Commit 1：initial and studentID

Commit 2：finish all tasks

Commit 3：finish report

Task 1.1

In nix-vector-helper.h

```
34  #include <string>
```

(in public)

- Add member variable m_pathFile • Add member function SetPathFile()

```
67    // 設定用戶自定義路徑檔案
68    void SetPathFile (std::string pathFile);
69    std::string   m_pathFile;     // NEW: 儲存輸入的路徑檔案
```

In nix-vector-helper.cc

```
30  #include <string>
```

- Update copy constructor to also assign m_pathFile when constructing new NixVectorHelper by copy

```
42  template <typename T>
43  NixVectorHelper<T>::NixVectorHelper (const NixVectorHelper<T> &o)
44    : m_agentFactory (o.m_agentFactory)
45    , m_pathFile    (o.m_pathFile)   // NEW: copy path filename//////////////////////////
46  {
47    // Check if the T is Ipv4RoutingHelper or Ipv6RoutingHelper.
48    NS_ASSERT_MSG ((IsIpv4::value || std::is_same <Ipv6RoutingHelper, T>::value),
49                   "Template parameter is not Ipv4RoutingHelper or Ipv6Routing Helper");
50  }
```

- Implement SetPathFile(): update m_pathfile

```
59  ////////////////////////////////////
60  // NEW: 存取路徑檔案
61  template <typename T>
62  void
63  NixVectorHelper<T>::SetPathFile (std::string pathFile)
64  {
65    m_pathFile = pathFile;
66  }
67  ////////////////////////////////////
```

• Call SetPaths in Create function after creating NixVectorRouting object

```
69  template <typename T>
70  Ptr<typename NixVectorHelper<T>::IpRoutingProtocol>
71  NixVectorHelper<T>::Create (Ptr<Node> node) const
72  {
73    Ptr<NixVectorRouting<IpRoutingProtocol>> agent = m_agentFactory
74    agent->SetNode (node);
75
76    ////////////////////////////////////
77    // NEW: 如果 helper 裡有設定 path file，呼叫路由物件的 SetPaths()
78    if (!m_pathFile.empty ())
79      {
80        agent->SetPaths (m_pathFile);
81      }
82    ////////////////////////////////////
83
84    node->AggregateObject (agent);
85    return agent;
86  }
```

In nix-vector-routing.h

```
49  #include <vector>          // NEW: 儲存 node 列表
50  #include <utility>         // NEW: std::pair
51  #include <string>          // NEW: std::string
```

(in public)

• Add member variable Table to store the paths • Add member function SetPaths(pathFile)

```
144   /** NEW: 由 helper 呼叫，從檔案讀取所有 SD-pair 路徑 *//////////////////////////
145   void SetPaths (const std::string &pathFile);
146   /** NEW: 路徑表: key=(src,dst), value=節點列表 */
147   std::map<std::pair<int,int>, std::vector<int>> m_table;
```

In nix-vector-routing.cc

```
38  #include <fstream>         // NEW: 讀檔
39  #include <sstream>         // NEW: 解析行
40  #include <ns3/log.h>       // NEW: 錯誤列印
41  #include <utility>         // for std::pair
```

• Implement SetPaths(pathFile): read the paths from the pathFile and store the paths into Table

用 ifstream 讀取 pathFile，逐行解析、存進暫存的 vector<int> path，最後一行把整條 path 塞到 m_table[{src,dst}]，再關閉檔案

```cpp
template <typename Protocol>
void
NixVectorRouting<Protocol>::SetPaths (const std::string &pathFile)
{
  std::ifstream file (pathFile);
  if (!file.is_open ())
    {
      NS_LOG_ERROR ("NixVectorRouting::SetPaths(): 無法開啟檔案 " << pathFile);
      return;
    }

  int src, dst, length;
  while (file >> src >> dst >> length)
    {
      std::vector<int> path;
      path.reserve (length);
      for (int i = 0; i < length; ++i)
        {
          int hop;
          file >> hop;
          path.push_back (hop);
        }
      m_table[std::make_pair (src, dst)] = path;
    }
  file.close ();
}
```

Task 1.2

「m_table」是一張從 (srcId, dstId) 到「完整 hop 序列」的映射。

先看 m_table.find(key) 是否命中，命中就不做 BFS。

parentVector 是 Nix-Vector 需要的格式：長度 = 總節點數，parentVector[j] 存放抵達節點 j 時的前驅 Ptr<Node>。

BuildNixVector() 會以這張 parentVector 逆向重組 source→dest 的路徑向量（neighbor index），最後塞進 nixVector。

若 table 沒命中，才「fallback」到原本的自動 BFS 機制。

```cpp
template <typename T>
Ptr<NixVector>
NixVectorRouting<T>::GetNixVector (Ptr<Node> source, IpAddress dest, Ptr<NetDevice> oif) const
{
  NS_LOG_FUNCTION (this << source << dest << oif);

  Ptr<NixVector> nixVector = Create<NixVector> ();

  // ==== NEW: Task1.2 手動 path 查找 ====////////////////////////////////////
  // 1. 從 table 拿 srcId/dstId
  int srcId = source->GetId ();
  Ptr<Node> destNode = GetNodeByIp (dest);
  if (destNode == 0)
    {
      NS_LOG_ERROR ("No routing path exists");
      return 0;
    }
  int dstId = destNode->GetId ();
  auto key = std::make_pair (srcId, dstId);
  auto it = m_table.find (key);
```

```
if (it != m_table.end ())
  {
    NS_LOG_LOGIC ("Use manual path for " << srcId << " -> " << dstId);
    // 把 vector<int> 轉成 parentVector
    std::vector< Ptr<Node> > parentVector;
    parentVector.assign (NodeList::GetNNodes (), 0);
    for (size_t i = 1; i < it->second.size (); ++i)
      {
        int prev = it->second[i-1];
        int curr = it->second[i];
        parentVector.at (curr) = NodeList::GetNode (prev);
      }
    // 交給 BuildNixVector 回傳 nixVector
    if (BuildNixVector (parentVector, srcId, dstId, nixVector))
      {
        return nixVector;
      }
    else
      {
        NS_LOG_ERROR ("BuildNixVector failed for manual path");
        return 0;
      }
  }
```

Task 1.3

把原本會回傳 in cache 的部分都註解掉

```
template <typename T>
Ptr<NixVector>
NixVectorRouting<T>::GetNixVectorInCache (const IpAddress &address, bool &foundInCache) const
{
  NS_LOG_FUNCTION (this << address);

  CheckCacheStateAndFlush ();

  // typename NixMap_t::iterator iter = m_nixCache.find (address);
  // if (iter != m_nixCache.end ())
  //   {
  //     NS_LOG_LOGIC ("Found Nix-vector in cache.");
  //     foundInCache = true;
  //     return iter->second;
  //   }

  // not in cache
  foundInCache = false;
  return 0;
}
```

```
template <typename T>
Ptr<typename NixVectorRouting<T>::IpRoute>
NixVectorRouting<T>::GetIpRouteInCache (IpAddress address)
{
  NS_LOG_FUNCTION (this << address);

  CheckCacheStateAndFlush ();

  // typename IpRouteMap_t::iterator iter = m_ipRouteCache.find (address);
  // if (iter != m_ipRouteCache.end ())
  //   {
  //     NS_LOG_LOGIC ("Found IpRoute in cache.");
  //     return iter->second;
  //   }

  // not in cache
  return 0;
}
```

Task 2.1 + Task 3.1

從 void 改成會回傳 Ptr<PacketSink>，方便計算 throughputs

```
25  Ptr<PacketSink> SendPacket (int srcId, int dstId);
```

先透過 NodeList 依 ID 找到對應的 Node。

在目的端安裝 PacketSink，PacketSinkHelper 在 dst 上開一個 TCP socket，監聽 port，並立刻啟動（Start(0.0)）。

之後把 ApplicationContainer 轉成 Ptr<PacketSink>，方便 Task 3 用 GetTotalRx() 查統計。

抓取目的端 IP address

在來源端安裝 BulkSendHelper

回傳 Ptr<PacketSink>

```
70  Ptr<PacketSink> SendPacket(int srcId, int dstId) {
71      // Task 2.1: Complete this function
72      // Task 2.1: Set MaxBytes to 512 & Task 3.1: Set MaxBytes to 0
73      // 找出 src, dst
74      Ptr<Node> src = NodeList::GetNode (srcId);
75      Ptr<Node> dst = NodeList::GetNode (dstId);
76
77      // 在 dst 安裝 PacketSink
78      PacketSinkHelper sinkHelper (
79          "ns3::TcpSocketFactory",
80          InetSocketAddress (Ipv4Address::GetAny (), port));
81      ApplicationContainer sinkApps = sinkHelper.Install (dst);
82      sinkApps.Start (Seconds (0.0));
83      // 取出 Ptr<PacketSink>
84      Ptr<PacketSink> sink = DynamicCast<PacketSink> (sinkApps.Get (0));
85
86      // 取 dst IP
87      Ptr<Ipv4> ipv4 = dst->GetObject<Ipv4> ();
88      Ipv4Address dstAddr = ipv4->GetAddress (1, 0).GetLocal ();
89
90      // 在 src 安裝 BulkSend，MaxBytes 依 Task 而定
91      BulkSendHelper sendHelper (
92          "ns3::TcpSocketFactory",
93          InetSocketAddress (dstAddr, port));
94      sendHelper.SetAttribute ("MaxBytes",
95          UintegerValue (Task == 3 ? 0 : 512)); // Task3 持續發
96      sendHelper.SetAttribute ("SendSize", UintegerValue (512));
97      ApplicationContainer sendApps = sendHelper.Install (src);
98      sendApps.Start (Seconds (0.0));
99
100     return sink;
```

Task 2.2 + Task 3.2

Call SendPacket() in main()

```
    // Task 2.2 & Task 3.2 : Call SendPacket()
    ////////////////////
    // NEW: Task2 只傳 36→38；Task3 傳三對 SD-pair
    if (Task == 2)
    {
    SendPacket (36, 38);
    }
    else
    {
    SendPacket (36, 38);
    SendPacket (37, 40);
    SendPacket (39, 41);
    }
    ////////////////////
```

Task 2.3

因為 TCP 控制封包（如 ACK、SYN、FIN）通常小於 100 bytes，資料段（有 payload）一定較大，這樣保證只列印實際帶資料的封包。

```
static void EchoMacTxRx(std::string context, const Ptr< const Packet > packet) {
    // Task 2.3: Complete this function
    // Hint: you can extract the header of the packet to check whether this packe

    // 1) 先用大小過濾，ACK／控制封包通常 <100 bytes
    uint32_t pktSize = packet->GetSize ();
    if (pktSize <= 100)
        {
        return;
        }
```

從 context 字串解析 Node ID：把它切成多個 / 分隔的 token，一旦看到 NodeList，下一個 token 就是節點 ID。

```
    // 2) 由 context 解析 nodeId
    int nodeId = -1;
    {
        std::istringstream iss (context);
        std::string token;
        while (std::getline (iss, token, '/'))
        {
            if (token == "NodeList" && std::getline (iss, token, '/'))
            {
                nodeId = std::stoi (token);
                break;
            }
        }
    }
```

判斷是 Tx 還是 Rx 並輸出結果

```
    // 3) 決定是 MacTx 還是 MacRx
    std::string ev = (context.find ("MacTx") != std::string::npos
                          ? "MacTx" : "MacRx");

    // 4) 印出
    std::cout << ev
              << " at node: " << nodeId
              << ", now: "  << Simulator::Now ()
              << std::endl;
```

Task 3.3

Calculate throughput

• Use GetTotalRx() to check how many bytes the destination has received when simulation ends

```
    // Task 3.3 : Calculate throughput
    ////////////////////
    if (Task == 3)
        {
        uint64_t rx1 = sink36_38->GetTotalRx ();
        uint64_t rx2 = sink37_40->GetTotalRx ();
        uint64_t rx3 = sink39_41->GetTotalRx ();
        uint64_t total = rx1 + rx2 + rx3;
        std::cout << "36->38: " << rx1 << std::endl;
        std::cout << "37->40: " << rx2 << std::endl;
        std::cout << "39->41: " << rx3 << std::endl;
        std::cout << "Total throughput: " << total << std::endl;
        }
    ////////////////////
```

Questions

• Q1: Explain how parentVector in nix-vector-routing describe a path

1. parentVector 定義 ‣ std::vector<Ptr<Node>> parentVector 的大小等於網路中 node 總數，每個索引 i 都儲存到「節點 i」的前驅 (parent) pointer。
2. BFS 建表 ‣ BFS 遍歷結束後，對於每個可達節點 j，parentVector[j] 指向從 source 走到 j 時的前一站。
3. 路徑重建 ‣ 要從 source 走到 dest，只要從 dest 開始，一次 j = parentVector[j]->GetId() 追溯到 source，即可反向重建整條節點序列。
4. NixVector 轉換 ‣ BuildNixVector() 會反覆地：
   - 對當前 dest 找 parent = parentVector[dest]
   - 在 parent 節點的鄰居列表中找到 dest 的 index，呼叫 nixVector->AddNeighborIndex(...)
   - 再把 dest = parent，重覆直到回到 source ‣ 最後 nixVector 就是一個「壓縮過的 source-routing vector」。

• Q2: Explain why you get different total throughputs for paths1.in and paths2.in? Does congestion occurs in paths1.in and paths2.in?

Path 1 的 total throughputs 比 path 2 少很多，因為他的路徑裡面有用到同樣的 station，尤其是 35 每次都會用到。相反的 path 2 的路徑裡面都沒有會用到第二次的 station，所以 path 1 會有塞車的問題，path 2 沒有

• Q3: Please provide more experiments to clarify your answer in Q2. (Hint: Try to transmit each SD pair separately)

Path 1：

```
36->38: 38400
37->40: 37376
39->41: 10752
Total throughput: 86528
```

這是全部一起傳的

```
36->38: 95232
Total throughput: 95232
```

```
37->40: 95232
Total throughput: 95232
```

這是分開傳送的，一起傳送的時候明顯 through put 小很多，代表 congestion 很嚴重

```
39->41: 95232
Total throughput: 95232
```

Path 2：

```
36->38: 95232
37->40: 95744
39->41: 118272
Total throughput: 309248
```

這是全部一起傳的

```
36->38: 95232
Total throughput: 95232
```

```
37->40: 95744
Total throughput: 95744
```

這是分開傳送的，跟一起傳的結果一樣，代表沒有 congestion

```
39->41: 118272
Total throughput: 118272
```