Package for Controlling Delivery of Messages

Robert Pate and Samuel Cherinet

December 2017

Introduction

Transit is package for controlling distribution of messages between java programs. The user can choose to use it asynchronously or synchronously as Transit asynchronously sends and receives messages into a queue.

Core features

- Abstracts away low level TCP handling
- Allows developers to add custom logic before delivering received message to their application like
  - Change the order of messages
  - Or deciding to accept or reject messages
- Works with any serializable object as a message

Restrictions at this time

- Only supports JAVA
- Only uses TCP on the back-end
- Intended for distributed architecture
- No stream support, only message passing

DEMO 1: Request Reply

1. Run hello.Server
2. Run hello.Client
3. Discuss Code and Kill All
4. Run inventory.Server 3003 inventory.txt aka homework 1
5. Run inventory.Client localhost 3003
    a. list
    b. purchase sam ps4 2
6. Run 3 other clients
    a. search sam
7. Discuss Code and Kill All

DEMO 2: Push Socket and Out of Order Messages

1. Run hello.CausalOrderCounterTest 3 times, entering 3, 2, 1 as server IDs
2. Hit Enter for 1 and 2 and show 3's messages are out of order
3. Discuss Sending One Way and Bind early then receive anytime

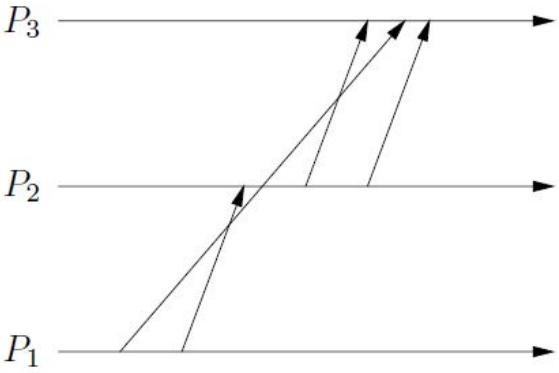


Figure 12.1: A FIFO computation that is not causally ordered

DEMO 3: Causally Ordered Messages

1. Run hello.CausalOrderTest 3 times, entering 3, 2, 1 as server IDs
2. Hit Enter for 1 and 2 and show 3's messages are IN order
3. Discuss Bind first because sender is important to receiver
4. Connect/Send and Receive anytime or never

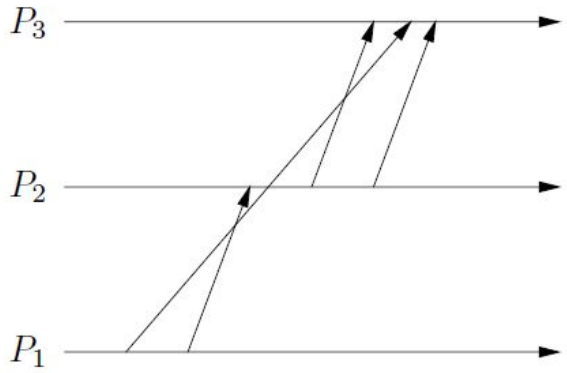
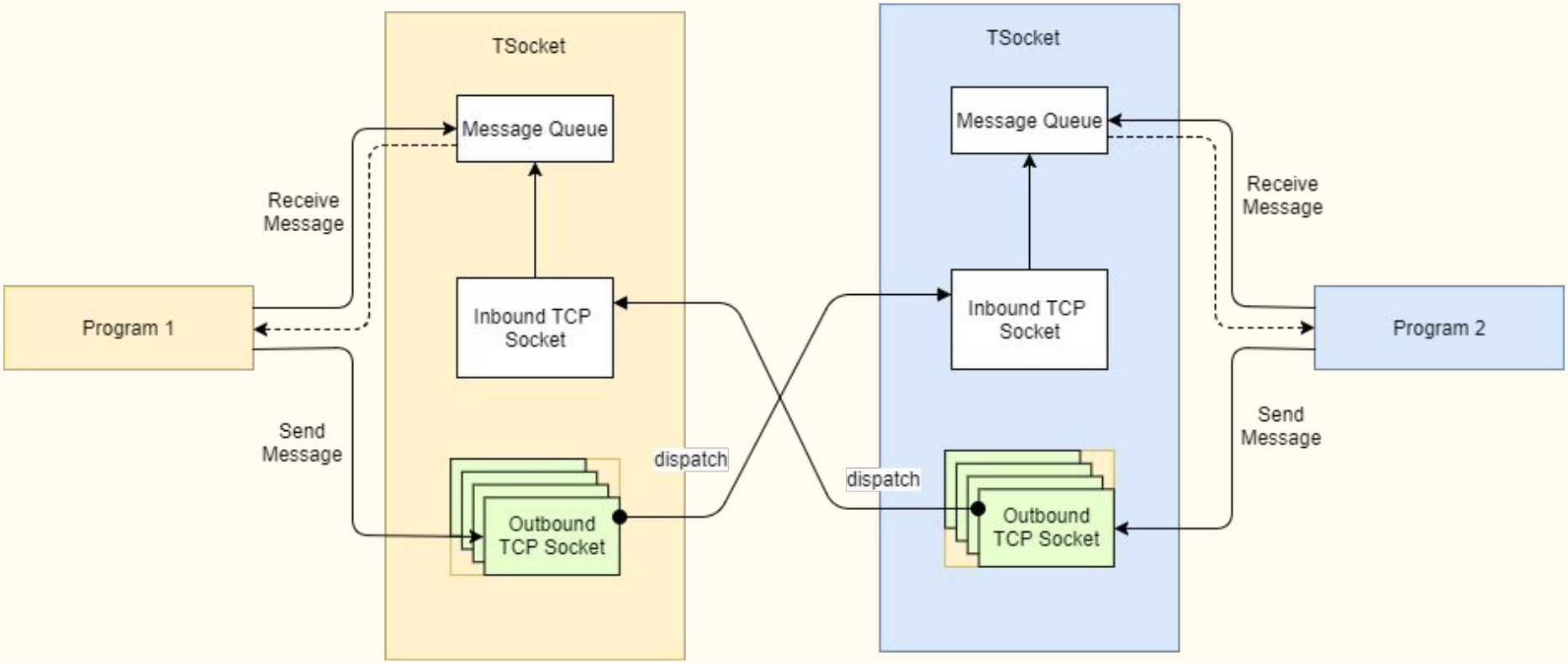


Figure 12.1: A FIFO computation that is not causally ordered

Message communication model



The BaseSocket

- Uses queue to buffer receives messages
- Binds to the configured port and listens to message on a different thread
- On send it opens a socket to the destination process
- Uses a packet that holds the message and source process information
- On receiving a new message it pushes it to the buffer and allows the consuming code to do anything with the messages

BaseSocket Implementation

```
class BaseSocket {
        protected TAddress _bindEndPointAddress;
        protected TAddress _connectEndPointAddress;
        protected Thread _serverRunnerThread;
        protected ConcurrentLinkedQueue<TPacket> messageQueue;
        ...
        protected void bind(String host, int port)
        protected void sendOneWay(Serializable message, TAddress address)
        protected void send(Serializable message, TAddress address)
        protected TPacket receivePacket()
        protected boolean peek()
...
}
```

Request/Reply

- Commonly seen pattern, e.g. homework 1
- Clients use the Requestor class and connect to a server
- Servers use the Replier class and bind to an IP and port
- Clients send a request to the server
- Servers call receive to get the next request in the queue and call reply
- Clients use receive to get the reply
- Both requests and replies are queued up so both sides can do work between receives without losing messages.

PushSocket

- No differentiation between senders and receivers
- FIFO if all in the same thread as send is synchronous
- Bind if you want to receive messages
- Send message to any IP/Port
- If you bound you can call receive to pop a message out of the queue
- Sender addresses are not attached automatically to messages
- Good for tasks such as homework 2’s recovery messages

Causal Order

- BaseSocket is extended to create a CausalSocket, and this class holds
  - Network topology
  - Message matrix
- Since the base implementation provides a queue with pending message only a delivery queue is added to apply the algorithm

Extending BaseSocket

```
public class CausalSocket extends BaseSocket {
        int M[][];
        int N;
        LinkedList deliveryQ = new LinkedList();
        ArrayList<CausalParticipant> participants;
        ...

        void connect(String host,int port)
        boolean okayToRecv(int w[][], int srcId)
        void checkPendingQ()

        ...
}
```

Proposed features

- Closing Receive Sockets and Request Socket
- Timeouts
- Message queue for outgoing messages
- Acks for outgoing messages
- Broadcast Socket
- Multicast Socket
- AMPQ