request
    input  : $U_{src}$, $request_{user}$
    output : communication request result
    // t: communication request timer
    $U_{src}$ send $request_{user}$ to local domain controller;
    // in the local domain controller
    if $request_{user}.U_{dst} \in T_{lsm}$ then
        // intra-domain-communication
        forward $request_{user}$ to $U_{dst}$;
        if $reply_{user}$ == "accept" && $t <= t_{max}$ then
            mstate, $cm_1$, $cm_2$ = FindMem($T_{lsm}$, LC);
            $um_{src}$ = $request_{user}$.um;
            $um_{dst}$ = $reply_{user}$.um;
            if $mstate$ == "idle" then
                ReserveResource(um/cm);
                update($T_{csm}$, $T_{lsm}$, MemoryState);
                Path = $U_{src}$[LC($cm_1$,$cm_2$),$um_{src}$] $\longrightarrow$
                        $U_{dst}$[LC($cm_1$,$cm_2$),$um_{dst}$] ;
                start entanglement preparation process;
            else announce communication failed;
        else announce communication failed;
    else
        // inter-domain-communication
        forward $request_{user}$ to central controller;
    // in the central controller
    if $request_{user}.U_{dst} \in T_{csm}$ then
        forward $request_{user}$ to $U_{dst}$;
        if $reply_{user}$ == "accept" && $t <= t_{max}$ then
            $um_{src}$ = $request_{user}$.um;
            $um_{dst}$ = $reply_{user}$.um;
            ReserveResource(um);
            update($T_{csm}$, $T_{lsm}$, MemoryState);
            start entanglement routing process;
        else announce communication failed;
    else announce $U_{dst}$ not found;
```

Algorithm: Quantum Teleportation Control

```
input  : $U_{src}$, $U_{dst}$
output : quantum teleportation result
// $result_{BSM}$: bell measurement result
if $t_{st} <= t_{max}$ then
    announce $U_{src}$ to exec Bell State Measurement;
    if $reply_{BSM}$ == "success" then
        $U_{src}$ forward $Result_{BSM}$ to $U_{dst}$;
        $U_{dst}$ do qubit correction with $Result_{BSM}$;
        if $reply_{correction}$ == "success" then
            announce communication success;
            update($T_{csm}$, $T_{lsm}$);
            reset $T_{csm}$, $T_{lsm}$, and all related devices;
        else announce communication failed;
    else announce communicate failed;
else if $retry <= retry_{max}$ then
    reset $t_{st}$ and memories;
    retry ++;
    retry entanglement preparation & distribution;
else restart entanglement routing process;
```