Implementation for Dynamic Routing Protocol

Steps for implementation

1. **Set up your ns3 :**
- Make sure that ns3 is installed in the computer. If not, install it.

2. **Include necessary libraries :**
- In your script, include the necessary libraries.

3. **Define network topology :**
- For your network topology, create the nodes and links.
4. **Implement the dynamic routing protocol :**
- Create a custom routing protocol class to dynamically update routes based on network conditions.
5. **Integrate the Custom Routing Protocol :**
- On the ns3 stack, integrate your custom routing protocol.
6. **Set Up Applications :**
- To generate traffic and test the routing, install applications.
7. **Run the Simulation :**
- Define the simulation parameters and run it.

Example implementation in C++

Here is a complete example on implementing a basic dynamic routing protocol in ns3:

Include libraries :

```
#include “ns3/core-module.h”
#include “ns3/network-module.h”
#include “ns3/internet-module.h”
#include “ns3/point-to-point-module.h”
#include “ns3/applications-module.h”
#include “ns3/ipv4-routing-protocol.h”
#include “ns3/ipv4-list-routing-helper.h”
```

Define Network Topology:

```
using namespace ns3;
int main(int argc, char *argv[]) {
CommandLine cmd;
cmd.Parse(argc, argv);
NodeContainer nodes;
nodes.Create(4);
```

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute("DataRate",
StringValue("5Mbps"));
pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
NetDeviceContainer devices;
devices.Add(pointToPoint.Install(nodes.Get(0), nodes.Get(1)));
devices.Add(pointToPoint.Install(nodes.Get(1), nodes.Get(2)));
devices.Add(pointToPoint.Install(nodes.Get(2), nodes.Get(3)));
devices.Add(pointToPoint.Install(nodes.Get(3), nodes.Get(0)));
InternetStackHelper stack;
stack.Install(nodes);
Ipv4AddressHelper address;
address.SetBase("10.1.1.0", "255.255.255.0");
address.Assign(devices);
```

Implement the Dynamic Routing Protocol:

- Create a new header file dynamic-routing.h:

```
#ifndef DYNAMIC_ROUTING_H
#define DYNAMIC_ROUTING_H
#include "ns3/ipv4-routing-protocol.h"
#include "ns3/ipv4.h"
#include "ns3/net-device.h"
#include "ns3/ptr.h"
#include "ns3/socket.h"
namespace ns3 {
class DynamicRouting : public Ipv4RoutingProtocol {
public:
static TypeId GetTypeId(void);
DynamicRouting();
virtual ~DynamicRouting();
virtual Ptr<Ipv4Route> RouteOutput(
Ptr<Packet> p, const Ipv4Header &header,
Ptr<NetDevice> oif, Socket::SocketErrno &sockerr);
virtual bool RouteInput(
Ptr<const Packet> p, const Ipv4Header &header,
Ptr<const NetDevice> idev,
UnicastForwardCallback ucb, MulticastForwardCallback mcb,
LocalDeliverCallback lcb, ErrorCallback ecb);
virtual void NotifyInterfaceUp(uint32_t interface);
virtual void NotifyInterfaceDown(uint32_t interface);
virtual void NotifyAddAddress(uint32_t interface,
Ipv4InterfaceAddress address);
```

```
virtual void NotifyRemoveAddress(uint32_t interface,
Ipv4InterfaceAddress address);
virtual void SetIpv4(Ptr<Ipv4> ipv4);
private:
void MonitorNetworkConditions();
void UpdateRoutingTable();
Ptr<Ipv4> m_ipv4;
std::map<Ipv4Address, Ptr<Socket>> m_socketMap;
// Other data structures for dynamic routing state
};
}
#endif // DYNAMIC_ROUTING_H
```

Create the corresponding implementation file dynamic-routing.cc:

```
#include “dynamic-routing.h”
#include “ns3/log.h”
#include “ns3/ipv4-routing-table-entry.h”
#include “ns3/simulator.h”
#include <algorithm>
namespace ns3 {
NS_LOG_COMPONENT_DEFINE(“DynamicRouting”);
NS_OBJECT_ENSURE_REGISTERED(DynamicRouting);
TypeId DynamicRouting::GetTypeId(void) {
static TypeId tid = TypeId(“ns3::DynamicRouting”)
.SetParent<Ipv4RoutingProtocol>()
.SetGroupName(“Internet”)
.AddConstructor<DynamicRouting>();
return tid;
}
DynamicRouting::DynamicRouting() {
NS_LOG_FUNCTION(this);
Simulator::Schedule(Seconds(1.0),
&DynamicRouting::MonitorNetworkConditions, this);
}
DynamicRouting::~DynamicRouting() {
NS_LOG_FUNCTION(this);
}
void DynamicRouting::SetIpv4(Ptr<Ipv4> ipv4) {
NS_LOG_FUNCTION(this << ipv4);
m_ipv4 = ipv4;
UpdateRoutingTable();
}
```

```
void DynamicRouting::MonitorNetworkConditions() {
NS_LOG_FUNCTION(this);
// Implement network monitoring logic
// This can include checking link utilization, delay, packet
loss, etc.
UpdateRoutingTable();
Simulator::Schedule(Seconds(1.0),
&DynamicRouting::MonitorNetworkConditions, this);
}
void DynamicRouting::UpdateRoutingTable() {
NS_LOG_FUNCTION(this);
// Implement routing table update logic based on network
conditions
// Example: Using link state information to update routes
// Clear existing routes
Ptr<Ipv4RoutingProtocol> staticRouting = m_ipv4-
>GetRoutingProtocol();
staticRouting->NotifyInterfaceDown(0);
// Get the list of all nodes in the network
NodeContainer allNodes = NodeContainer::GetGlobal();
// Calculate the shortest path to each node
for (NodeContainer::Iterator it = allNodes.Begin(); it !=
allNodes.End(); ++it) {
Ptr<Node> node = *it;
// Skip the current node
if (node == m_ipv4->GetObject<Node>()) continue;
// Calculate the shortest path to this node
Ptr<Ipv4Route> route = Create<Ipv4Route>();
route->SetDestination(node->GetObject<Ipv4>()->GetAddress(1,
0).GetLocal());
route->SetGateway(Ipv4Address("0.0.0.0")); // Assuming direct
connection for simplicity
route->SetOutputDevice(m_ipv4->GetNetDevice(1));
// Add the route
staticRouting->AddHostRouteTo(route->GetDestination(), route-
>GetGateway(), route->GetOutputDevice());
}
}
Ptr<Ipv4Route> DynamicRouting::RouteOutput(
Ptr<Packet> p, const Ipv4Header &header, Ptr<NetDevice> oif,
Socket::SocketErrno &sockerr) {
```

```
NS_LOG_FUNCTION(this << p << header << oif << sockerr);
// Implement route output logic
// For simplicity, use a static route
Ptr<Ipv4Route> route = Create<Ipv4Route>();
route->SetDestination(header.GetDestination());
route->SetGateway(Ipv4Address("10.1.1.2"));
route->SetOutputDevice(m_ipv4->GetNetDevice(1));
return route;
}
bool DynamicRouting::RouteInput(
Ptr<const Packet> p, const Ipv4Header &header,
Ptr<const NetDevice> idev, UnicastForwardCallback ucb,
MulticastForwardCallback mcb, LocalDeliverCallback lcb,
ErrorCallback ecb) {
NS_LOG_FUNCTION(this << p << header << idev << ucb << mcb
<< lcb << ecb);
// Implement route input logic
// Check if the packet is destined for this node
if (header.GetDestination() == m_ipv4->GetAddress(1,
0).GetLocal()) {
lcb(p, header, idev);
return true;
}
// Otherwise, forward the packet using the dynamic routing table
Ptr<Ipv4Route> route = Create<Ipv4Route>();
route->SetDestination(header.GetDestination());
route->SetGateway(Ipv4Address("10.1.1.2")); // Example gateway
route->SetOutputDevice(m_ipv4->GetNetDevice(1));
ucb(route, p, header);
return true;
}
void DynamicRouting::NotifyInterfaceUp(uint32_t interface) {
NS_LOG_FUNCTION(this << interface);
}
void DynamicRouting::NotifyInterfaceDown(uint32_t interface) {
NS_LOG_FUNCTION(this << interface);
}
void DynamicRouting::NotifyAddAddress(uint32_t interface,
Ipv4InterfaceAddress address) {
NS_LOG_FUNCTION(this << interface << address);
}
```

```
void DynamicRouting::NotifyRemoveAddress(uint32_t interface,
Ipv4InterfaceAddress address) {
NS_LOG_FUNCTION(this << interface << address);
}
}
```

Integrate the Custom Routing Protocol:

```
#include “dynamic-routing.h”
int main(int argc, char *argv[]) {
CommandLine cmd;
cmd.Parse(argc, argv);
NodeContainer nodes;
nodes.Create(4);
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute(“DataRate”,
StringValue(“5Mbps”));
pointToPoint.SetChannelAttribute(“Delay”, StringValue(“2ms”));
NetDeviceContainer devices;
devices.Add(pointToPoint.Install(nodes.Get(0), nodes.Get(1)));
devices.Add(pointToPoint.Install(nodes.Get(1), nodes.Get(2)));
devices.Add(pointToPoint.Install(nodes.Get(2), nodes.Get(3)));
devices.Add(pointToPoint.Install(nodes.Get(3), nodes.Get(0)));
InternetStackHelper stack;
DynamicRoutingHelper dynamicRouting;
Ipv4ListRoutingHelper list;
list.Add(dynamicRouting, 0);
stack.SetRoutingHelper(list);
stack.Install(nodes);
Ipv4AddressHelper address;
address.SetBase(“10.1.1.0”, “255.255.255.0”);
address.Assign(devices);
// Set up applications
uint16_t port = 9;
UdpEchoServerHelper server(port);
ApplicationContainer apps = server.Install(nodes.Get(3));
apps.Start(Seconds(1.0));
apps.Stop(Seconds(10.0));
UdpEchoClientHelper client(address.GetAddress(3), port);
client.SetAttribute(“MaxPackets”, UintegerValue(1));
client.SetAttribute(“Interval”, TimeValue(Seconds(1.0)));
client.SetAttribute(“PacketSize”, UintegerValue(1024));
apps = client.Install(nodes.Get(0));
```

```
apps.Start(Seconds(2.0));
apps.Stop(Seconds(10.0));
Simulator::Run();
Simulator::Destroy();
return 0;
}
```

Explanation

1. **Network Topology :**

The four nodes connected in a ring, to set up the network topology.

2. **Dynamic routing protocol :**

A custom dynamic routing protocol is implemented for monitoring network conditions and updates the routing table accordingly.

3. **Monitor network Condition :**

The MonitorNetworkConditions method is called periodically to check network conditions and update the routing table.

4. **Update routing table :**

The updateRoutingTable method recalculate routes in the network.

5. **Integrate Custom Routing Protocol :**

The custom routing protocol is integrated into the ns3 stack using the dynamic RoutingHelper.

6. **Applications :**

To set up to test the routing, UdpEchoServer and UdpEchoClient applications are used.

Running the Code :

Save your script to a file, for example, dynamic-routing.cc and Compile the script using the ns3 build system

```
./waf configure –enable-examples
./waf build
./waf –run scratch/dynamic-routing
```

Overall, we had successfully learned on implementing Dynamic routing in ns3 which involves creating a custom routing protocol that adapts to network changes.