Inter-domain Communication using Virtual Sockets

David Vrabel <david.vrabel@citrix.com

Draft C

Contents

1 Introduction

1.1 Revision History

Version	Date	Changes
Draft C	11 Jun 2013	Minor clarifications.
Draft B	10 Jun 2013	Added a section on the low-level shared ring transport.
		Added a section on using v4v as the low-level transport.
Draft A	28 May 2013	Initial draft.

1.2 Purpose

In the Windsor architecture for XenServer, dom0 is disaggregated into several *service domains*. Examples of service domains include network and storage driver domains, and qemu (stub) domains.

To allow the toolstack to manage service domains there needs to be a communication mechanism between the toolstack running in one domain and all the service domains.

The principle focus of this new transport is control-plane traffic (low latency and low data rates) but consideration is given to future uses requiring higher data rates.

Linux 3.9 support virtual sockets which is a new type of socket (the new AF_VSOCK address family) for inter-domain communication. This was originally implemented for VMWare's VMCI transport but has hooks for other transports. This will be used to provide the interface to applications.

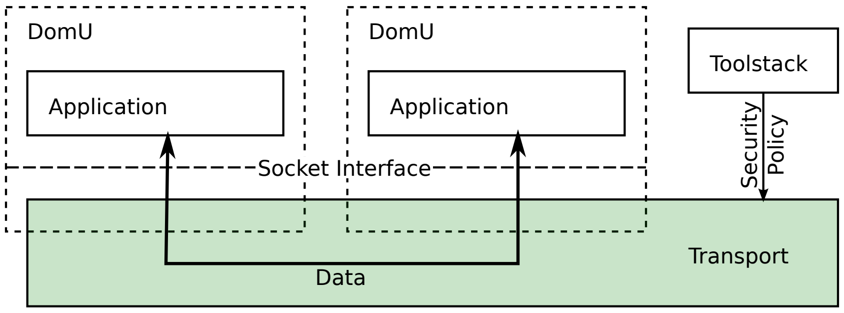

Figure 1: System Overview

1.3 System Overview

1.4 Design Map

The linux kernel requires a Xen-specific virtual socket transport and front and back drivers.

The connection manager is a new user space daemon running in the backend domain.

Toolstacks will require changes to allow them to set the policy used by the connection manager. The design of these changes is out of scope of this document.

1.5 Definitions and Acronyms

AF_VSOCK The address family for virtual sockets.

CID (Context ID) The domain ID portion of the AF_VSOCK address format.

Port The part of the AF_VSOCK address format identifying a specific service. Similar to the port number used in TCP connection.

Virtual Socket A socket using the AF_VSOCK protocol.

1.6 References

Windsor Architecture slides from XenSummit 2012

2 Design Considerations

2.1 Assumptions

- There exists a low-level peer-to-peer, datagram based transport mechanism using shared rings (as in libvchan).

2.2 Constraints

- The AF_VSOCK address format is limited to a 32-bit CID and a 32-bit port number. This is sufficient as Xen only has 16-bit domain IDs.

2.3 Risks and Volatile Areas

- The transport may be used between untrusted peers. A domain may be subject to malicious activity or denial of service attacks.

3 Architecture

3.1 Overview

Linux's virtual sockets are used as the interface to applications. Virtual sockets were introduced in Linux 3.9 and provides a hypervisor independent[1] interface to user space applications for inter-domain communication.

An internal API is provided to implement a low-level virtual socket transport. This will be implemented within a pair of front and back drivers. The use of the standard front/back driver method allows the toolstack to handle the suspend, resume and migration in a similar way to the existing drivers.

The front/back pair provides a point-to-point link between the two domains. This is used to communicate between applications on those hosts and between the frontend domain and the *connection manager* running on the backend.

The connection manager allows domUs to request direct connections to peer domains. Without the connection manager, peers have no mechanism to exchange the information ncessary for setting up the direct connections. The toolstack sets the policy in the connection manager to allow connection requests. The default policy is to deny connection requests.

[1]The API and address format is hypervisor independent but the address values are not.

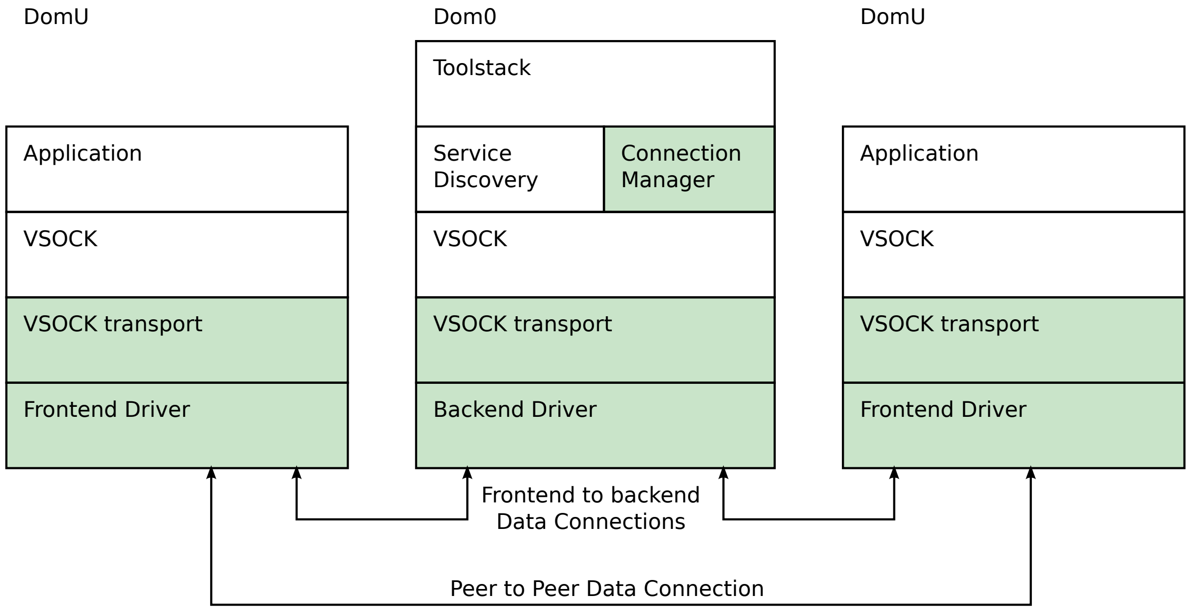

Figure 2: Architecture Overview

4 High Level Design

4.1 Virtual Sockets

The AF_VSOCK socket address family in the Linux kernel has a two part address format: a uint32_t *context ID* (*CID*) identifying the domain and a uint32_t port for the specific service in that domain.

The CID shall be the domain ID and some CIDs have a specific meaning.

CID	Purpose
0x7FF0 (DOMID_SELF)	The local domain.
0x7FF1	The backend domain (where the connection manager is).

Some port numbers are reserved.

Port	Purpose
0	Reserved
1	Connection Manager
2–1023	Reserved for well-known services (such as a service discovery service).

4.2 Front / Back Drivers

Using a front or back driver to provide the virtual socket transport allows the toolstack to only make the inter-domain communication facility available to selected domains.

The "standard" xenbus connection state machine shall be used. See figures 3 and 4 on pages 6 and 10.

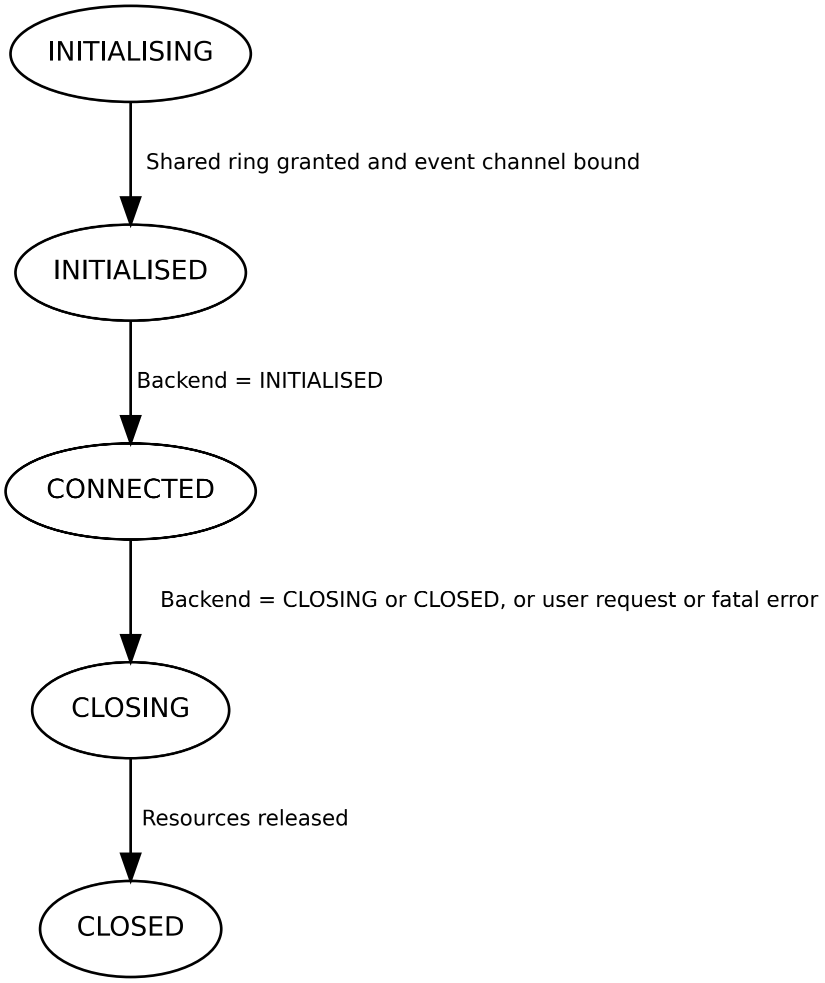
INITIALISING
Shared ring granted and event channel bound
INITIALISED
Backend = INITIALISED
CONNECTED
Backend = CLOSING or CLOSED, or user request or fatal error
CLOSING
Resources released
CLOSED

Figure 3: Frontend Connection State Machine

4.3 Connection Manager

The connection manager has two main purposes.

1. Checking that two domains are permitted to connect.
2. Providing a mechanism for two domains to exchange the grant references and event channels needed for them to setup a shared ring transport.

Domains commnicate with the connection manager over the front-back transport link. The connection manager must be in the same domain as the virtual socket backend driver.

The connection manager opens a virtual socket and listens on a well defined port (port 1).

The following messages are defined.

Message	Purpose
CONNECT_req	Request connection to another peer.
CONNECT_rsp	Response to a connection request.
CONNECT_ind	Indicate that a peer is trying to connect.
CONNECT_ack	Acknowledge a connection request.

Before forwarding a connection request to a peer, the connection manager checks that the connection is permitted. The toolstack sets these permissions.

Disconnecting transport links to an uncooperative (or dead) domain is required. Therefore there are no messages for disconnecting transport links (as these may be ignore or delayed). Instead a transport link is disconnected by tearing down the local end. The peer will notice the remote end going away and then teardown its end.

5 Low-level transport

[This exact details are yet to be determined but this section should provide a reasonably summary of the mechanisms used.]

5.1 Frontend and backend domains

As is typical for frontend and backend drivers, the frontend will grant copy-only access to two rings — one for from-front messages and one for to-front messages. Each ring shall have an event channel for notifying when requests and responses are placed on the ring.

5.2 Peer domains

The initiator grants copy-only access to a from-initiator (transmit) ring and provides an event channel for notifications for this ring. This information is included in the CONNECT_req and CONNECT_ind messages.

The responder grants copy-only access to a from-responder (transmit) ring and provides an event channel for notifications for this ring. The information is included in the CONNECT_ack and CONNECT_rsp messages.

After the initial connection, the two domains operate as identical peers. Disconnection is signalled by a domain ungranting its transmit ring, notifying the peer via the associated event channel. The event channel is then unbound.

6 Appendix

6.1 V4V

An alternative low-level transport (V4V) has been proposed. The hypervisor copies messages from the source domain into a destination ring provided by the destination domain.

Because peers are untrusted, in order to prevent them from being able to denial-of-service the processing of messages from other peers, each receiver must have a per-peer receive ring. A listening service does not know in advance which peers may connect so it cannot create these rings in advance.

The connection manager service running in a trusted domain (as in the shared ring transport described above) may be used. The CONNECT_ind message is used to trigger the creation of receive ring for that specific sender.

A peer must be able to find the connection manager service both at start of day and if the connection manager service is restarted in a new domain. This can be done in two possible ways:

1. Watch a Xenstore key which contains the connection manager service domain ID.
2. Use a frontend/backend driver pair.

6.1.1 Advantages

- Does not use grant table resource. If shared rings are used then a busy guest with hundreds of peers will require more grant table entries than the current default.

6.1.2 Disadvantages

- Any changes or extentions to the protocol or ring format would require a hypervisor change. This is more difficult than making changes to guests.
- The connection-less, "shared-bus" model of v4v is unsuitable for untrusted peers. This requires layering a connection model on top and much of the simplicity of the v4v ABI is lost.
- The mechanism for handling full destination rings will not scale up on busy domains. The event channel only indicates that some ring may have space — it does not identify which ring has space.

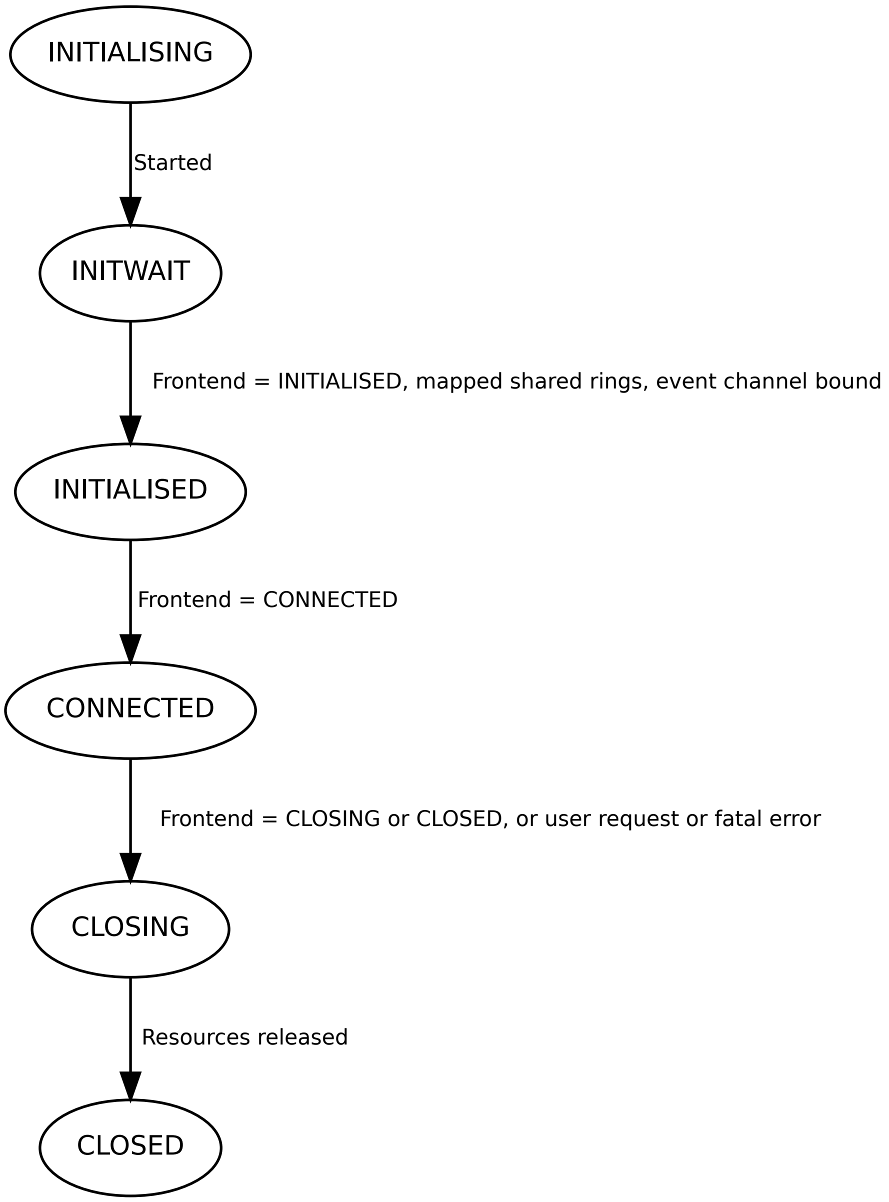
INITIALISING
Started
INITWAIT
Frontend = INITIALISED, mapped shared rings, event channel bound
INITIALISED
Frontend = CONNECTED
CONNECTED
Frontend = CLOSING or CLOSED, or user request or fatal error
CLOSING
Resources released
CLOSED

Figure 4: Backend Connection State Machine

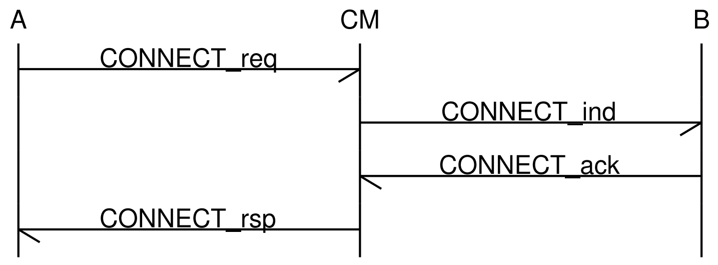

Figure 5: Connect Message Sequence Chart