Adaptive In-Network Queue Management with Custom AQM Model in ns-3 simulations

Project Description: This repository hosts a simulation project developed with ns-3.35. The project incorporates a custom Active Queue Management (AQM) model, referred to as dAQM, using higher order derivatives of sojourn time and buffer size. To integrate this model, the associated files must be placed in the `ns-allinone-3.35/ns-3.35/src/traffic-control/model` directory.

Getting Started

Prerequisites

- It is required to install ns-3.35 that uses the Waf-based build system for ns-3.
- While other releases may be used, please note that using other versions may necessitate modifications to the code to ensure compatibility with the build system and compilers.
- Installation of Python 3.8.10 is required.
- A source code editor is required, for example gedit.
- A supported Operating System is required, for example Linux.

Initial Setup

1. For specific installation guidelines related to ns-3.35, please consult the official ns-3 release documentation available at ns-3.35 Official Site (https://www.nsnam.org/releases/ns-3-35/) or its git repository.
- For the detailed overview of installation procedures across different systems, please consult ns-3 Installation Wiki (https://www.nsnam.org/wiki/Installation) or navigate to the main website nsnam (https://www.nsnam.org).

Installation Procedure for source archive release

1. Download the source archive release from the above mentioned ns-3.35 Official Site (https://www.nsnam.org/releases/ns-3-35/).
2. Unzip the downloaded file with: `$ tar xjf ns-allinone-3.35.tar.bz2`
3. Enter the ns-allinone-3.35 directory by: `$ cd ns-allinone-3.35`
4. Upon listing the directory contents with `ls`, you should see sub-directories and files including but not limited to `bake`, `constants.py`, `ns-3.35`, and `README`.

Configuration and Building

1. Steps for ns-3 for source archive release version.
2. Inside the `ns-allinone-3.35` directory, initiate the build process: `$ ./build.py --enable-examples --enable-tests`
3. Navigate to the `ns-3.35` directory and choose one of the following build options:

- For optimized performance:
  - `$ ./waf clean`
  - `$ ./waf configure --build-profile=optimized --enable-examples --enable-tests`
- For development and debugging:
  - `$ ./waf clean`
  - `$ ./waf configure --build-profile=debug --enable-examples --enable-tests`

Integrating Custom AQM

1. Place the custom AQM files `daqm-queue-disc.h` and `daqm-queue-disc.cc` into `ns-allinone-3.35/ns-3.35/src/traffic-control/model`.
2. Update the `wscript` in the `src/traffic-control` directory to include the these files.

```
# ... existing files
module.source = [
          # ... existing files
          'model/daqm-queue-disc.cc',
          'helper/traffic-control-helper.cc',
          'helper/queue-disc-container.cc'
           ]
        # ... existing files
headers.source = [
          # ... existing files
          'model/daqm-queue-disc.h',
          'helper/traffic-control-helper.h',
          'helper/queue-disc-container.h'
           ]
```

Usage

1. Create a `.txt` configuration file containing network settings and dAQM parameters for each flow type. Please refer to the example configuration files in the repository: dAQM_Artifacts/Source_Code/Different_Configurations. The current configuration file in use is `Configuration_dAQM1.txt`.

- Modify simulation parameters like the number of clients and servers, link bandwidth, client transmission rate, packet size, etc., in this file.
- The 'client lambda' parameter is tailored for the Poisson Process Model. While it's unique to this model, please keep it in the configuration file, as it will be automatically bypassed when other models are in use.
- The traffic type at the configuration file's top defines the naming convention for files produced by the simulation. Example: `_[flowType]_[numberOfClients]_[trafficModel]_[ShortFlow/LongFlow]_dAQM_[configurationNumber]`.
- Example configuration structure:

```
traffic_type = _ftp_100_weibull_LF_dAQM_11
nclient = 100
nserver = 5
csma_data_rate = 2Gbps
csma_delay = 1
client_lambda = 892.0
client_data_rate = 10Mbps
packet_size = 1400
sojourn1_drop_threshold = 10
sojourn1_drop_duration = 400
sojourn1_drop_rate = 0.05
sojourn1_packet_interval = 1
sojourn2_drop_threshold = 10
sojourn2_drop_duration = 400
sojourn2_drop_rate = 0.05
sojourn2_packet_interval = 1
sojourn3_drop_threshold = 10
sojourn3_drop_duration = 400
sojourn3_drop_rate = 0.05
sojourn3_packet_interval = 1
sojourn4_drop_threshold = 1000
sojourn4_drop_duration = 400
sojourn4_drop_rate = 0.05
sojourn4_packet_interval = 1
buffer1_drop_threshold = 1
buffer1_drop_duration = 400
buffer1_drop_rate = 0.05
buffer2_drop_threshold = 10
buffer2_drop_duration = 400
buffer2_drop_rate = 0.05
buffer3_drop_threshold = 10
buffer3_drop_duration = 400
buffer3_drop_rate = 0.05
buffer4_drop_threshold = 120
buffer4_drop_duration = 400
buffer4_drop_rate = 0.05
```

- For modifications to other AQMs (like RED, CoDel, etc.) outside of dAQM, changes to the traffic models, or switching between the TCP and UDP transport layer protocols, please make adjustments in the `Call_function.cc` file.

2. Replace the configuration file(`Configuration_dAQM1.txt`) being loaded in the `Call_function.cc`:

```
if (!loadParametersFromFile("Configuration_dAQM1.txt", configSets)) {
        std::cerr << "Failed to load parameters from file.\n";
        return 1;
}
```

3. Update Transportation Protocols:

- Modify protocols in the Server and Client loops. Changing to UDP connections, comment the TcpSocketFactory usage and activate the UdpSocketFactory usage.

```
//PacketSinkHelper sink("ns3::TcpSocketFactory", serverAddress);
PacketSinkHelper sink("ns3::UdpSocketFactory", serverAddress);

//OnOffHelper weibullclient ("ns3::TcpSocketFactory", InetSocketAddress
(csmaInterfaces.GetAddress (serverIndex), port));
OnOffHelper weibullclient ("ns3::UdpSocketFactory", InetSocketAddress
(csmaInterfaces.GetAddress (serverIndex), port));
```

4. Update Traffic Models in use from Weibull to Poisson:

```
double lambda = config.client_lambda; //Arrival rate of the Poisson process
double mean = 1.0 / lambda; //Mean inter-arrival time that follows a Poisson process

//Set up Clients
//This block includes a combination of various traffic models. Uncomment the appropriate model as needed.
for (uint32_t i = 0; i < nClients; ++i) {

    //other models

    //Poisson Traffic Configuration
    OnOffHelper poissonclient ("ns3::TcpSocketFactory", InetSocketAddress (csmaInterfaces.GetAddress (serverIndex), port));
    ns3::Ptr < ExponentialRandomVariable > poisson = CreateObject < ExponentialRandomVariable > ();
    poisson -> SetAttribute("Mean", DoubleValue(mean)); //scale parameter
    ...
    poiApps.Start(Seconds(startTime));
    poiApps.Stop(Seconds(simulationTime));

    //other models

    // Weibull Traffic Configuration
    /*OnOffHelper weibullclient ("ns3::TcpSocketFactory", InetSocketAddress (csmaInterfaces.GetAddress (serverIndex), port));
    //OnOffHelper weibullclient ("ns3::UdpSocketFactory", InetSocketAddress (csmaInterfaces.GetAddress (serverIndex), port));
    ns3::Ptr<WeibullRandomVariable> weibull = CreateObject<WeibullRandomVariable> ();
    weibull->SetAttribute ("Scale", DoubleValue (1)); //scale parameter
    weibull->SetAttribute ("Shape", DoubleValue (1.5)); //shape parameter
        ...
    weiclientApps.Start (Seconds (0.0));
    weiclientApps.Stop (Seconds (simulationTime));*/

    ...
}
```

5. Specify the AQMs to be used:

- Please note that the dAQM1 to dAQM8 in the code refers to the 8 features variation. The configuration file `Configuration_dAQM1.txt` is refering to the dAQM9 in code,and `Configuration_dAQM2.txt`, `Configuration_dAQM3.txt` refering the dAQM10 and dAQM11, respectivly.

```
std::vector<std::string> aqmAlgorithms = {"Fifo", "Red", "CoDel",
"FqCoDel", "Cobalt", "Pie", "dAQM1", "dAQM2", "dAQM3", "dAQM4", "dAQM5",
"dAQM6", "dAQM7", "dAQM8"};
//std::vector<std::string> aqmAlgorithms = { "dAQM9"};
//std::vector<std::string> aqmAlgorithms = { "dAQM10"};
//std::vector<std::string> aqmAlgorithms = { "dAQM11"};
```

Running the Simulation

1. The type of build (`debug` vs `optimized`) that will be used for the run depends on the last configuration in ns-3. To run a simulation file located in the `scratch` folder, under the `ns-3.35` directory:

- Place the configuration file under `ns-3.35`.
- `$ export NS_LOG=networkSimulation=info`
- `$ ./waf --run scratch/Call_function.cc`

Author

- Sunny Shu: Code developer.

Contributor

- Saad Saleh: Originator of the dAQM concept.

License

- This simulation project is an academic endeavor and is built upon the free software network simulator ns-3, under the terms of the GNU General Public License version 2. See the LICENSE file for more information.