Protopeer Documentation v0.1

2018-09-04

- List of all changes since 2016
- branch: `pilot.2017.f`
- [ ] `To be placed in DIAS Documentation`

Intro

The *Protopeer Framework* was first designed in X by X with the intention of providing a robust platform for carrying out peer-to-peer experiemnts. One important characteristic of the *Protopeer Framework* is the ability to carry out the experiments in either simulated mode or real-time mode, with minimal (if any) change to the code. At the time of the initial release, the *Protopeer Framework* was the only platform with that capability. In simulated mode, all the peers reside in a single process running inside a single Java Virtual Machine instance on a single machine. In real-time mode, each peer is a separate process running inside it's own Java Virtual Machine instance and can be located anywhere on the Internet.

In 2016 we started working on the real-time implementation of DIAS. Back then, the *Protopeer Framework* still had many low-level technical issues with it's real-time mode such as memory leaks, static port allocation, etc All of these issues needed to be solved to allow stable real-time DIAS experiments.

This section is organised into the following sub-sections which review the improvements that were made to the *Protopeer Framework*:

1. Memory Leaks
2. Networking Optimisations
3. DIAS Logging Framework

Memory Leaks

1. Replaced Apache Mina with ZeroMQ
    - the original Apache Mina library had memory leaks
    - implemented JeroMQ (native zeroMQ port for java)
        - tested with jeromq versions:
            - 3.0
            - 4.0

2. MeasurementLog
    - round robin was not implemented correctly -> log kept growing in memory

3. ByteBuffer.allocate() calls repeatedly made throughout the codebase
    - ByteBuffer.allocate() allocates an array of bytes that bypasses Java GC, leading to memory leaks

- code modified so that the allocate() call is only made once per class instance

Networking Optimisations

- 1. off-thread message serialisation
    - when a thread calls sendMessage(), message is serialised immediataly by the calling thread
    - previously, the serialisation was performed off-thread just before sending the message
    - this created syhcnronisation issues, for example with DIAS::Disseminator

- 2. dynamic port allocation
    - previously the port was determined by the startup script and passed as an argument to the executable
    - this can create problems when scaling to hundreds of peers on a host where other processes are using ports as well
    - peers will not launch and fail if the assigned port is already used
    - if port is not set, then port is dynamically allocated using ZeroMQ::bind function, which searches in the dynamic port range for the next available port 49152 -> 65535
- 3. connection pool
    - a connection pool of 20 connections is maintained opened with peers
    - if the peer runs out of connections, an older connection is closed and a new connection is created

DIAS Logging Framework

1. uses the DIAS-Logging-System library functionality
    - RawLog, a global static that can be accessed from anywhere within the code
    - e.g RawLog.print(2, "my message here")