DDS Performance Tester

What the Application Does

The provided code is for a C++ application named **DdsPerfTest**, designed to measure the performance and reliability of a DDS (Data Distribution Service) network, specifically using the **CycloneDDS** implementation.

At its core, the application is a flexible tool for creating a configurable number of DDS **publishers** and **subscribers** across multiple computers on a network. It features a graphical user interface (GUI) built with **Dear ImGui**, allowing a user on one machine to centrally control and monitor an entire multi-machine test scenario in real-time.

Here's a breakdown of its architecture and key features:

- **Multi-App Coordination**: The application can be run on several machines simultaneously. A discovery mechanism (
AppScan) allows each instance to find all other running instances on the network[111111111].

- **Centralized Control**: One instance acts as a controller. Through its UI (
MsgEdit), the user defines the test parameters, such as which message types to use, their publish rate, and their size[222222222]. These settings (

MasterSettings) are then published over DDS to all other app instances[333333333].

- **Dynamic Test Execution**: Each application instance receives the MasterSettings and dynamically adjusts its state. It creates or destroys publisher and subscriber objects to match the specification for its particular machine[444444444]. To simulate multiple applications on a single machine, each new publisher or subscriber can be assigned a different DDS participant[5555].

- **Real-time Statistics**: Subscribers monitor the data they receive, tracking the total message count, the current message rate, and—most importantly—the number of **lost messages** by checking a sequence number in each message[666]. These statistics are published back over DDS[7].

- **Interactive Monitoring**: The SubsStatsMgr component collects the statistics from all subscribers across all machines and displays them in a consolidated table in the UI, providing a live overview of the network's health and performance[8888].

Usage Guide for Performance Testing

This guide explains how to use the DdsPerfTest tool to analyze the performance of your DDS network.

1. UI-Driven Interactive Testing

The application's UI is divided into several windows that allow you to set up, control, and monitor tests.

Step 1: Initial Setup and App Discovery

1. Launch the DdsPerfTest.exe application on all computers you want to include in the test.
2. On one of the machines (which will act as your controller), go to the **"App"** window.
3. Click the
   "Collect apps" button[9]. The application will use DDS to discover all other running instances. After a moment, you will see all applications appear in the UI, enabling you to assign publishers and subscribers to them.

Step 2: Configuring a Test Scenario in the "Msg Setting" Window

This is the main control panel for your test.

1. **Add a Message Class**: Click the + button[10]. A dropdown will appear with all message types defined in

   MsgDefs.csv[11]. Select one to add it to the test scenario.

2. **Configure Parameters**: For each message class you add, a collapsible header will appear. Click it to expand the settings[12]:

   ○ **Disabled**: A checkbox to enable or disable testing for this specific message class. Unchecking it will create the publishers and subscribers across the network[13].

   ○ **Rate**: A slider to control how many messages per second each publisher will send (in Hz)[14].

   ○ **Size**: A slider to set the approximate payload size (in bytes) of each message[15151515].

   ○ **Pubs/Subs**: These rows are for assigning work. For both publishers ("Pubs") and subscribers ("Subs"), you will see a series of input boxes. Each box corresponds to a discovered application instance[16161616]. Enter the

     number of publishers or subscribers you want to create on that specific machine[17].

Step 3: Monitoring and Assessing Performance

The key to assessing performance is the **"Subscriber Stats"** window. This table aggregates data from every subscriber in your test network.

- **What the Columns Mean**:
  - Msg: The message class being subscribed to.
  - App: The index of the application instance running the subscriber.
  - Idx: The index of the subscriber within that app (if there are multiple).
  - Recv: Total number of messages received by this subscriber.
  - Rate: The calculated rate of messages per second this subscriber is receiving.
  - Lost: The total number of messages this subscriber has missed, calculated by checking for gaps in the publisher's sequence number[18].

- **How to Assess Performance**:
  - **Verify Throughput**: The Rate column should closely match the Rate you set for publishers in the "Msg Setting" window. If the Rate is consistently lower, it indicates a bottleneck, which could be due to network saturation, high CPU load on the publisher or subscriber machine, or inefficient QoS settings.
  - **Identify Reliability Issues**: The **Lost column is the most critical indicator of network problems**. For a reliable QoS configuration, this number should ideally be **0**.

        - If Lost is greater than zero, your network is dropping packets, or DDS is unable to keep up.
        - **Experiment**: Try increasing the message Rate and Size. Observe at what point the Lost count starts to climb. This helps you find the performance limits of your network hardware and DDS configuration.
        - **Compare QoS**: Run a test with a reliable message type and note the Lost count. Then, run the same test with a bestEffort message type. You will likely see a much higher Lost count, demonstrating the trade-off between reliability and performance.

The

"Local Settings" window provides a read-only summary of the settings currently active on the instance you are viewing, which is useful for verification[19].

---

2. Using Pre-defined Message Types

The application's behavior is heavily influenced by the message definitions in MsgDefs.csv. Each definition combines different DDS QoS (Quality of Service) settings to simulate common use cases.

- stat_UVL1_LI (**BestEffort, Volatile, KeepLast(1), ListenImmed**):
    - **Use Case**: Simulates high-frequency, non-critical data like streaming sensor readings or video game character positions.
    - **Performance Assessment**: Use this to find the raw throughput of your network. Since it's bestEffort, expect the Lost count to rise under heavy load. The goal is to see how high you can push the Rate before the Lost count becomes unacceptable for your application.
      ListenImmed means the subscriber processes data directly in the DDS callback thread, which is fast but can cause issues if processing is slow[20202020].

- cmd_RTA_P (**Reliable, TransientLocal, KeepAll, Poll**):
    - **Use Case**: Simulates critical commands that must not be lost and must be available to applications that start later.
    - **Performance Assessment**: This is a test of reliability. The
      Lost count should remain at 0. TransientLocal durability and KeepAll history mean that a newly started subscriber will be flooded with all previously sent messages, which can be a stress test in itself[21]. The

Poll strategy means the main application thread is responsible for reading data, which can be safer but potentially slower than listening[22222222].

- stat_RVL1_LD (**Reliable, Volatile, KeepLast(1), ListenDefer**):
  - **Use Case**: Simulates reliable state updates where only the most recent value is important (e.g., the current status of a device).
  - **Performance Assessment**: Tests reliable delivery without the history overhead. ListenDefer is a hybrid strategy where the DDS thread just flags that data is available, and the main thread reads it later[23232323]. This can be a good balance between responsiveness and thread safety.

---

3. Defining New Message Types

You can easily define new message types to test different QoS configurations without changing any C++ code.

1. **Open MsgDefs.csv** in a text editor.
2. **Add a new line** to the end of the file.
3. **Fill in the columns** with the desired QoS parameters:
   - **Name**: A unique name for your test case (e.g., MyTest_HighReliability).
   - **Reliability**: Reliable or BestEffort.
   - **Durability**: Volatile (messages are not persisted) or TransientLocal (messages are persisted for late-joining readers).
   - **History**: KeepAll (keep all messages in history) or KeepLast (only keep a certain number).
   - **HistoryDepth**: The number of messages to keep if History is KeepLast.
   - **ReadStrategy**: Poll, ListenImmed, or ListenDefer.
4. **Save the file** and restart the DdsPerfTest application on your controller machine.

Your new message type will now be available for selection when you click the

+ button in the **"Msg Setting"** window[24], allowing you to immediately use it in your tests.