Bridging Communities with pMOOSBridge

Paul Newman

June 21, 2009

Abstract

This document will give you a description of how to use pMOOSBridge to link multiple MOOS communities together.

Contents

1 Introduction

pMOOSBridge is a powerful tool in building MOOS-derived systems. It allows messages to pass between communities and is able to rename the messages as they are shuffled between communities. Many of the sections in this document rely on pMOOSBridge to set up different communications topologies. There is no correct topology — choose one that works for your own needs. One instance of pMOOSBridge can "talk" to a limitless number of communities.

2 Basic Configuration

The configuration block specifies what should be mapped or "shared" between communities and how it should be done. The SHARE command specifies precisely what variables should be shared between which communities and the

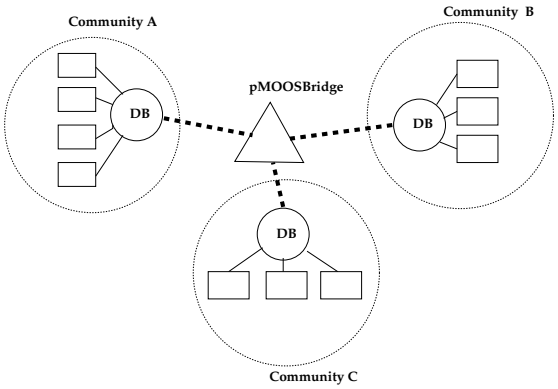

Figure 1: A possible MOOSBridge configuration. One instance of `pMOOSBridge` can "talk" to a limitless number of communities. The configuration block specifies what should be mapped or "shared" between communities and how it should be done. The `SHARE` command specifies precisely what variables should be shared between which communities.

syntax is intuitive:

`SHARE=` *Comm@Host:Port[V1[,V2...]] − > Comm@Host:port [V1,V2...]*

The triplet Comm@Host:Port is a description of a community — name and hostname/port pair. The community description can be omitted on the LHS of the arrow, in which case the mission-file-scope defaults are assumed (see the example below). Each variable ("V") on the LHS (source) community will be inserted into the community on the RHS If no variable names are specified on the RHS (destination) community the original names are used, otherwise there is a one-to-one mapping between variable names on the LHS and new variable names (aliases) on the RHS. If however there are more named shared variables than aliases, the variables for which an alias is not specified retain their original names. For example

```
SHARE= VehA@nym.robots.ox.ac.uk:9000 [GPS_X]->VehB@kayak.mit.edu:9000 [GPS_X]
```

Here the variable `GPS_X` is shared between a community called "VehA", running from a `MOOSDB` on the machine called "nym.robots.ox.ac.uk" listening on port 9000, is being inserted into a community called "VehB" using a `MOOSDB` running on the machine called "VehB@kayak.mit.edu" also listening on port 9000. In both communities the variable is called "GPS_X". However when viewed with something like `uMS` (see the document on Graphical Tools) it can be seen that the `m_sOrginatingCommunity` member of the `MOOSMsg` carrying this data in the "VehB" commmunity will be "VehA".

```
SHARE= VehA@nym.robots.ox.ac.uk:9000 [GPS_X]->VehB@kayak.mit.edu:9000 [GPS_X_A]
```

This is similar to the above example, only `pMOOSBridge` will rename "GPS_X" to "GPS_X_A" in the destination community. The next example shows how the source address need not be specified. When omitted the source community is taken to be the community on which `pMOOSBridge` is running at the time.

This example also shows how destination variable names may be omitted, in which case the original (source community) variable name is preserved.

```
SHARE= [GPS_X]->VehB@kayak.mit.edu:9000
```

Finally, more than one mapping can be specified in one line:

```
SHARE= [GPS_X,OVEN_TEMP]->VehB@kayak.mit.edu:9000 [GPS_X_A]
```

Here GPS_X is being mapped and renamed to `GPS_X_A` in community "VehB" but the variable `OVEN_TEMP` is simply being shared without renaming. It is important to realise that sharing is not bidirectional. In this case, a process notifying change in `GPS_X_A` in community "VehB" *would not* result in pMOOSBridge notifying the `MOOSDB` in community "VehA" that "GPS_X" has changed.

2.1 Specifying Sharing Bandwidth

Recall that when using the `CMOOSCommClient` class or `CMOOSApp::Register()` it is possible to specify the maximum rate at which one wishes to be notified of variable changes. So for example if a variable is actually been changed at 50Hz it is possible to request a notification at no more that 1Hz. Alternatively one can register to receive every notification. The same paradigm exists in code `pMOOSBridge` although it cannot be configured on a per variable basis. By setting `BridgeFrequency` to be some integer value the maximum frequency of notifications being bridge between communities can be controlled. For example

$$\texttt{BridgeFrequency} = 10$$

will mean that no variables are bridge with frequency greater than 10Hz. A special case is

$$\texttt{BridgeFrequency} = 0$$

which means that **every** change in bridged variables is sent to destination communities.

3 Topologies

The configuration of `pMOOSBridge` facilitates many connection topologies. One such topology is show in Figure , here we have one central `pMOOSBridge` joining several communities together. A common alternative is shown in in which each community has its own bridge. Although functionally the two setup are identical the latter offers the opportunity to use UDP sharing as discussed in Section 4.

4 Sharing by UDP

It is possible that an application domain requires that the link between distinct MOOS communities by made not with tcp/ip but with udp. An obvious example would be when intermittent wireless communications is available between communities - using tcp/ip here (the protocol used by standard MOOS comms) will result in poor performance as it is connection oriented substantial time will

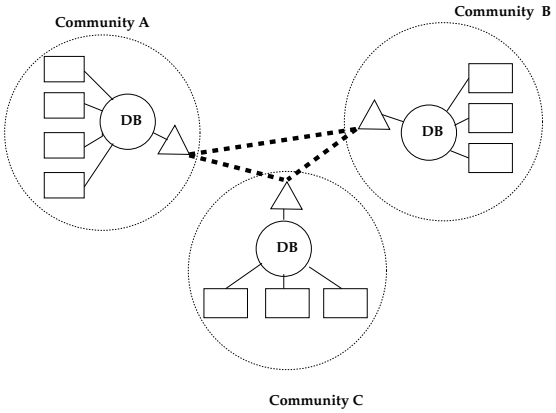

Figure 2: An alternative MOOSBridge Configuration — one bridge per community. This may be preferable; it is undesirable to have one process manage all the sharing of data. However it offers no *functional* advantage over the topology shown in Figure 3. That said, this is the only configuration in which UDP sharing can be used — see section **??**

be expended behind the scenes in reconnecting and blocking read and write calls at various places in the comms architecture [1]. UDP however offers more of a "fire and forget" paradigm - if packets can make it across the inter-community link then they will but the sender is oblivious to their fate.

4.1 Basic Configuration

Before describing how UDP sharing can be configured it is important to note the following points.

- UDP sharing is only possible when using the topology shown in Figure 2 — that is with one `pMOOSBridge` running in each community.
- Any bridge that wants to receive transmissions from other bridges (think "bridge= community' because of the above point — one bridge per community for UDP sharing) must be told the port on which it is to listen for incoming UDP packets. This is done with the `UDPListen` variable in the configuration block. For example

  ```
  UDPListen = 9100
  ```

  will cause the Bridge to listen on port 9100.
- `pMOOSBridge` supports to modes of UDP sharing. The first called "Directed Sharing" sends MOOS mail to just one named recipient (another Bridge in another community) . The second mode, called "Broadcast Sharing" sends data to all and sundry - any bridge who is listening on the relevant port will receive the transmission.
- Both UDP sharing modes are configured with the `UDPShare` keyword.

[1] when a write completes on a tcp/ip socket you are guaranteed that the data has arrived at its destination

4.2 UDP Directed Sharing

In this case the syntax is similar to the regular `SHARE` already discussed although the source community need not be described because the enforced topology constraint means this is implicit. For example, for a Bridge running on community "VehA" a line such as this

```
UDPSHARE= [GPS_X]->VehB@kayak.mit.edu:9200 [GPS_X_VEH_A]
```

in the pMOOSBridge configuration block will cause the `GPS_X` variable within community "VehA" to be sent to a `pMOOSBridge` running on Community "VehB" which has been configured to listening for incoming UDP packets on port 9200 on a machine called kayak.mit.edu. This data will appear as `GPS_X_VEH_A` in the "VehB" community.

4.3 UDP Broadcast Sharing

It is also possible to broadcast messages across the network indiscriminately meaning that every pMOOSBridge listening on the required port will receive the MOOS variable and insert it inot its local community's `MOOSDB` . The required syntax is exactly as for the directed UDP share, the difference comes from specifying the destination address to be "all@broadcast" followed by the required port address. So, carrying on with the example above,

```
UDPSHARE= [GPS_X]->all@broadcast:9200 [GPS_X_VEH_A]
```

will mean any `pMOOSbridge` listening on port 9200 on the local network will receive `GPS_X_VEH_A` which was originally `GPS_X` in community "VehA" (assuming of course that the bridge configured with this line is still runing in a community called "VehA").

4.4 Example Configurations

Listing 1: pMOOSBridge configuration block for a community called V1 illustrating two types of UDP sharing —directed and broadcast

```
// This is an example configuration for testing pMOOSBridgeUDP

ServerHost = localhost
ServerPort = 9000

// you can name tag processes connected to a particular
// DB under a community name
Community = V1

ProcessConfig = pMOOSBridge
{
        //this is the port *this* bridge will be listening for incoming UDP
            packets from other
        //bridges in other communities. If you omit this line then no incoming
            UDP packets
        //will be inserted into the local community
        UDPListen = 9100

        //set up a UDP share of some local variables. Here we are sending
            DB_UPTIME and DB_CLIENTS to
```

```
        //a community called V2 which is receiving UDP on port 9200. So in the
            mission file for that
        //community your would expect to see a bridge being configured with
            UDPListen =9200
        UDPSHARE=  [DB_UPTIME,DB_CLIENTS]->V2@localhost:9200 [V1_UP,V1_DB_CLIENTS
            ]

        //if you want to broadcast a message to everyone then use this special
            address....
        //here every bridge with a "UDPListen=XYZ" statement will receive the
            DB_TIME on community
        //V1 as V1_DB_TIME
        UDPSHARE=  [DB_TIME]->ALL@BROADCAST:9100 [V1_DB_TIME]


}
```

Listing 2: pMOOSBridge configuration block for a community called V2 publishing one variable via directed UDP sharing and one via regular tcp/ip

```
// This is an example configuration for testing pMOOSBridgeUDP

ServerHost = localhost
ServerPort = 9001

// you can name tag processes connected to a particular
// DB under a community name
Community = V2

ProcessConfig = pMOOSBridge
{
        //this is the port *this* bridge will be listening for incoming UDP
            packets from other
        //bridges in other communities. If you omit this line then *no* incoming
            UDP packets
        //will be inserted into the local community
        UDPListen = 9200

        //set up a UDP share of some local variables. Here we are sending DB_TIME
             and to
        //a community called V1 which is receiving UDP on port 9100. So in the
            mission file for that
        //community your would expect to see a bridge being configured with
            UDPListen =9100
        UDPShare = [DB_CLIENTS] -> V1@localhost:9100 [V2_DB_CLIENTS]

        //this is a standard tcp/ip share sending the local DB's time to
            community V1 which
        //is expected to have a DB serving on port 9000
        SHARE= [DB_TIME]->V1@localhost:9000 [V2_DB_TIME]


}
```