THE JOINT
ARCHITECTURE
FOR
UNMANNED SYSTEMS

Transport Interface

Version 1.4

October 25, 2004
1. Introduction
The purpose of this document is to specify the Transport Layer format and associated protocols required to implement the proposed JAUS-over-IP transport standard for the 2.75 JAUS Interoperability Experiment.

This document is divided into five sections:
1. Introduction – This section.
2. Network Environments – Describes the network topologies in which this specification is intended to function.
3. JAUS-Over-IP: Describes the proposed JAUS-Over-IP standard.
   3.1. Packet Format: Format of an individual UDP or TCP message.
   3.2. Network Infrastructure: Describes how to set-up the network infrastructure/addressing.
   3.3. Broadcast Implementation: Describes how the broadcast semantics on JAUS are implemented in the transport layer.
A. Appendices
   A.1. Acronym Reference – Expands the various abbreviations used in this document to their full names.
   A.2. Document Reference – Indicates where to find all documents, websites, etc. referred to in this document.
   A.3. New Messages – Descriptions of any new messages required to implement the proposed formats/protocols.
   A.4. Configuring Routing for iptables – Describes the procedures by which the iptables functionality is typically built and configured in Linux. This is a brief guide/overview for those who are less familiar with IP networking, and is intended to provide some basic advice/insight.
   A.5. Example Code for Transport & Network Infrastructure – Describes reference code for an example implementation of the packet formats and network infrastructure required to implement this specification.

Where possible, this document will provide references, examples and pseudo-code in order to be as clear as possible. While the author has attempted to verify each such instance, there may of course be errors. The examples presented here are done using Linux. Users of Windows and other operating systems will have to go through some extra effort here – the author hopes that these examples will at least point the way.

2. Network Environments
The proposed JAUS-Over-IP Transport standard is intended to function in the typical network topology depicted in Figure 1.
Figure 1: Typical Network Topology for JAUS-Over-IP Transport

Note that when the robot supports a payload network, it operates as a multi-homed host (that is, a computer with two, independent network interfaces). This network layout allows the robot itself to act as a firewall/proxy for the payloads behind it. This is the most typical design used for payload-bearing entities like robots.

However, while the design above is the most common, there are two significant minority cases here that must also be accommodated. The first such case is a payload that provides a communications interface on the Subsystem Network. An example is depicted in Figure 2.

Notice that in this case, it is “Payload 1” that is the multi-homed host, as it is this processor that has two, independent network interfaces and must act as a firewall/proxy for the rest of the subsystem. The developer of “Payload 1” must fulfill that constraint, in effect making the payload an IP router for the subnets on which it has network interfaces (assuming, of course, that these are IP-based networks to begin with).

The second, alternative network environment we must consider is that of

Figure 2: Alternative Network Topology for
a robot and its payloads existing on a single network with a single COTS router providing the interface to the radio network. This type of network environment is depicted in Figure 3.

Note that if one were to consider the “COTS Router with Radio Interface” to be just another payload, this case has the same topology as in Figure 2. The difference is that with a payload specifically designed to operate in a JAUS environment, the routing parameters required to efficiently implement the JAUS-over-IP standard are more likely to be accessible. With a COTS router, the developer is more likely to have to work with whatever functionality that router provides and adjust the network infrastructure accordingly.

A Final Note: To this point, none of the attributes of the Subsystem Network topology and/or infrastructure have been specified. The JAUS-Over-IP transport imposes no requirements in this area. It is up to the System Designer to create this infrastructure and to configure the elements within it correctly for that environment.

Designers of robots, OCUs and payloads should keep this flexibility in IP addressing and infrastructure in mind, and provide convenient interfaces for modifying the parameters of the network environment, such as IP addresses and subnets, TTL values, name resolutions for hosts and services, etc.

3. JAUS-Over-IP
The proposed standard for JAUS-Over-IP has three components: The first is the format of an individual data packet; the second is the network infrastructure required; the third is the means by which broadcast semantics are implemented on the subsystem and payload networks. Each of these is described in turn in the following subsections.

3.1 JAUS-Over-IP Packet Format
The protocol stack for JAUS-Over-IP (UDP version) is identical to the protocol format used in prior OPC experiments. I.e., in a JUDP packet, the 16-byte JAUS application header is maintained, and is preceded by the text string “JAUS01.0”
3.2 Network Infrastructure
For purposes of the interoperability experiment, the Subsystem Network will be a single, flat IP subnet with static addressing.

Static addressing will be done at the interoperability experiment itself.

The Payload Network will be whatever the host platform wants it to be – it is up to the designer of that platform to specify this network infrastructure – as long as it conforms to one of the three network topologies identified above. Note that OCUs may have “payload” networks as well as robots – for example, a robot’s payload may be paired with a specific controller on the OCU side of the communications link. This experiment allows for this kind of configuration, should it be desired. However, it is expected that payloads will typically be attached to robots, not OCUs. We recommend retaining the 192.168.192.0/24 subnet from the 2nd Interoperability Experiment, but this is not required.

3.3 Port-Based Routing
The core transport-level change in OPC 2.75 is the addition of port-based routing to increase system efficiency and reduce message routing latency. To date, all JAUS-Over-IP routing has been done at the application level. An application, typically the Node Manager, is required to examine each packet and determine the IP address to forward the packet to. This was primarily due to the restriction that all JAUS traffic must travel over a single port – 3794.

An alternative approach is to take advantage of built in IP routing capabilities present in all modern operating systems. In this approach, Node 1 (the “Primary Node”) continues to communicate over port 3794. Payload Nodes reside on an internal subnet (as has traditionally been the case), and also communicate over port 3794 on the payload network.

In the “normal” application-based routing approach, a packet destined for a payload node would be forwarded to the payload node via the Primary Node manager. Therefore, each packet sourced from, or destined to, the payload node, would have to pass through the primary node manager.

In the port-based routing approach, a table is created which maps port numbers on the subsystem network to IP address/3794 pairs on the payload network. This table can reside at the application level, or at the Operating System level. When it resides at the application level, there is no gain in routing efficiency, as each packet must still pass through the primary node manager. When OS-level port-forwarding is used, a packet destined to a payload node is not examined by the primary node manager application. Rather, it is forwarded by the operating system directly to the appropriate IP/Port on the payload network. The figure below illustrates the topology of a subsystem and payload network under this concept.

For the developer who wishes to use OS-level port-forwarding, appendix A.4 shows how to use the Linux iptables functionality to provide port-forwarding capabilities. An alternative approach is to maintain the table at the application level. This requires the fewest changes to one’s software base, but does not result in any additional efficiency gains.
The one issue not addressed in the above scenario is how the port-forwarded payload is ever discovered by the OCU in the first place – i.e. how the discovery protocol boots up the transport infrastructure on the non-3794 ports, given the multicast protocol currently in place for discovery.

For the OPC 2.75 experiment, the discovery of non-standard ports will not actually be implemented. Instead, an a priori mapping is specified, based upon Node numbers in the source ID fields of the JAUS messages:

Node 1: Addressed at Port Number 3794.
Node 2: Addressed at Port Number 10002.
Node 3: Addressed at Port Number 10003.
Node 4: Addressed at Port Number 10004.

Node Y: Addressed at Port Number 10000+Y.

A. Appendices

A.1: Acronym Reference

The terms in this list are used in this document; the definitions are provided here.

1. JAUS: Joint Architecture for Unmanned Systems.
2. COTS: Commercial Off-The-Shelf.
3. OCU: Operator Control Unit.
4. TTL: Time-To-Live.
A.2 Document Reference
The documents, websites, etc. listed here provide more detailed explanations and authoritative references for the information presented in this document.

1. iptables Tutorial: http://iptables-tutorial.frozentux.net/iptables-tutorial.html
2. JAUS Working Group: http://www.jauswg.org

A.3 New JAUS Messages
None.

A.4 Using iptables in Linux
The purpose of this section is to briefly describe the process by which the iptables facility is enabled and configured in Linux. The iptables functionality comes as a standard part of the more recent 2.4 Linux kernels, as well as in the 2.6 kernel line. The instructions in this section are written based on the 2.4.24 Linux kernel, but should be valid for most such cases.

Most of the information in this section is gleaned from the “iptables Tutorial”, referenced in Section A.2.

A.4.1 Configuring the Linux Kernel
The iptables functionality must be built into your Linux kernel in order for it to function. This is done by turning on the following options in the Linux kernel configuration:

- CONFIG_PACKET [required]
- CONFIG_NETFILTER [required]
- CONFIG_IP_NF_CONNTRACK [required]
- CONFIG_IP_NF_IPTABLES [required]
- CONFIG_IP_NF_NAT [required]
- CONFIG_IP_NF_FILTER [required]
- CONFIG_IP_NF_MATCH_STATE [required]
- CONFIG_IP_NF_TARGET_MASQUERADE [required]
- CONFIG_IP_NF_MATCH_LIMIT [optional – useful against DOS attacks, and so generally good hygiene to include this]
- CONFIG_IP_NF_FTP [optional – useful if you’re going to want to have ftp connections passed through the NAT service]
- CONFIG_IP_NF_TARGET_LOG [optional – useful for debugging]

For more detail on what these options actually do, see the “iptables Tutorial” reference listed in Section A.2 above, in Section 2.2 of the referenced document.

If you have a previously built kernel, you can determine if these options were selected by looking at the /usr/src/linux/.config file to see if they have a “y” or “m” value assigned to them. If you’ve never worked with building a Linux kernel, please talk to whoever did the original build of your kernel for help (there’s just too much variability in kernel functionality to lay out all the possible considerations here).
A.4.2 Building iptables

Once the kernel has been configured, you need to build and/or install the **iptables** package. This provides the utilities needed to manipulate this functionality. You may be fortunate and have a Linux distribution that provides an already-built **iptables** package – for instance, the RedHat distributions do so.

If you already have the iptables package, then just install it. If you do not already have this source code, it can be downloaded from [http://www.netfilter.org](http://www.netfilter.org).

The “iptables Tutorial” has information about patching more exotic options into the kernel – you probably want to ignore this, and just build the tools, by doing

```
make KERNEL_DIR=/usr/src/linux
make install KERNEL_DIR=/usr/src/linux
```

The value for the `KERNEL_DIR` variable should point to wherever the source tree is for your Linux kernel.

A.4.3 Booting-Up iptables

If you are running an older Linux distribution (say, RedHat 7.1), you will need to turn off the `ipchains` functionality before turning on `iptables`. If you are using the standard kind of Sys5 init procedure in your OS, this will probably require executing commands that look something like this:

```
chkconfig --level 0123456 ipchains off
service ipchains stop
```

To turn on the `iptables` functionality:

```
chkconfig --level 235 iptables on
service iptables start
```

If you get errors about “device or resource busy” in the `init_module` call when trying to start the `iptables` service (or if the service doesn’t start at all), you should unload the `ipchains` kernel module (assuming that your kernel has it compiled in as a module, as most kernels that have `ipchains` do):

```
rmmod ipchains
ldconfig
modprobe ip_tables
```

Then restart the `iptables` service. You can tell if the service is running by typing

```
service iptables status
```

It should print out “Table” entries for “filter”, “nat”, and “mangle” with 2-3 “Chain” entries under each.

Lastly, you must set the flag in the Linux kernel that enables ip-forwarding functionality:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

And you can make sure that the flag is set via:

```
cat /proc/sys/net/ipv4/ip_forward
```

This should return a zero if ip-forwarding functionality is off, and a one if ip-forwarding functionality is on. Note that you will have to do this every time you boot-up your system.
A.4.4 Configuring Routing for iptables
Now that the iptables functionality is up and running, it is time to configure its routing tables to implement the NAT (Network Address Translation) required to support payload networks. For purposes of this explanation we assume that the payload subnet for the interoperability experiment will be 192.168.192.0/24. This is a typical, “Class C” private subnet, and is in keeping with the IP address assignments from the 2nd Interoperability Experiment.

In this example, we will assume that the payload has been assigned IP address 192.168.192.101 and that it provides the following services:
- JAUS: Port 3794
- JAUS: Port 12345

To expose these services to the outside world, the following iptables commands would be used:
```
iptables –t nat –A PREROUTING –i eth0 –p tcp --dport 10001 –j DNAT --to 192.168.192.101:3794
iptables –t nat –A PREROUTING –i eth0 –p udp --dport 10001 –j DNAT --to 192.168.192.101:3794
iptables –t nat –A PREROUTING –i eth0 –p tcp --dport 10002 –j DNAT --to 192.168.192.101:12345
iptables –t nat –A PREROUTING –i eth0 –p udp --dport 10002 –j DNAT --to 192.168.192.101:12345
iptables –t nat –A POSTROUTING –o eth1 –j MASQUERADE
iptables –t nat –A POSTROUTING –o eth0 –j MASQUERADE
```
These commands assume that “eth0” is the network interface on the Subsystem Network.

The first four of the above commands rewrite incoming TCP & UDP packets on the “eth0” interface for port numbers 10001 and 10002, forwarding them on to IP address 192.168.192.101, ports 3794 and 12345 respectively.

The last two of the above commands cause the source address in packets to be rewritten so that:
- The packets headed onto the payload network from the subnet network look like they’re coming from the iptables host platform – in this case, 192.168.192.24.
- The packets headed onto the subnet network from the payload network look like they’re coming from the iptables host platform – in this case 192.168.128.18 (assuming that we’re following the example laid out below).

These commands are needed because they allow the source address in the IP headers to function as the return routing for any reply.

Jeff Wit provides an alternative means of exercising port-based routing, without the necessity of rewriting the source address inbound (or the return destination address outbound):
```
```
The purpose of the first command is to rewrite the destination address on packets coming in from the subsystem interface, forcing them to be routed to the appropriate payload IP address and port number. The purpose of the second command is to rewrite the source address on packets coming in from the payload interface, forcing them to identify their source as the robot’s port-forwarding interface.

If you wish to test this basic functionality (of either variety presented above), you may obtain three test platforms and configure their networks as indicated in the figure below, with Test Platform #2 running **iptables**. [Note that you will probably have to turn off the ipchains functionality on Test Platforms #1 & #3 as well in order to get them to accept incoming connections.]

Once you have confirmed basic network connectivity among the platforms on each subnet, execute the above commands on Test Platform #2, then compile and execute the following code listed in Appendix A.6.1 on the indicated platforms.

### A.5 Example Code for Transport & Network Infrastructure

Code provided in accompanying archive files. Code is available from the author upon request.