Measurement of Middleware Performance over HF Radio and Satcom Links

Kjell Rose, Bjørn Jervell Hansen and Anton B Leere
Norwegian Defence Research Establishment (FFI)
P. O. Box 25, NO-2027 Kjeller, Norway
Phone: +47 63 80 70 00
E-Mail: {kjell.rose | bjorn-jervell.hansen | anton-b.leere@ffi.no}

Keywords: middleware, client-server systems, measurement, satellite communication, simulation, radio communication, command and control systems, distributed processing.

Abstract

In a geographically distributed Command and Control Information System (C2IS), the different components can be tied together by middleware technology like the Control of Agent Based Systems (CoABS) Grid. The CoABS Grid is a technology that integrates heterogeneous software applications, objects, services, and agents using Sun's Jini network technology.

The Norwegian Defence Research Establishment (FFI) has examined the performance of the CoABS Grid over High Frequency (HF) radio and satellite communication (satcom) links to examine its suitability for these communication media typically used in geographically distributed military systems.

FFI has developed a flexible demonstrator for investigation and experimentation of a range of technologies for C2ISs. The demonstrator has been equipped with a HF radio link consisting of two modems connected back-to-back and a satcom link simulator capable of a transfer rate from 9600 bit/s to 2 Mbit/s. This paper presents the results of tests with the CoABS Grid communicating over the HF radio and satcom links.

1. Introduction

In order to study various aspects of the maritime command and control, FFI has developed a demonstrator consisting of a High Level Architecture (HLA) based simulation environment, software components and legacy systems tied together by middleware. A subject of special interest is how geographically distributed picture compilation can be done relying on communication media with less that Ethernet bandwidth. This paper presents the results of tests with a specific middleware, the CoABS Grid, over HF radio and satcom links.

The paper is organized as follows. In section 2, the CoABS Grid is presented. The CoABS Grid is based on Jini, which is also presented in this section. The test configuration used is described in section 3, while the communication equipment setup is presented in section 4. In sections 5 and 6 the HF radio and satcom link measurements are presented, while the paper closes with a conclusion in section 7.

2. Jini and the CoABS Grid

2.1 *Jini*

Effective linking of battlespace entities requires the establishment of a robust, high-performance information infrastructure, or infostructure, that provides all entities with access to high-quality information services. The infostructure has to also allow the entities to link in a dynamic way, allowing them to form networks adapted to the ongoing military operation. This can be achieved by using Sun's Jini Network Technology [Sun Microsystems, 2002 i]. Jini provides a set of Java packages that allows reliable distributed applications to be built upon underlying unreliable networks.

The core of a Jini network, called a Jini community, is the look-up service (LUS). In the LUS, all available services announce their capabilities, and all clients use the LUS to search for services.

When services register with the LUS, they register a proxy object corresponding to the service along with a description of their capabilities. This proxy object, a service representative (service rep), is then downloaded from the LUS by any client asking for this service and all access to the service is handled by the service rep. The client only has to interact locally with the service rep, and doesn't know, or care, how the service rep fulfills its task. Details such as which other services are used and what protocols are used to interact with them are totally unknown to the client.

To keep the content of the LUS current, Jini introduces the concept of leasing. Services registering with a LUS are granted a lease: a limited period of time when the LUS will hold their entries. Before the lease expires, the service has to renew its lease if it is still alive and wants to be a member of the community. This means that a crashed or otherwise unavailable service's entry in the LUS will be deleted when the lease is expired.

2.2 *The CoABS Grid*

Control of Agent Based Systems (CoABS) is an agent research program funded by the US Defense Advanced Research Projects Agency (DARPA) and the US Air Force Rome Laboratories. It is investigating the use of agent technology to improve military command, control, communication, and intelligence gathering. It also seeks to enhance the dynamic connection and operation of military planning, command, execution, and combat support systems to quickly respond to a changing operational picture.

The CoABS Grid [Global Infotek, 2002] is middleware integrating heterogeneous agent-based systems, object-based applications, and legacy systems using Jini. The CoABS Grid provides helper utility classes, and shields the user from direct contact with Jini. Other additions to Jini provided by the Grid are messaging between agents, a logging service, and a security service.

In the demonstrator, the CoABS Grid is used as a wrapping of Jini, using the helper utility classes. In addition, the CoABS Grid's GridManager is used for:

- keeping track of the members of the Jini community,
- managing the Java Remote Method Invocation (RMI) [Sun Microsystems, 2002 ii] daemon necessary to run the LUS,
- managing the LUS itself, and
- managing the Hyper Text Transfer Protocol (HTTP) server needed for downloading class files.

3. **Test configuration**

An operational configuration would consist of two Local Area Networks (LANs) connected by a satellite communication link or a HF radio link as shown in Figure 1. Both LANs have complete CoABS Grid installations consisting of a GridManager, LUS, HTTP server, and various clients and services constituting the system.

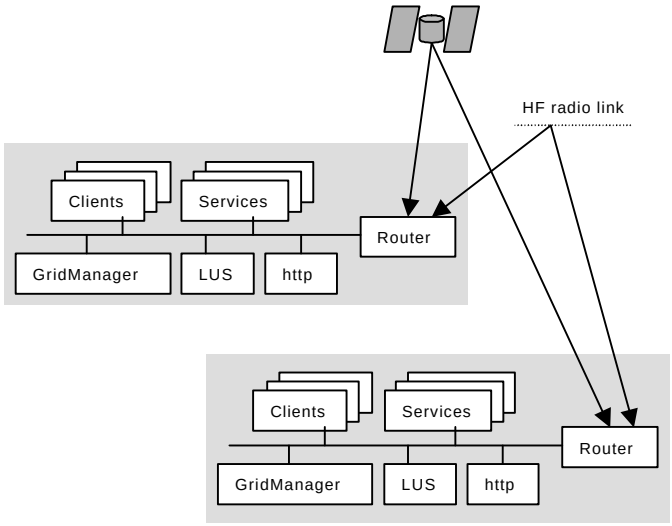

Figure 1 The operational configuration

In order to have full control of the measurements, the simplified test configuration shown in Figure 2 was used. The system consists of only one client and one service. The service registers with a local LUS, and the client invokes the service remotely via a satcom link or a HF radio link. The time it took for the client to establish contact with the LUS (discovery) and the service (lookup) was measured. The measurements were made both with a local and with a remote HTTP server. The HTTP server is used for downloading class files. The service is a track reporting service, and the link capacity in tracks per minute was measured.

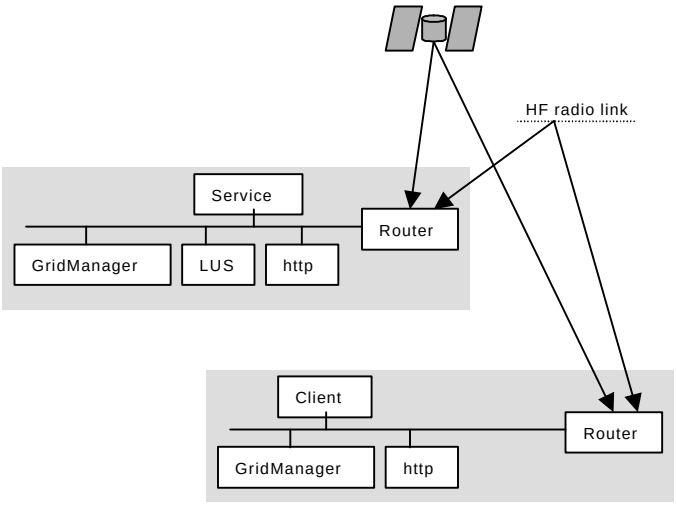

Figure 2 The test configuration used for the measurements.

4. **Communication equipment**

The communication equipment setup is shown in Figure 3.

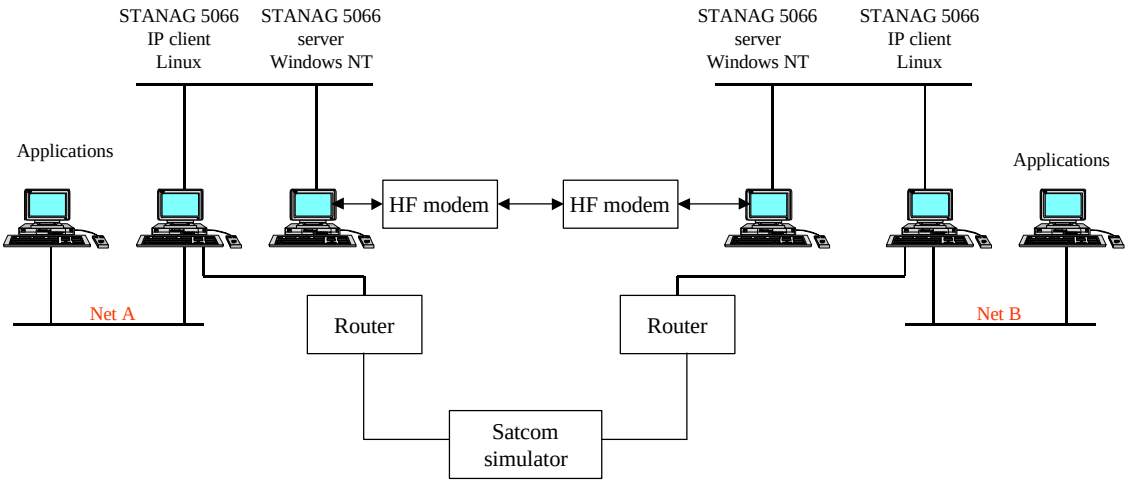

Figure 3 The communication equipment setup.

The layout shown in Figure 3 allows for two alternative, and easily controllable, routes between the applications running on the application computers. The possible routes are:

1. Simulated HF radio wide area network (WAN) (simplex radio net)
2. Simulated satellite WAN (duplex satellite channel)

Internet Protocol (IP) is used for data transfer in all cases. The only differences observed by the applications on the various means of transferring information are the speed of transmission and the delay introduced by the link.

4.1 *HF radio communication*

The HF radio link is a simplex link, that is the link can only transmit data in one direction at any moment. One radio is transmitting while the other radio is receiving. It takes some time to turn the transmission around, due to radio and modem characteristics. The simulated link is implemented through the use of the following hardware/software components:

- two Harris RF-5710A HF modems set compliant to NATO STANAG 4285C waveform,
- two NATO STANAG 5066 servers running on Windows NT PCs, and
- two IP clients for STANAG 5066, running on Linux PCs.

Profile for Maritime High Frequency (HF) Radio Data Communications, NATO STANAG 5066 [NATO] is a link and physical layer protocol designed for use on HF radio channels. The aim of this profile is to define the functions and interfaces required for networked, error-free communication over HF radio channels, nominally for beyond-line-of-sight communications. The HF profile provides interoperability at the two major interfaces: first, the "common air interface", describing how information is exchanged between nodes by radio; and second, the non-HF interfaces which allow external users or clients to interact with the subnetwork and with each other over the subnetwork.

The link layer protocol is designed to transfer special packets, called S-primitives, over HF radio implementing automatic retransmission request, ARQ, and non-ARQ protocols. There are various vendors offering implementations of STANAG 5066. The implementation used in this setup was from Marconi Mobile Limited [Marconi, 2001]. The Marconi implementation of STANAG 5066 allows for a variable link speed, depending on link quality. This feature was disabled during these tests.

Since STANAG 5066 is designed to transfer S-primitives, and not IP packets, an IP client utilizing STANAG 5066 for data transfer is used. Such an IP client software package has been implemented by NATO C3 Agency (NC3A), The Hague [Smaal, 2001]. This IP client was installed as shown in
Figure 3. Further information about STANAG 5066 and the IP client software can be found in [NC3A, 2002].

In order to activate routing over the HF radio link, the IP client software has to be running. This software creates a virtual point-to-point connection from Net A to Net B and vice versa.

4.2 *Satellite communication*

The simulated satcom link is full duplex.

Except for setting up the routing tables in the Linux PCs, satcom is implemented using only commercial off the shelf (COTS) units. These are:

- two SR3000 Internet ThinRouter [Moxa Technologies, 2002] from Moxa Technologies Co, acting as interfaces between the local LAN and the WAN and
- one Simulator 2 [Vocality International, 2002] from Vocality International Ltd. acting as the WAN.

The SR3000 Internet ThinRouter has two ports, one LAN port and one WAN port. The SR3000 then routes IP packets between the two ports.

The Simulator 2 has two WAN ports for connection to data terminal equipment (DTE) and simulates a satellite channel.

When satcom is to be used as a communication link, the routing tables in the Linux PCs have to be modified in such a way that all traffic between Net A and Net B are routed over the satcom simulator.

5. **The HF radio link measurements**

The performance of the HF radio link was measured using the configuration in Figure 2, and the measurements were performed in four steps:

- test of the discovery protocol,
- test of a simple socket connection,
- test of the Lookup Service (LUS), and
- test of the service invocation.

The ARQ protocol in STANAG 5066 was disabled during these experiments to prevent its messages from interfering with the measurements.

5.1 *Discovery*

Discovery is the process of finding and obtaining references to LUSs. There are two versions of the discovery protocol, the unicast and the multicast discovery protocols, using unicast Transmission Control Protocol (TCP) and multicast User Datagram Protocol (UDP) respectively.

A client or service using the unicast discovery protocol makes a TCP connection to a known IP address and TCP port, and sends a unicast discovery request to the LUS known to be located there. The LUS responds by sending a proxy to the requesting client or service. A client or service using the multicast discovery protocol, on the other hand, sends a UDP packet to a specified

multicast group to find any LUS located nearby. Any LUS listening to the specified multicast group then responds by sending a proxy to the client or service. Unicast TCP and multicast UDP are both described in [Li *et al*., 2000].

In the application used in these experiments, services always register with a local LUS, using multicast discovery to obtain a reference to it. All communication between the services and the LUS, including renewal of leases, are handled locally.

Although the HF radio link does support multicast, it is not used for this application. This was done to prevent local traffic, for instance leasing traffic between a service and a LUS, from being transmitted over HF. Clients looking for services on the other side of the HF radio link therefore have to use the unicast discovery protocol to find the LUS.

The unicast discovery protocol was tested at different bandwidths, giving the response times in Table 1.

Discovery						
Bandwidth [bit/s]	75	150	300	600	1200	2400
Response time [seconds]	-	-	-	236	142	90

Table 1 Discovery response times using the unicast discovery protocol at different bandwidths

At 600 bit/s discovery took 4 minutes, and at lower rates it will take even longer. Although this seems like a long time to wait, it must be taken into consideration that the HF radio link may be the only means of communication.

At 300 bit/s and less, the socket connection failed. This is further investigated in the next subsection.

5.2 ***Limitations of the socket connections***

The reason for the failing socket connections is believed to be that Reggie, Sun's reference implementation LUS, uses the standard socket classes (java.net), and that these classes are not able to cope with the long delays introduced by the low bandwidths.

In order to find the limitations of the Java socket connection, a simple socket application was tested. The example is taken from [Li *et al*., 2000], and the example consists of two applications, a server and a client, with a TCP socket connection between them. The simple client prints out anything the server returns, which in this case is the current server date and time. The server connection was tested at different bandwidths and the response times are given in Table 2.

Socket connection						
Bandwidth [bit/s]	75	150	300	600	1200	2400
Response time [seconds]	-	23	15	11	8.6	7.3

Table 2 Response times for the simple socket test

At 75 bit/s the socket failed in two different ways. The socket constructor timed out and returned a java.net.NoRouteToHostException or the socket failed to deliver the results. In the latter case, the socket tried to connect a second time before the response for the first attempt was ready. This second attempt generated added communication load, which in turn lead to a timeout.

The fact that a simple socket connection failed at 75 bit/s is an indication that (java.net) sockets are not adapted to low bandwidths.

5.3 *Lookup*

When a client has obtained a reference to the LUS it can look for the services it needs. The LUS is then called with a service description template and returns a service description along with the service reps for the matching services. This is called the lookup.

The class file (with the computer code) of the service rep is not stored in the LUS but is downloaded from a HTTP server. Measurements were performed both with a local and a remote HTTP server. When using a remote HTTP server, the downloading was done over the HF radio link. This was very time consuming and was failing for bandwidths lower than 2400 bit/s. With a local HTTP server, no downloading of class files was done over the HF link. In the latter case, the participants of the HF radio link would have to make sure that they have the correct service reps stored locally by communication means other than the HF radio link.

The lookup was performed with a UDP based service rep and a RMI based service rep and tested at different bandwidths giving the response times (in seconds) in Table 3. The only difference between these two lookups were the sizes of the service reps and the fact that when using the RMI based service rep, two class files had to be downloaded during lookup: the class file of the service rep and the stub file of the service. The latter file is necessary to manage the RMI communication between the client and the service. When using the UDP based service rep only one class file had to be downloaded. This explains the difference in lookup response times seen in the table.

Lookup							
Bandwidth [bit/s]		75	150	300	600	1200	2400
Local HTTP server	UDP based service rep	-	-	-	169	53	36
	RMI based service rep	-	-	-	262	114	73
Remote HTTP Server	UDP based service rep	-	-	-	-	-	249
	RMI based service rep	-	-	-	-	-	406

Table 3 Lookup response times (seconds) with different bandwidths

The long response times using a remote HTTP server indicates that a local HTTP server should be used during the lookup if possible. The HTTP server failed to provide the needed service reps at 1200 bit/s and below when a remote HTTP server was used.

Again the socket connections were not able to cope with bandwidths of 300 bit/s and below. The reason for this has been discussed in section 5.2.

At 600 bit/s, the lookup took 3 minutes, and at lower rates the lookup times will be even longer. It is possible to reduce the lookup time by making the service description and the service rep more compact. The service rep could be made more compact by declaring fields in the service rep to be transient and thus not stored in the LUS.

5.4 *Service invocation*

A client program only needs to know the interface type of the service rep to invoke a service. The service rep in our case implements the interface ReporterInterface, shown in Figure 4.

```
public interface ReporterInterface extends Remote
{
  UnitPoint      getUnitPointReport     (int reportedTrackID, int reportTime) throws RemoteException;
  SurfaceUnit    getSurfaceUnitReport   (int reportedTrackID, int reportTime) throws RemoteException;
  long           register               (long ev_id, RemoteEventListener listener) throws
                                                                                  RemoteException;
  void           deregister             (RemoteEventListener listener) throws RemoteException;
}
```

Figure 4 The ReporterInterface

The methods are getUnitPointReport, which returns a ships position, course, and speed, etc, and getSurfaceUnit, which returns classification data, ships name, type, class, etc. The register method is for registering interest in events created when new data are available.

Two service reps were made for these experiments: one implemented using Java RMI in a straightforward way and the other tailor made for low bandwidths using UDP datagram sockets.

5.4.1 *UDP based service rep*

When the client registers with the service using the ReporterInterface register method, the UDP based service rep sends a registration datagram to the service with the host and the port-number of the client, and starts to listen for track messages. Each message contains both position and classification data and is compressed into 134 bytes. When track messages are received, the service rep generates an event and the client may fetch the data using the getUnitPointReport and getSurfaceUnitReport methods to access the service rep.

The measurements were taken by saturating the HF radio link. The rate of the track messages were controlled on the server side, and recorded on the client side. The rate was increased until the client side could not keep up with the server side.

The sizes of the track datagrams were varied from 134 bytes to 1500 bytes. The results using the UDP based service rep are given in Table 4.

Service invocation							
Bandwidth [bit/s]		75	150	300	600	1200	2400
UDP Datagram Size	134 bytes	0.93	1.6	4.6	13	27	55
	700 bytes	0.54	0.70	1.4	4.0	8.1	14
	1,400 bytes	0.26	0.56	0.99	2.0	4.3	8.6
	1,500 bytes	-	0.41	0.91	1.9	4.0	7.5

Table 4 The transfer rate (tracks/min) UDP packets at different bandwidths

The measurements for bandwidths below 600 bit/s were made by first doing the discovery and lookup at a higher bandwidth and then, when the connection between the client and the service was established, the bandwidth was turned down to 300, 150 and 75 bit/s.

The HF data link is a simplex link and it takes some time to change the direction of a transmission. Therefore the system sent several UDP datagrams stacked together in one conitinous transmission, the data popped out at the receiving side as soon as it was ready. UDP datagrams larger than 1480 bytes were fragmented into fragments of no more than 1480 bytes. These fragments would normally be transmitted back to back in the same transmission. The IP client failed for fragmented datagrams (1500 bytes) at 75 bits/s because the IP reassembly time was exceeded.

As can be seen from Table 4, the transfer rate was not linear in the datagram size. Thus, to get an effective use of the HF radio link, 11 tracks of 134 bytes could be packed together in one UDP datagram of 1,474 bytes achieving a transfer rate of about 8 datagrams/min at 2400 bit/s. This would give a total of 88 tracks/min as opposed to only 55 tracks/min if each track were sent separately. A total of about 2 tracks/min could be transmitted with the same packing at 75 bit/s. This is shown in Figure 5.

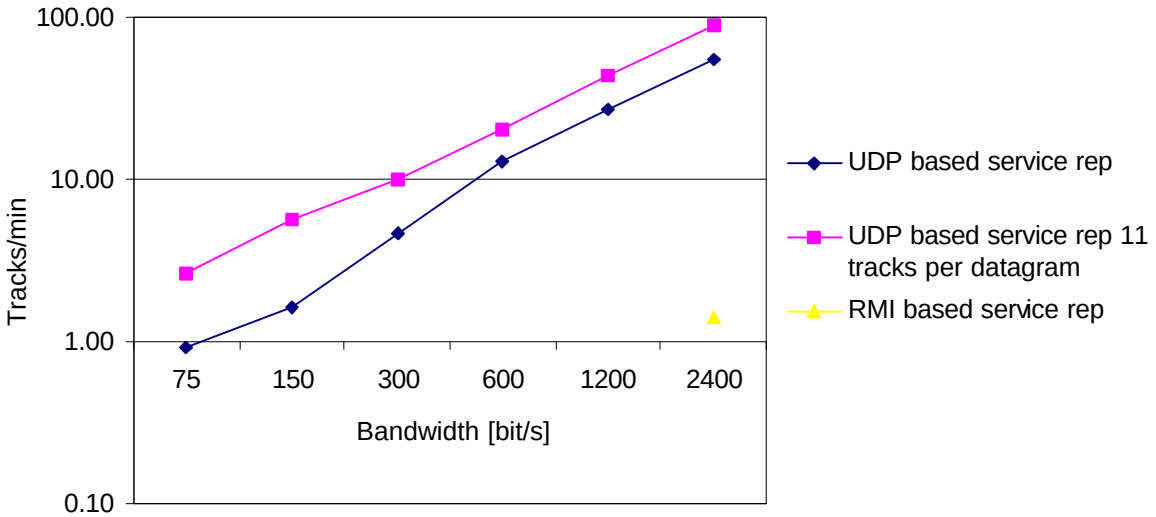

Figure 5 The transfer rate (track/min) for the HF radio link

5.4.2 RMI based service rep

The results from testing the RMI based service rep can be seen in Table 5.

Service invocation						
Bandwidth [bit/s]	75	150	300	600	1200	2400
Transfer rate, RMI [tracks/min]	-	-	-	-	-	1.4

Table 5 The transfer rate (tracks/min) using RMI at different bandwidths

The RMI based service rep used in these experiments is almost useless at these bandwidths. A transfer rate of 1.4 tracks/min at 2400 bit/s is only 1.5% of the performance that is possible with the tailor made UDP based service rep.

It is important to point out that RMI was used in a straightforward manner in these experiments. Much of the same optimization done in the UDP based service rep could have been applied inside the framework of RMI. The serialization process could have been improved, a custom socket factory could have been made, and finally multithreading could have been used letting each event start a new thread. This was not done during these experiments because it was much easier to tailor make a service rep implementing the ReporterInterface. In fact it took only about one day to complete the UDP based service rep. To optimize the RMI based service rep would probably have taken much longer.

6. **The satcom link measurements**

The satcom link measurements were made with a constant delay of 300 ms, representing the transmission delay of a geostationary satellite.

The response times of the unicast discovery protocol are given in Table 6, and show that even at 9600 bit/s a client use only 8 s to discover the LUS.

Discovery				
Bandwidth [bit/s]	9.6 k	64 k	256 k	2 M
Response time [seconds]	8	5	5	5

Table 6 The response times of the discovery protocol using the satcom link

The lookup was tested with the UDP based and the RMI based service rep with a remote and a local HTTP server, as for the HF link measurements. Even at these high bandwidths, the lookup is considerably faster when using a local HTTP server instead of a remote HTTP server as can be seen in Table 7.

Lookup

Bandwidth [bit/s]		9.6 k	64 k	256 k	2 M
Local HTTP server	UDP based service rep	4.4	1.9	1.6	1.5
	RMI based service rep	6.9	3.8	3.4	3.3
Remote HTTP Server	UDP based service rep	26	9.3	6.9	5.8
	RMI based service rep	30	9.8	8.3	7.8

Table 7 Lookup response times in seconds using the satcom link

The service invocation using the satcom link was tested on a saturated link with a fixed UDP packet size of 134 bytes. The results of the measurements using the UDP based service rep given in Table 8 and Figure 6, show that 430 tracks/min can be transferred at 9600 bit/s, increasing to 2900 tracks/min at 64k bits/s and higher. At 64 kbits/s and higher bandwidths the network is not the limiting factor.

Service invocation				
Bandwidth [bit/s]	9.6 k	64 k	256 k	2 M
Transfer rate, UDP [tracks/min]	430	2900	2900	2900

Table 8 The transfer rate using an UDP based service rep at different bandwidths

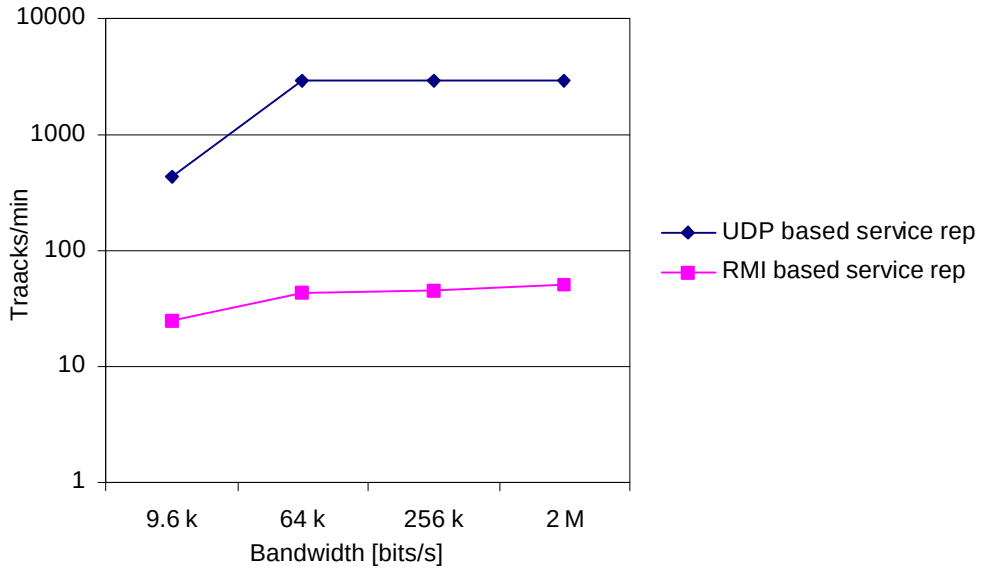

Figure 6 The transfer rate (tracks/min) for the satcom link

The measurements of the service invocation using the RMI based service rep are given in Table 9 and Figure 6. These measurements show that at 9600 bit/s, 25 tracks/min can be transferred.

Service invocation				
Bandwidth [bit/s]	9.6 k	64 k	256 k	2 M

Transfer rate, RMI [tracks/min]	25	43	46	50

Table 9 The transfer rate using a RMI based service rep

Finally the influence of a delay when using the RMI based service rep was tested with the delays on the satcom link varying from 0 to 2 seconds. The bandwidth was kept constant at 2 Mbit/s. The results in Table 10 and Figure 7 show a nearly linear relation between the transfer rate and the delay.

Discovery, lookup and service invocation									
Delay [seconds]	0	0.025	0.050	0.100	0.200	0.300	0.500	1	2
Discovery [seconds]	0	0	1	1	3	5	8	16	29
Lookup [seconds]	0	1.3	1.7	3.8	5.2	7.6	13	24	45
Transfer rate, RMI [tracks/min]	600	429	250	136	72	50	30	15	8.1

Table 10 The results of discovery, lookup, and transfer rate with a RMI based service rep, varying the delay

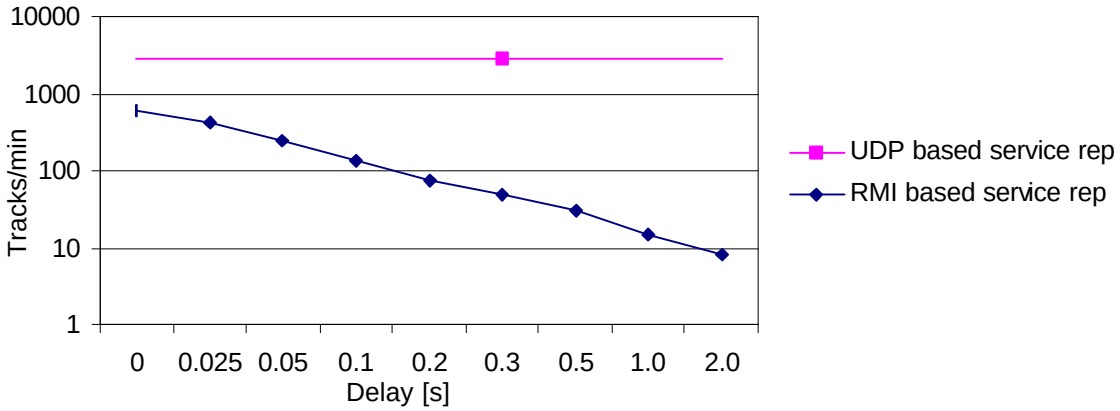

Figure 7 The influence of delay when using the RMI service rep

7. **Conclusion**

In this paper, results from testing the CoABS Grid / Jini over a HF radio and a satcom link have been presented. The results show that middleware technology can support distributed picture compilation, but the use of a HF radio link requires a tailor made protocol for low bandwidths in order to get adequate capacity.

The main obstacle to using Jini for low bandwidths at the present time, is the socket packages. They are the foundation that all modern data exchange is based on. Presently they are not well adapted to low bandwidths and fails at various bandwidths depending upon the amount of data that is transferred. It should be simple to fix this problem, as it is only a matter of selecting wider values for timeout, etc.

The Jini technology's ability to hide the implementation details and protocol of the service invocation while offering a standard interface is well suited for systems with limited bandwidth. This allows a developer to tailor make a protocol for limited bandwidths and hide the protocol from the client behind a standard interface.

The low bandwidth of the HF radio link is not well suited for straightforward use of Java RMI. RMI is a general method and has to work for all sizes of data. By utilizing knowledge about the size of the data it was possible to pack the data into a datagram and use UDP as the transport protocol.

The RMI based service rep has a low throughput on the satcom link due to the transport delay, but it does not use much of the links capacity, which is free for other services. The UDP based service rep is not influenced (capacity vice) by the transport delay. The picture compilation would only take a fraction of the capacity of satellite communication link, and the link could be shared by other services much like a LAN.

The CoABS Grid / Jini over HF radio and satcom links looks promising with regard to a future use in picture compilation.

8. *Acknowledgements*

Many thanks to Jan-Willem Smaal, NC3A, The Hague, for helping us getting IP over STANAG 5066 up and running.

9. References

[Global Infotek, 2002] Control of Agent Based Systems: http://coabs.globalinfotek.com

[Li *et al*., 2000] Sing Li, Ronald Ashri, Milé Buurmeijer, Eric Hol, Bob Flenner, Jerome Scheuring and Andrew Schneider. *Professional Jini*. Wrox Press Ltd., Birmingham, UK, 2000.

[Marconi, 2001] Marconi Mobile Limited. *Marconi Digital HF, Installation and Configuration Guide*. March 2001.

[Moxa Technologies, 2002] Moxa Technologies: http://www.moxa.com

[NATO] NATO C3 Agency, The Hague, Radio Branch. *Profile for (HF) Radio Data Communications STANAG 5066*. Version 1.2.

[NC3A, 2002] NATO C3 Agency: http://elayne.nc3a.nato.int/

[Smaal, 2001] Smaal J-W. IP *Multicast over STANAG 5066* NATO C3 Agency Technical Note 839 [NATO unclassified]. January 2001.

[Sun Microsystems, 2002 i] Jini Network Technology: http://wwws.sun.com/software/jini

[Sun Microsystems, 2002 ii] Java Remote Method Invocation: http://java.sun.com/products/jdk/rmi/

[Vocality International, 2002] Vocality International Ltd.: http://www.vocality.com