Communication in Bond

This technical report provides a detailed explanation of the communication fabric part of Bond. It answers the following two questions: how message is exchanged in Bond? How multicast is achieved in Bond?

1. Communicator and Communication Engine Interface

Each object in Bond has the ability to send and receive messages. It is achieved by using class 'bondCommunicator'. There is a static variable in class 'bondObject' called 'com', which is an instance of Class 'bondCommunicator':

```
public class bondObject
implements Serializable, Cloneable, bondKQMLHandler
{
 static public bondDirectory dir = null;
 static public bondCommunicator com = null;
 …….
 public static void initbond(int port) {
  ……
 }
 ……
}
```

Bond Objects call the method provided 'com' to send and receive messages. When the static method bondObject.initbond(int port) is invoked, an instance of 'bondCommunicator' will be created and assigned to 'com'. 'com' is running as a separate thread called 'messaging thread'. This messaging thread implements all the function of message sending, receiving and delivering. Following is the skeleton of Bond class 'bondCommunicator'.

```
public class bondCommunicator extends bondObject implements Runnable {

 private bondCommunicationEngine ce;   // unicast communication object
 private bondMPCommunicationEngine mpce;  // multicast communication object

 public bondCommunicator(int port, String comengine) {
  // create unicast and multicast communication objects here;
 }

 public void send(bondShadow bs, bondKQML m) {  // unicast message delivery
   if (bs.local == null) {
     ce.send(bs,m);
   } else { // call distribute directly
     distribute(m, bs.local);
   }
 }

 public boolean multicast(bondKQML m) {  // multicast message delivery
    if( mpce.myMPDaemon == null )return false;
    m.setParameter(":sender_unicast_address", localaddress+":"+localport);
```

```
    mpce.multicast(m);
    return true;
  }

  void distribute(bondKQML mess, bondObject bo) {  // distribute the message to receiver

    String receiver = (String)mess.getParameter(":receiver");
    if (receiver.startsWith("bondID")) { // it is object's bondID
        bo = (bondObject)(dir.objects.get(receiver));
        new SubMessageThread(bo,mess);
        return;
    } else {
        if (mess.performative == PF_BROADCAST) { // this is a message sent to all objects
          for(Enumeration e = dir.alias.getAllObjects(receiver); e.hasMoreElements(); ){
              bondObject bonext = (bondObject)e.nextElement();
              new SubMessageThread(bonext,mess);
          }
          return;
        }else{  // it is a message sent to an alias
          bo = dir.alias.getAnObject(receiver);
          new SubMessageThread(bo,mess);
          return;
        }
    }
  }

  public void run() {
      bondKQML m;
      while(true) {
        m = ce.receive ();  // query the incoming unicast message in a non-blocking way
        if (m != null) {
            distribute(m, null);
        }
        if( mpce.myMPDaemon == null)continue;
        m = mpce.receive ();  // query the incoming multicast message in a non-blocking way
        if (m != null) {
          distribute(m, null);
        }
      }
  }
}

/**
   This class implements the new thread which is currently created at every arrival of a message.
*/
class SubMessageThread extends Thread {
   bondObject bo;
   bondKQML mess;

   SubMessageThread(bondObject bo, bondKQML mess) {
     this.bo = bo;
     this.mess = mess;
     this.start();
   }

   public void run() {
```

```
    bondObject preemptive = (bondObject)bo.get("Preemptive");
    if (preemptive != null) {
       preemptive.say(mess,null);
    } else {
       bo.say(mess, null);
    }
  }
}
```

Each instance of class ‘bondCommunicator’ will contain two communication engines. One is an unicast communication engine called ‘ce’; another one is a multicast communication engine called ‘mpce’.

Each Bond Object calls the ‘send()’ method to send a message to another object by using the unicast communication engine, and calls the ‘multicast()’ method to send a message to a group of objects by using the multicast engine.

The ‘com’ object also checks the underlying unicast and multicast communication engines in a non-blocking way to receive a message and deliver it to the receiver. There can be three types of receivers: receiver can be the BondID of an object; receiver can be the alias of an object; receiver can also be the alias of a group of objects. Alias is a way to let an object have different names. One Bond object can register to the local directory with several aliases, the ‘com’ object uses the ‘distribute()’ method to delivery the message to the right object(s).

Bond supports several different communication mechanisms, such as reliable message delivery, best-effort message delivery and multicast message delivery. To make ‘bondCommunicator’ independent and hide the details of underlying communication mechanism, Bond uses an abstraction called communication engine to describe the interface between ‘bondCommunicator’ and the real communication objects. Following is the definition of this interface:

```
public interface bondCommunicationEngine {
  public   void          send(bondShadow bs, bondKQML m);
  public   bondKQML   receive();
  public   void          sendObject(bondShadow bs, bondObject bo, String in_reply_to);
}
```

The ‘send()’ method is used by the communicator to send out a KQML message through the real communication object; the ‘receive()’ method is used by the communicator to receive a KQML message from the underlying communication fabric; the ‘sendObject()’ method is used to send a Bond Object through the underlying communication fabric to a remote site.

Currently Bond supports four types of communication mechanisms. They are implemented as Infospheres-based, TCP-based, UDP-based, and IP-multicast based communication engines. Infospheres is an infrastructure developed by Caltech, Bond only uses the message oriented peer to peer communication facility of Infospheres. All

these four communication engines have implemented the 'bondCommunicationEngine' interface, which provides a uniform access interface to different communication mechanisms.

These four communication engines can be divided into three types based on the services provided: reliable unicast message delivery, such as Infospheres-based and TCP-based communication engine; best-effort unicast message delivery, such as UDP-based communication; and multicast message delivery, such as IP-multicast based communication engine.

Table 1 shows the performance evaluation result of three communication mechanisms.

	UDP-based	TCP-based	Infospheres-based
KQML Message	30~40	45~55	90~110 milliseconds
Object Message	50~60	60~70	70~90 milliseconds

It seems:
1) When sending a KQML message, UDP is about 3 times faster than Inforspheres, TCP is about 2 times faster than Inforspheres.
2) When sending a object message, UDP is about 1.5 times faster than Inforspheres, TCP is about 1.2 times faster than Inforspheres.
3) Sending a basic bond object in Inforsphere is faster than sending a KQML message, but in TCP & UDP, it's slower than sending KQML message.
4) IP-multicast based communication engine takes the same time as UDP-based communication engine when sending a KQML message to only one receiver. It will cost the same time when sending this message to a group of objects, but the time needed by UDP-based engine will be (number of receivers * time_to_send_one_message).

When choosing the communication engine, Bond programmers should make their decision based on their application's functional and performance requirements. The default communication engine used in Bond is UDP-based engine.

2. Multicast Group Communication in Bond

Many distributed applications can benefit from efficient mechanism for one-to-many and many-to-many conversations. Bond provides an IP-multicast based group communication mechanism to support the development of these types of applications.

As we saw in the class definition of 'bondCommunicator', each instance of this class will contain two communication engines. One is an unicast communication engine called 'ce', it can be Infospheres-based, TCP-based or UDP-based communication engine; another one is an IP-multicast based communication engine called 'mpce'.

Each multicast communication engine, when initiated, will start a daemon listening on a unique multicast socket address 228.5.6.7:5678, where 228.5.6.7 is a class-D IP multicast address temporarily chose for Bond's mutlicast. A datagram sent to a class-D address will be received by all the applications that are listening on the same address. So this IP-

multicast based communication engine provides a way for a Bond object to send a message to a group of Bond objects.

The most important problem in multicast communication is the management of group membership. There are two different ways to do group membership management, sender initiated and receiver initiated. In sender initiated group management, the message sender can explicitly control the membership of a multicast group, or in another word, the message receivers. While in receiver initiated group management, each receiver can voluntarily join and leave a multicast group without the involvement of the message sender.

Bond achieves group membership management in a distributed and flexible way, which can accommodate both sender initiated and receiver initiated group management. In the following, we will discuss the multicast in Bond step by step.

2.1 Deliver Multicast Message to Group Members

As we pointed out in the first section, when the Bond communicator receives a message, the reciever's identifier in this message can be any of the following three: a BondID, an alias of a bond object, an alias of a group of objects.

When the local communicator receives a message destined to an alias, it will deliver this message to all the local objects registered with this alias. Because all multicast messages in Bond have the name of multicast group as the target receiver, so to receive the multicast message, each member object inside a multicast group should register to the local directory with the multicast group name as an alias.

2.2 Multicast Group Membership Control Subprotocol

Each Bond object can voluntarily join or leave a multicast group by using the following methods which is provided by the local directory:

Join a multicast group: **dir.addAlias**(groupname, object);
Leave a multicast group: **dir.deleteAlias**(groupname, object);

This is very useful to achieve the receiver initiated multicast group management.

Bond also provides a way to remotely control a Bond object to join or leave a multicast group, this task is finished by using the Multicast Group Control subprotocol. Following are the two messages in this subprotocol:

(**achieve :subprotocol** *MulticastGroupControl* **:content** *joinMulticastGroup* :groupID groupname)
(**achieve :subprotocol** *MulticastGroupControl* **:content** *quitMulticastGroup* :groupID groupname)

This is very useful to achieve the sender initiated multicast group management. Actually, Bond provides a dedicated object to finish this task. It is the multicast-mode 'bondVirtualNetwork'.

2.3 Bond Virtual Network

Class 'bondVirtualNetwork' defines a multicast group in Bond. It contains the shadows of a group of Bond objects and also provides the ways to send a message to all of them. It can be initiated into two modes: unicast mode or multicast mode. In unicast mode, 'bondVirtualNetwork' will send the message to all group members one by one using the unicast message delivery. While in multicast mode, 'bondVirtualNetwork' will send the message to all group members by using the IP-multicast based communication engine.

Class 'bondVirtualNetwork' provides the following methods to perform the sender initiated group management and message multicasting:

add(bondShadow**)**: Notify a remote object to join this multicast group by using the multicast group control subprotocol;

delete(bondShadow**)**: Notify a remote object to leave this multicast group by using the multicast group control subprotocol;

multicast(bondKQML m, bondObject sender): the first parameter indicate the message, the second parameter indicate the sender object.