The ASP Virtual Networking (VNET) Component

23 July 2002

Abstract

This document describes the implementation of the virtual networking (VNET) component of the ASP EE. VNET is a simple and modular implementation of a network protocol stack written in Java. The ASP EE uses VNET to construct virtual topologies of ASP EEs which communicate with both reliable and unreliable datagram oriented protocols over point-to-point links.

1. Introduction

The ASP Virtual Networking (VNET) component provides a framework for building network protocol stacks in Java. These stacks are used by the ASP EE to construct virtual topologies of ASP EEs which communicate with both reliable and unreliable datagram oriented protocols over point-to-point links.

The structure of the VNET framework stacks layer objects to build protocols. Currently VNET has examples of layers 2-4 of the OSI protocol reference model.

To create an implementation of a protocol layer, the definition of appropriate Java address, header, and layer classes are required. This class triple provides the basic definitions needed to insert this layer into a protocol stack (a stack of layers). By definition, the layer class implements the layer specific protocol processing rules. The address, in turn, is used to name instances of the layers. Finally, the header is used to communicate a representation of the protocol header in a packet.

Packets traverse through a protocol stack by mapping the appropriate source or destination addresses for a given layer to an object (layer) instance with that address. The packet is then handed to that layer instance for subsequent processing. Protocol stacks on a given host will have layer instances corresponding to the layer addresses currently in use at that node. This will include network interface addresses, the corresponding link layer addresses for these interfaces, and transport layer addresses in use by running applications.

Packets which arrive carrying non-local addresses, such as packets which are destined for

Lindell
the forwarding engine, cannot be mapped to an appropriately named layer instance. To handle this case, each layer implementation must delegate a layer instance as the "default" instance to handle all packet requests for unknown address values.

Layered above the protocol stacks (stacks of layers) are APIs that will be used by applications. VNET currently implements socket interfaces to the protocol implementations that mimic the Java JDK socket interfaces DatagramSocket, ServerSocket, and Socket. In the Java JDK, DatagramSocket is used as the API to UDP/IP whereas ServerSocket and Socket are used as the server and client interfaces to TCP/IP connections. In addition, VNET provides a network management interface to create, modify, and query the network interface and routing tables used by the VNET protocols.

2. Protocol Application Programming Interfaces (APIs)

2.1. User Datagram Protocol

The Java JDK provides a sockets based interface to UDP/IP. There is a similar interface to the unreliable datagram protocol in VNET. Since this interface mimics the API used in the Java JDK, we will merely highlight the subtle differences rather than providing extensive documentation.

- The corresponding name of the Java JDK DatagramSocket class in VNET is SocketDatagramV. The Java JDK DatagramPacket class is named DatagramV in VNET.

- The Java JDK represents IP addresses and ports using the types InetAddress and int, respectively. VNET uses AddressVT for ports and AddressVN for network layer addresses.

VNET datagram sockets provide two features not found in the Java JDK implementation: a zero copy interface and a packets with "router alert".

Alternative send and receive methods provide the ability to send a receive a NetBuffer object and avoid the data copy which occurs with the use of the standard DatagramPacket object. When performing a send, the application passes down a NetBuffer to the protocol stack. This buffer should contain empty space at the top of the buffer which can be used by VNET to insert the appropriate protocol headers. Applications must assume that the transmission of the packet will be asynchronous with respect to the send method invocation. This implies that no further references should be made to the NetBuffer once it has
been passed to VNET.

Applications can send datagrams using a DatagramSocketVS if they want hop-by-hop interception of the datagram along a path. It is similar to the use of the IP router alert option or the RSVP proto 46 intercept mechanisms. The port space of the DatagramSocketVS is distinct from the DatagramSocketV sockets. Therefore, an application needs an open DatagramSocketVS socket on the appropriate port at each router along the path. Datagrams are addressed to a final destination but do never travel more that one hop towards that destination. If there is no open socket at the intermediate router, the datagram is dropped rather than forwarded.

2.2. Transmission Control Protocol

The Java JDK also provides a sockets based interface to TCP/IP. VNET does not currently have an implementation of TCP. Instead, VNET provides an interface to an RDP implementation written in Java. Again, the interface mimics the Java JDK with the following differences highlighted.

- The corresponding name of the Java JDK ServerSocket class in VNET is ServerSocketV. The Java JDK Socket class is named SocketV.

- The Java JDK represents IP addresses and ports using the types InetAddress and int, respectively. VNET uses AddressVT for ports and AddressVN for network layer addresses.

2.3. The Network Management Interface

The network management interface provides the ability to add, delete, change, or query elements of the network interface and routing tables. Applications can also register for upcalls from the NMI informing the application that something in these tables has been changed by another application.

Need to say more...

3. Layer 4 Protocol Implementations
This section describes the Layer 4 protocol implementations that are available in the current prototype. Each of these layer is a subclass of the LayerL4 abstract base class.

3.1. LayerVT

This layer 4 protocol implementation provides a datagram service analogous to UDP. This layer defines a port space using an integral value. Only one layer instance can be bound to a particular port at any time. Incoming datagrams are queued internally for deliver to applications at a later time. Packets are dropped if the queues fill to capacity.

Packets for this protocol layer carry a header, based on the HeaderVT object. The source and destination fields correspond to port numbers in this layer.

\[
\text{HeaderVT Object: Header Type = 105 (Hex 0x0069), Address Types = 5}
\]

\[
\text{Header size : 14 bytes}
\]

\[
\begin{array}{cccc}
| \text{Header Type} | \text{Src Address Type} | \text{Src Address} | \\
| \text{Dst Address Type} | \text{Dst Address (MSW)} | \text{Dst Address (LSW)} |
\end{array}
\]

3.2. LayerVTS

This layer is very similar to LayerVT, providing equivalent functionality for the hop-by-hop datagram service. The port space defined in this layer is distinct from the port space used by LayerVT. Thus, it is possible to have a layer bound to the same port number in each of these spaces simultaneously.

Packets for this protocol layer carry a header, based on the HeaderVTS object. The source and destination fields correspond to port numbers in this layer.
3.3. **LayerRDP**

Details need to be filled.
HeaderRDP Object: Header Type = 106 (Hex 0x006a), Address Types = 6
(EAK ? (4 * no of EAKS):0)

| +-------------+-------------+-------------+-------------+
| Header Type  | Flags       | Hlen        |
| +-------------+-------------+-------------+-------------+
| Src Port     | Dst Port    |             |
| +-------------+-------------+-------------+-------------+
| Payload Len  | Seq No(MSW) |             |
| +-------------+-------------+-------------+-------------+
| Seq No(LSW)  | ACK No(MSW) |             |
| +-------------+-------------+-------------+-------------+
| ACK No(MSW)  | Checksum    |             |

Where Flags is:

| +---+---+---+---+---+---+---+---+ |
| S  | A  | E  | R  | N  | W  | V  | V  |
| Y  | C  | A  | S  | U  | I  | E  | E  |
| N  | K  | K  | T  | L  | N  | R  | R  |

If SYN is set; append the following options Header:

| +-------------+-------------+-------------+-------------+
| MaxOut       | MSS          |
| +-------------+-------------+-------------+-------------+
| 0000 0000    | 0000 000S    |

S = SDM.

If WIN is set; append the following ACC Header:

| +-------------+-------------+-------------+-------------+
| ACC Flag     | 0000 0000   | Winsize     |
| +-------------+-------------+-------------+-------------+
| +-- R T T T --+

| +-------------+-------------+-------------+-------------+
| Seq No(LSW)  | ACK No(MSW) |
| +-------------+-------------+-------------+-------------+
| ACK No(MSW)  | Checksum |

where ACC Flag is

| +-------------+-------------+-------------+-------------+
| Version      | 0 0 0 0     |
| +-------------+-------------+-------------+-------------+
If Version > 2 append the following to the ACC Header:

```
+---+---+---+---+---+---+---+---+
| ACC Ack No |
+---+---+---+---+---+---+---+---+
```

if EAK is set; append the following Extended ACK Seq. Nos till end of header.

```
+---+---+---+---+---+---+---+---+
| EAK Seq. No. |
+---+---+---+---+---+---+---+---+
```

4. Layer 3 Protocol Implementations

This section describes the Layer 3 protocol implementations that are available in the current prototype. Each of these layers is a subclass of the LayerL3 abstract base class.

4.1. LayerVN

The LayerVN implementation provides a network layer similar to IP with fewer features. Packets are forwarded end-to-end using only point-to-point interfaces. No packet fragmentation is supported. Packets carry a time-to-live (TTL) field to prevent excessive resource consumption in the presence of routing loops.

Packets for this protocol layer carry a header, based on the HeaderVN object.

```
HeaderVN Object: Header Type = 103(Hex 0x0067), Address Types = 3
Header size : 20
```

```
+-------------+-------------+-------------+-------------+
| Header Type | Src Address Type |
+-------------+-------------+-------------+-------------+
| Src Address |
+-------------+-------------+-------------+-------------+
| Dst Address Type | Dst Address (MSW) |
+-------------+-------------+-------------+-------------+
| Dst Address (LSW) | TTL |
+-------------+-------------+-------------+-------------+
```
4.2. LayerVNS

The LayerVNS network layer shares interfaces and routes with the LayerVN layer. The main difference between these two layers is that LayerVNS has no forwarding engine. Packets are sent only one hop. When a packet arrives, it is either delivered to the corresponding layer 4 protocol stack or it is dropped. Because packets are never forwarded, a TTL field is unnecessary for this protocol.

Packets for this protocol layer carry a header, based on the HeaderVNS object.

HeaderVNS Object: Header Type = 102 (Hex 0x0066), Address Types = 2
Header size : 14

+-----------------------------------+-------------------+-------------------+
<table>
<thead>
<tr>
<th>Header Type</th>
<th>Src Address Type</th>
</tr>
</thead>
</table>
| +-----------------------------------+-------------------+
| Src Address                       | Dst Address Type   |
| +-----------------------------------+-------------------+
| Dst Address (MSW)                 | Dst Address (LSW) |
| +-----------------------------------+-------------------+

5. Layer 2 Protocol Implementations

This section describes the Layer 2 protocol implementations that are available in the current prototype. Each of these layer is a subclass of the LayerL2 abstract base class.

5.1. LayerUI

LayerUI implements a Layer 2 protocol in a VNET based protocol stack. Normally, a Layer 2 protocol provides direct access to particular transmission media technologies. LayerUI, in contrast, transports packets using UDP/IP. By using IP, this layer makes it easy to experiment with VNET protocol stacks using hosts in the Internet. Packets are framed using both a UDP/IP header as well as a HeaderUI header. The packet size at this layer is limited to the IP message size limit of 64k less the header overhead introduced by LayerUI. Addressing at this layer uses AddressUI objects. AddressUI objects contain an IP address and UDP port number used in the source and destination fields of UDP data-grams.
A LayerUI packet has the following format:

```
+-------------+-------------+-------------+-------------+
|              |              |              |              |
|              |              |              |              |
| IP Header    |              |              |              |
+-------------+-------------+-------------+-------------+
+-------------+-------------+-------------+-------------+
|              |              |              |              |
|              |              |              |              |
| UDP Header   |              |              |              |
+-------------+-------------+-------------+-------------+
+-------------+-------------+-------------+-------------+
|              |              |              |              |
|              |              |              |              |
| HeaderUI Header |        |              |              |
+-------------+-------------+-------------+-------------+
+-------------+-------------+-------------+-------------+
|              |              |              |              |
|              |              |              |              |
|              |              |              |              |
+-------------+-------------+-------------+-------------+
+-------------+-------------+-------------+-------------+
|              |              |              |              |
|              |              |              |              |
|              |              |              |              |
+-------------+-------------+-------------+-------------+
|              |              |              |              |
|              |              |              |              |
| Data         |              |              |              |
+-------------+-------------+-------------+-------------+
+-------------+-------------+-------------+-------------+
```

Where the HeaderUI Object Format is:

- **HeaderUI Object**: Header Type = 100 (Hex 0x0064), Address Types = 0
- **Header size**: 18

```
+-------------+-------------+-------------+-------------+
| Header Type | Src Address Type |              |              |
+-------------+-------------+-------------+-------------+
| Src Address |              |              |              |
+-------------+-------------+-------------+-------------+
+-------------+-------------+-------------+-------------+
| Dst Address Type | Src Port   |              |              |
+-------------+-------------+-------------+-------------+
| Dst Address  |              |              |              |
+-------------+-------------+-------------+-------------+
|              | Dst Port    |              |              |
+-------------+-------------+-------------+-------------+
```

An instance of a LayerUI object is created by supplying a source AddressUI for that instance. The LayerUI implementation opens a socket on the given source IP address and UDP port number. When packets are passed down from Layer 3, they are encapsulated with a HeaderUI header and sent on this socket to the appropriate destination. A listening thread receives datagrams from this socket, decapsulates the packet, and sends it to the
appropriate Layer 3 protocol for subsequent processing.

5.2. LayerAnep

LayerAnep implements a Layer 2 protocol in a VNET based protocol stack. It provides packet transport using Anep [ref]. Since there are no current implementations of Anep directly on link layer technologies, this layer emulates this functionality using UDP/IP. Packets are framed using both a UDP/IP header and an Anep header. The packet size at this layer is limited to the IP message size limit of 64k less the header overhead introduced by LayerAnep. Addressing at this layer uses AddressAnep objects. AddressAnep objects contain an IP address, UDP port number, and type id used in the destination fields of Anep packets.

A LayerAnep packet has the following format:

```
+-------------+-------------+-------------+-------------+
|              |              |              |              |
| IP Header    |              |              |              |
+-------------+-------------+-------------+-------------+
|              | UDP Header  |              |              |
+-------------+-------------+-------------+-------------+
|              | Anep Header |              |              |
+-------------+-------------+-------------+-------------+
|              |              | Data        |              |
+-------------+-------------+-------------+-------------+
```

An instance of a LayerAnep object is created by supplying a source AddressAnep for that instance. The LayerAnep implementation opens a socket on a given source IP address, UDP port number. When packets are passed down from Layer 3, they are encapsulated with a HeaderAnep header and sent on this socket to the appropriate destination. A listening thread receives datagrams from this socket, decapsulates the packet, and sends it to the appropriate Layer 3 protocol for subsequent processing.

The LayerAnep class has been designed to operate with both Java JDK UDP sockets and
Anetd [ref]. Packets arriving from Anetd appear on the standard input to the JVM rather than a particular UDP port number. A LayerAnep object instance can be created using port 0 to listen on standard input. All other non-zero port numbers cause LayerAnep to listen on the corresponding UDP port number.