

Native AF_VSOCK API for userspace device emulation

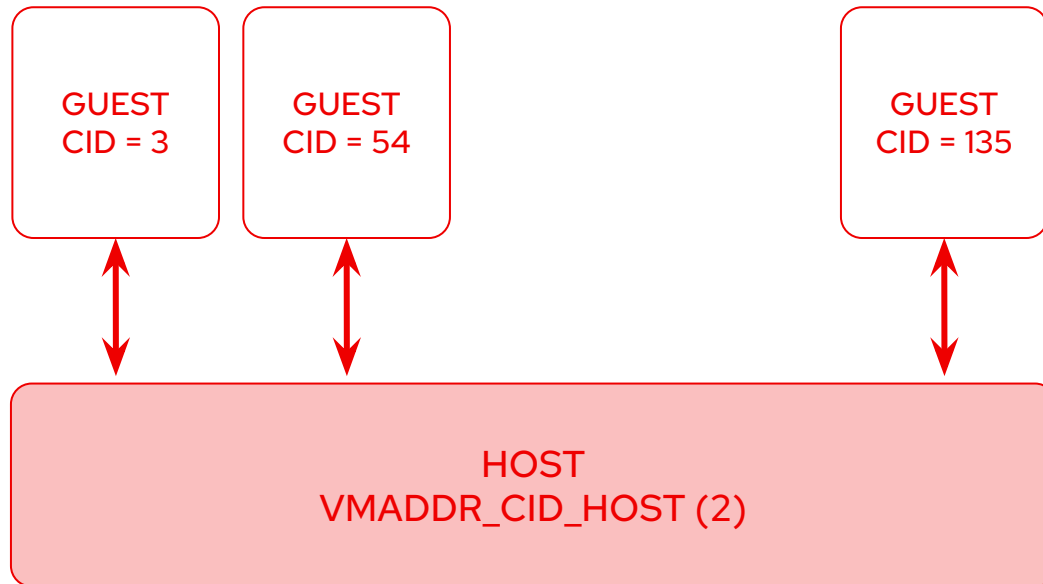
LPC 2026

Stefano Garzarella

[<sgarzare@redhat.com>](mailto:sgarzare@redhat.com)

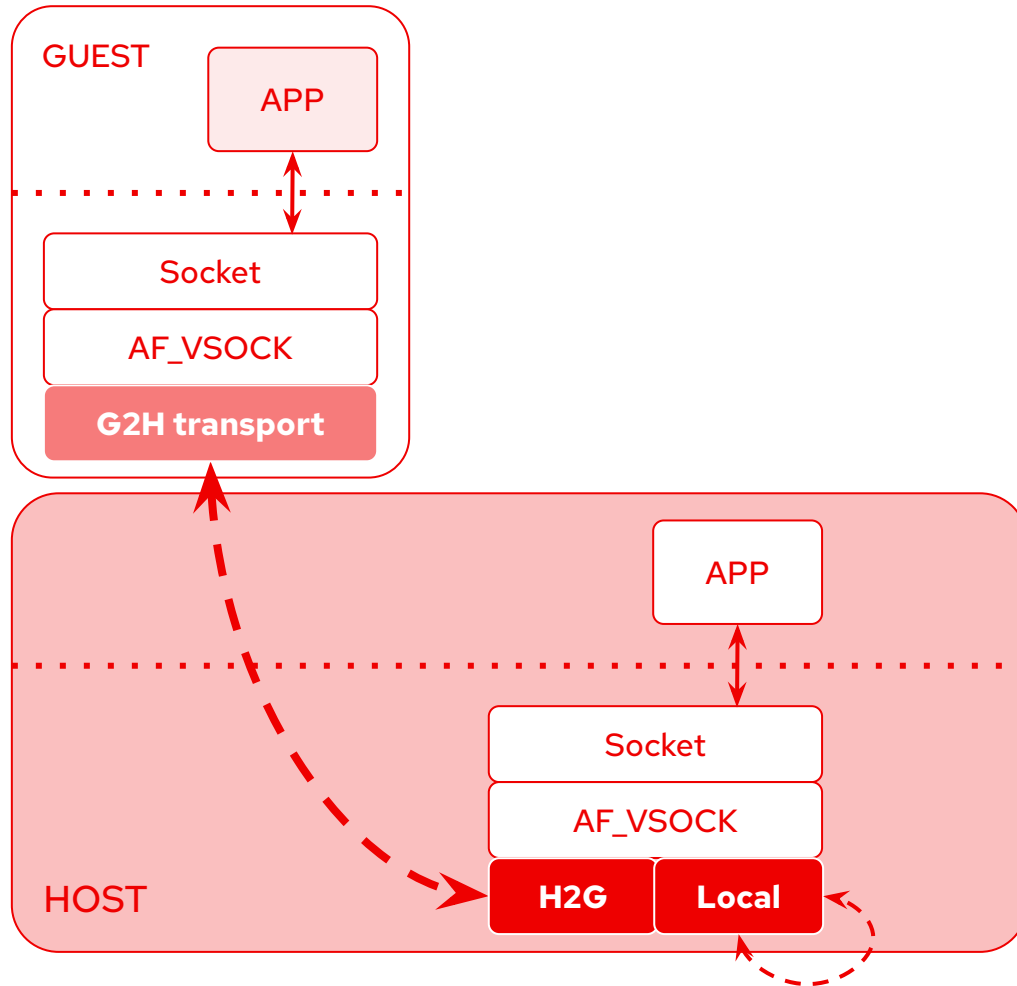
AF_VSOCK

VM Socket



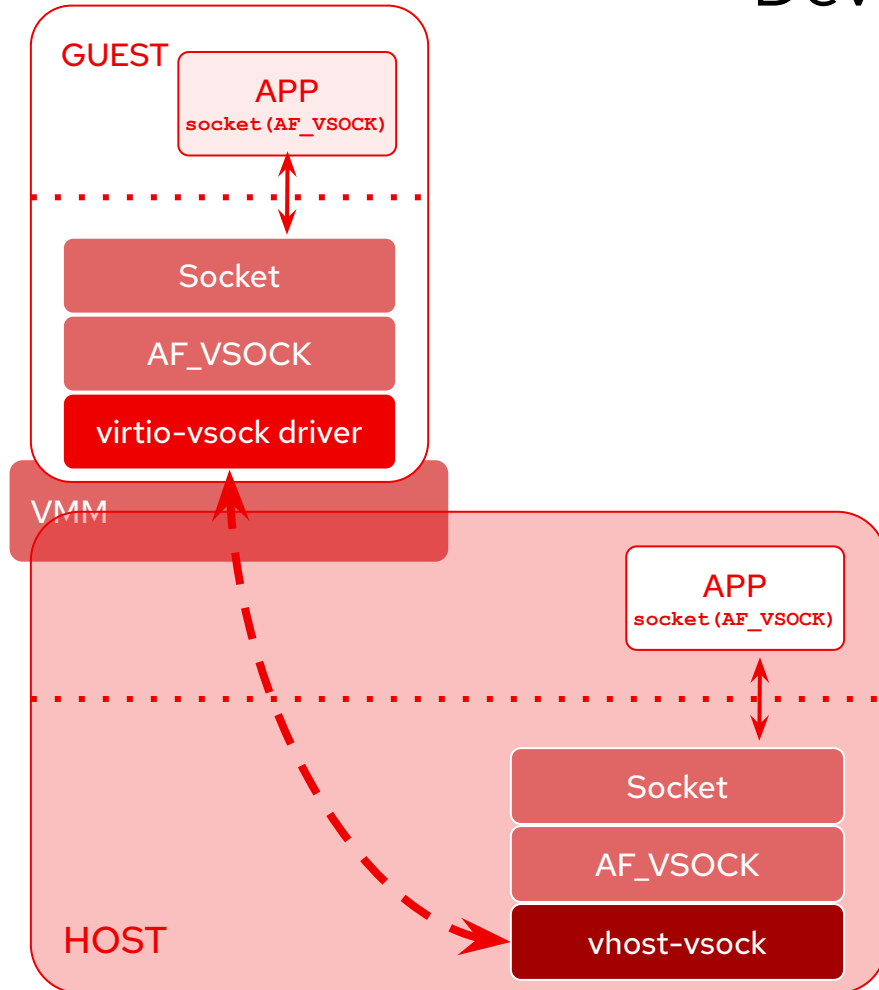
- POSIX Socket API
- Address: **CID** (Context IDentifier) + **port**
 - Peer (guest or host) identifier
 - Well defined CIDs
 - VMADDR_CID_HYPERVISOR (0)
 - VMADDR_CID_LOCAL (1)
 - VMADDR_CID_HOST (2)
 - VMADDR_CID_ANY (0xFFFFFFFF)
- Minimal configuration
 - Host
 - assigns a CID to each guest
 - chosen by user or management tool
 - Guest
 - no configuration needed

VSOCK transports



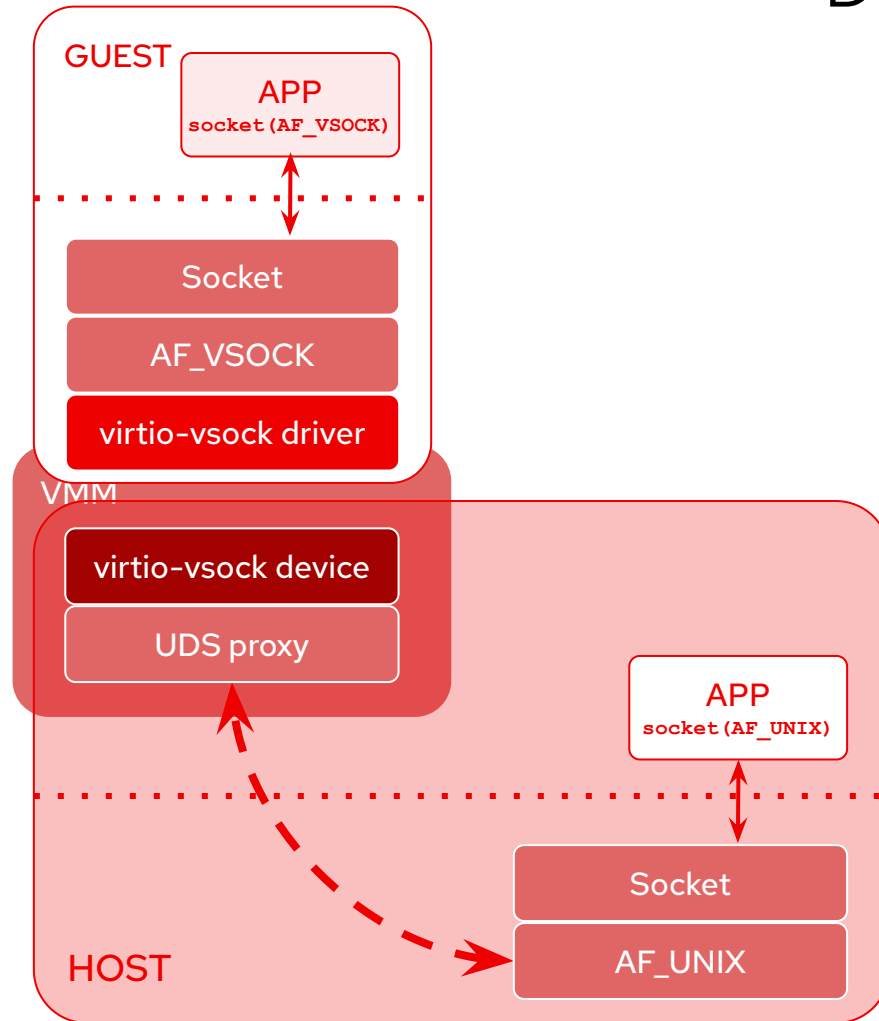
- Communication channel between guest and host
 - hypervisor-dependent
- guest → host (**G2H**) transports
 - virtio
 - vmci
 - hyperv
- host → guest (**H2G**) transports
 - vhost (virtio-vsock device)
 - vmci
- **local** transport
 - vsock_loopback

Device in the host kernel



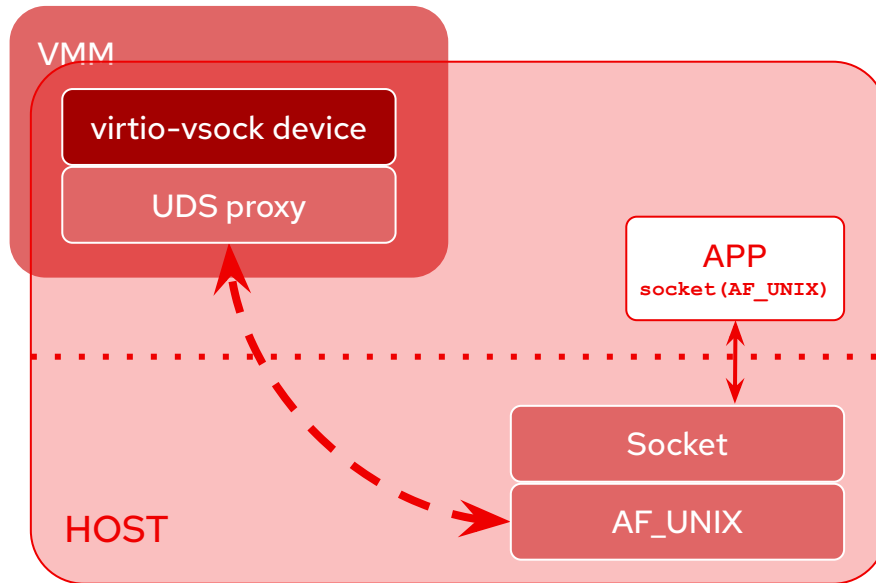
- virtio-vsock device emulated in a kernel module
 - vhost-vsock
 - Available since Linux 4.8 [2016/10/02]
- Host and guest applications use the same address family
 - `socket(AF_VSOCK, ...)`
 - `struct sockaddr_vm {cid, port}`
- Best performance

Device in userspace



- virtio-vsock device emulated in the VMM
 - Firecracker, Cloud Hypervisor, libkrnl
 - vhost-user-vsock (QEMU)
- Reduced attack surface
- Portability
- **Hybrid vsock**
 - VMMs expose Unix domain sockets
 - Application in the host must use AF_UNIX instead of AF_VSOCK

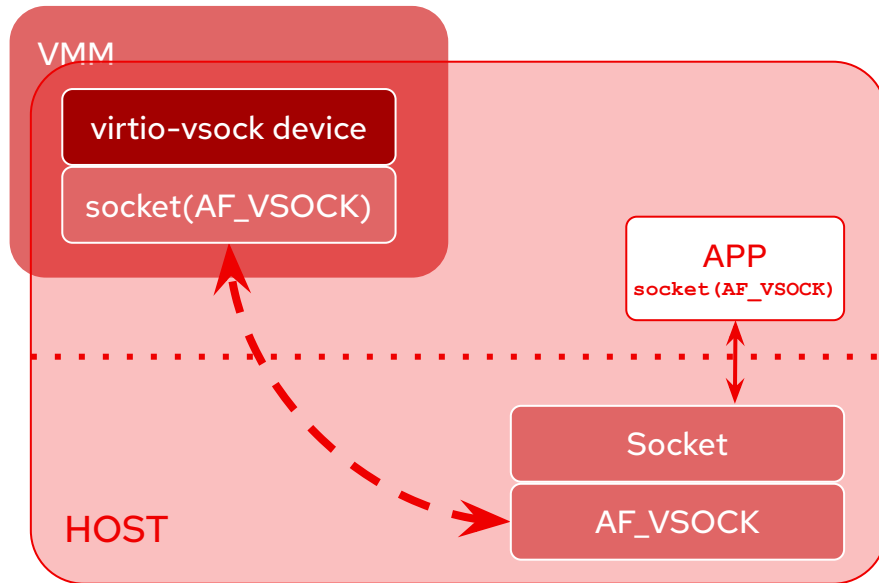
Hybrid vsock



- Host apps must implement an ad-hoc protocol
 - VMM's `uds_path` parameter (e.g. `/tmp/v.sock`)
 - Host application connecting
 - `connect(fd, "/tmp/v.sock")`
 - `send(fd, "CONNECT <port>")`
 - Host application listening
 - `bind(fd, "/tmp/v.sock_<port>")`
 - `listen(fd)`
- Guest CID invisible to host kernel
- No integration with vsock netns recently merged
- **Two different APIs for the same thing depending on how VMM emulates the vsock device**

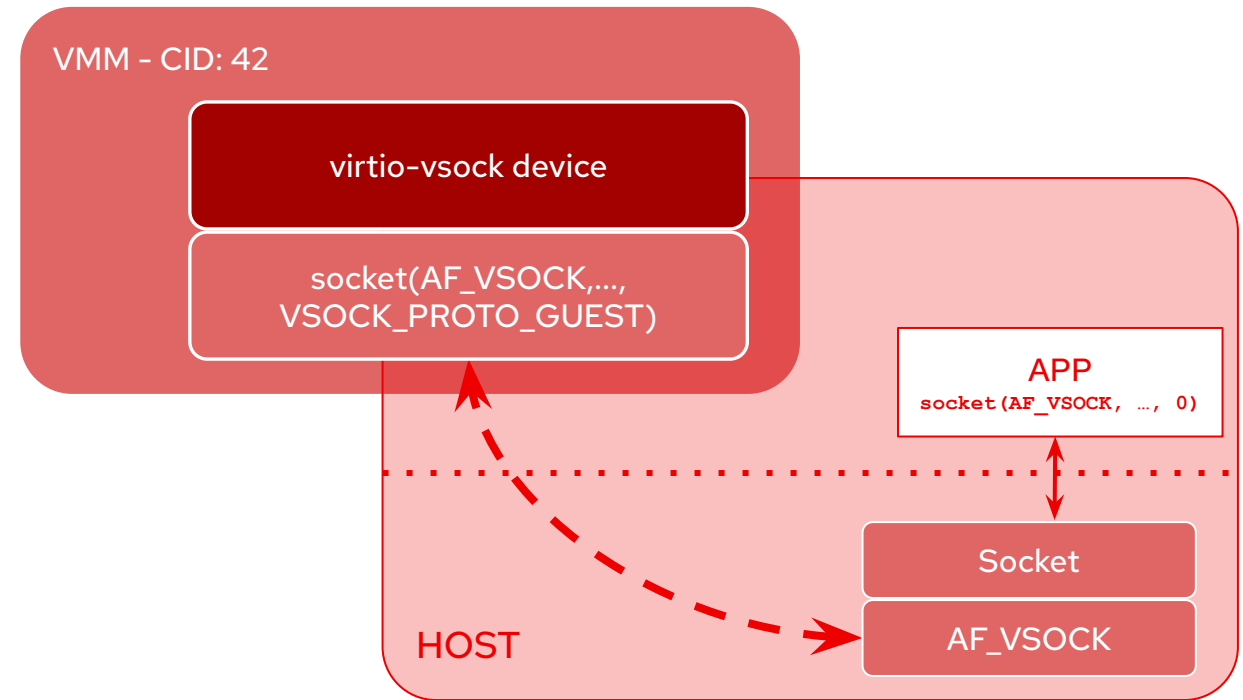
Our goal

- **Single API** for host application
 - regardless of kernel or userspace device implementation
- **Keep changes minimal** in VMMs emulating vsock device in userspace
 - swap AF_UNIX for AF_VSOCK
 - one socket per connection
 - userspace device emulation



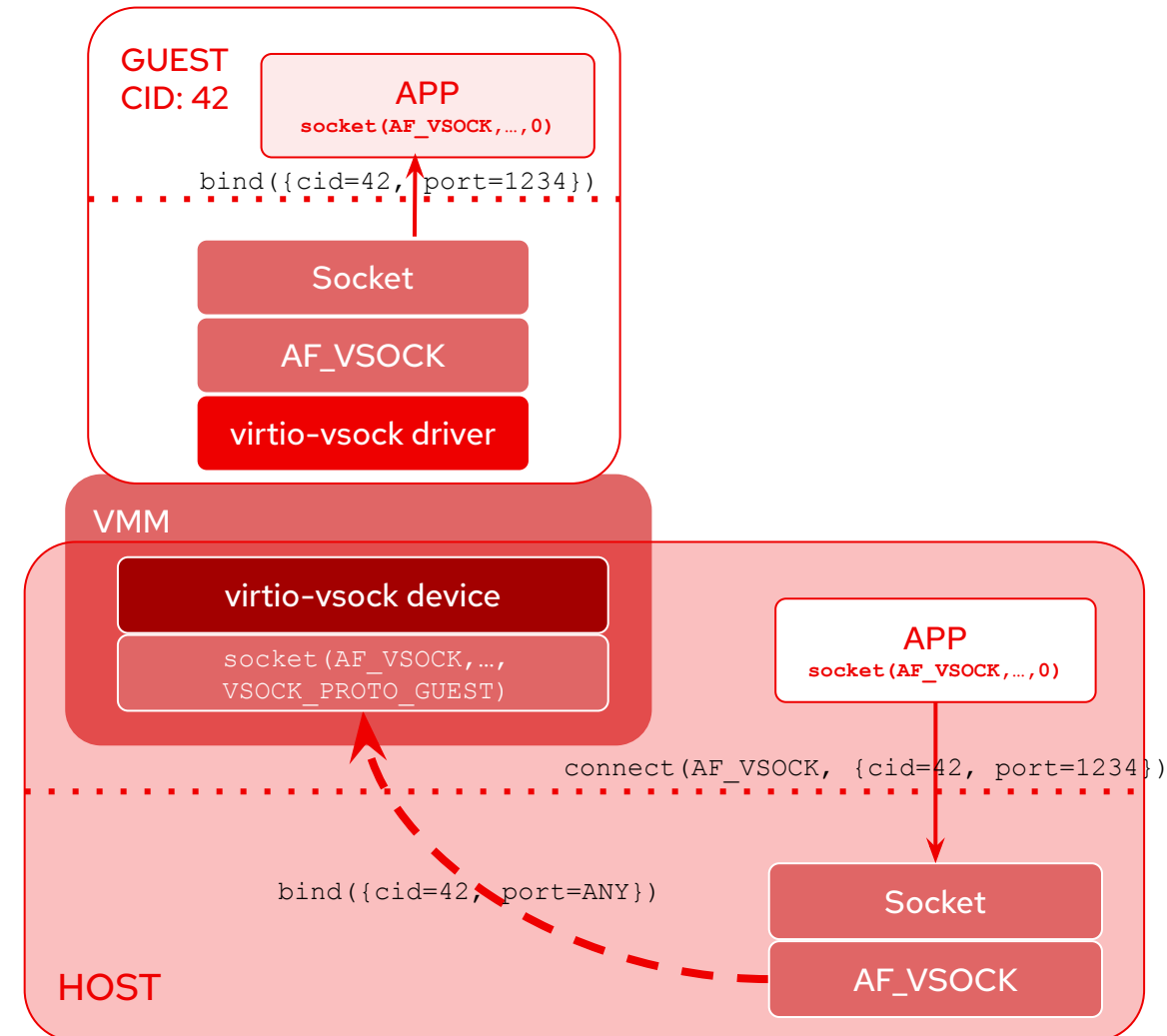
New AF_VSOCK protocol type

- `socket(AF_VSOCK, SOCK_STREAM, VSOCK_PROTO_GUEST)`
- Inversion:
 - does not connect **to** a guest, it **becomes** one
 - represents the guest from the host's AF_VSOCK stack perspective
 - similar to sockets handled by H2G transports
 - CID assigned and visible to the host kernel
- `bind({cid=42, port=VMADDR_PORT_ANY})`
 - verifies CID is available and reserves it
 - requires CAP_NET_ADMIN
 - alternative
 - `ioctl()` to reserve the CID



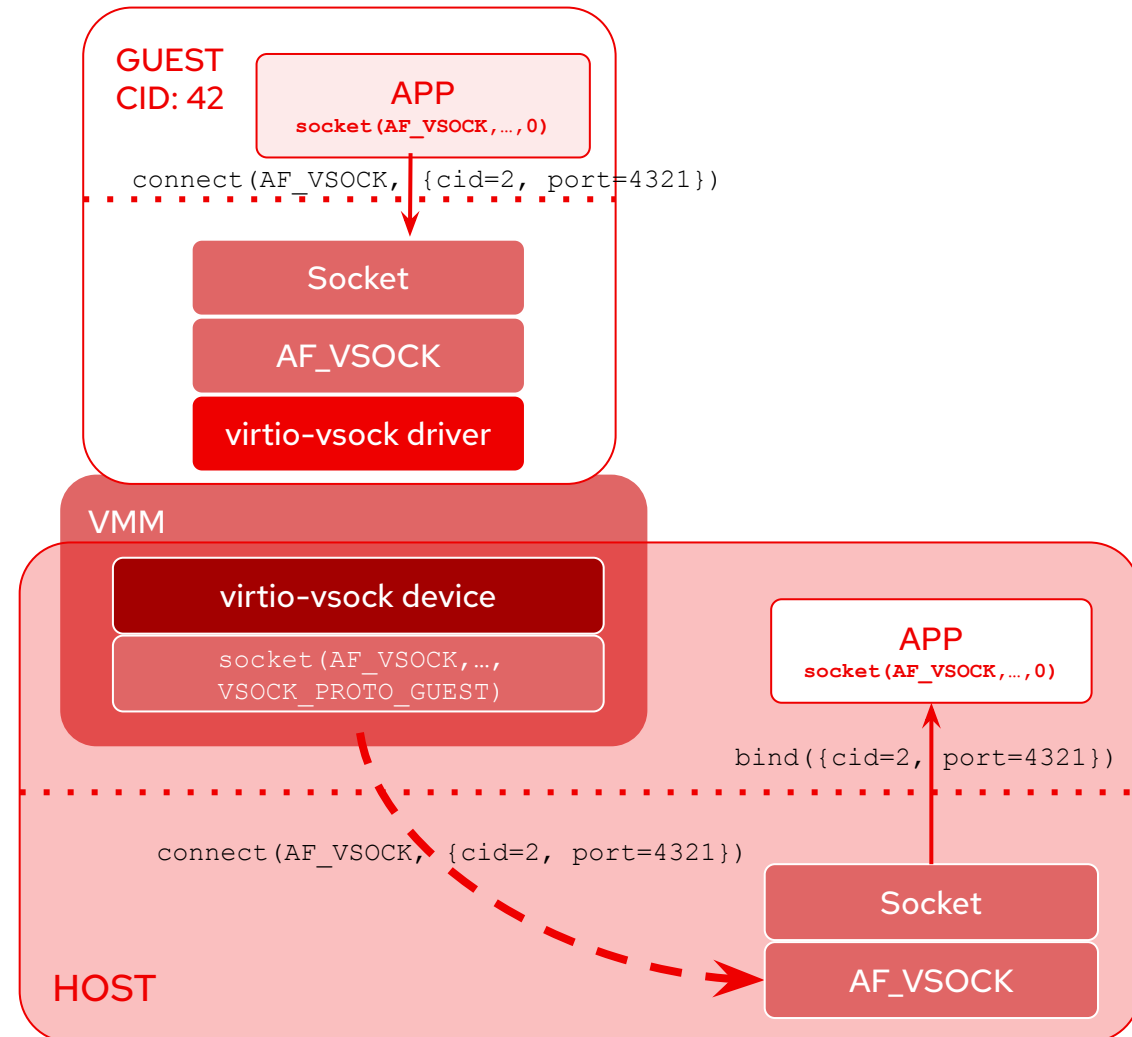
Host -> Guest flow

- VMM initialization
 - `socket(AF_VSOCK, SOCK_STREAM, VSOCK_PROTO_GUEST)`
 - `bind({cid=42, port=VMADDR_PORT_ANY})`
 - `listen()`
- Host app (vsock service)
 - `connect(AF_VSOCK, {cid=42, port=1234})`
 - same as kernel vhost-vsock
 - compared to hybrid vsock:
 - `connect(fd, "/tmp/v.sock")`
 - `send(fd, "CONNECT 1234")`
- VMM
 - `host_socket = accept()`
 - identical to the hybrid vsock approach from this point

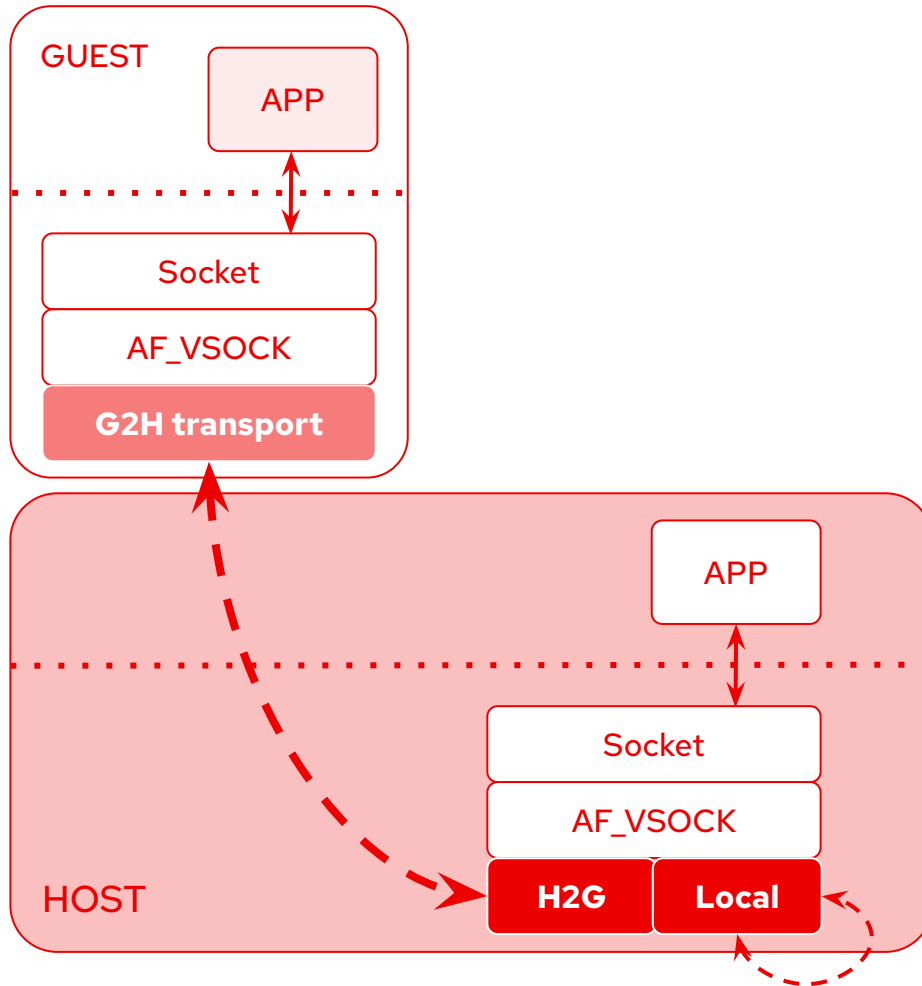


Guest -> Host flow

- Host app (vsock service)
 - `bind(AF_VSOCK, {cid=2, port=4321})`
 - same as kernel vhost-vsock
 - compared to hybrid vsock:
 - `bind(AF_UNIX, {path="/tmp/v.sock_4321"})`
- Guest connects to VMADDR_CID_HOST (CID 2), port 4321
- VMM receives via virtio-vsock emulation
 - `socket(AF_VSOCK, SOCK_STREAM, VSOCK_PROTO_GUEST)`
 - `bind({cid=42, port=src_port})`
 - `host_socket = connect({cid=2, port=4321})`
 - identical to the hybrid vsock approach from this point

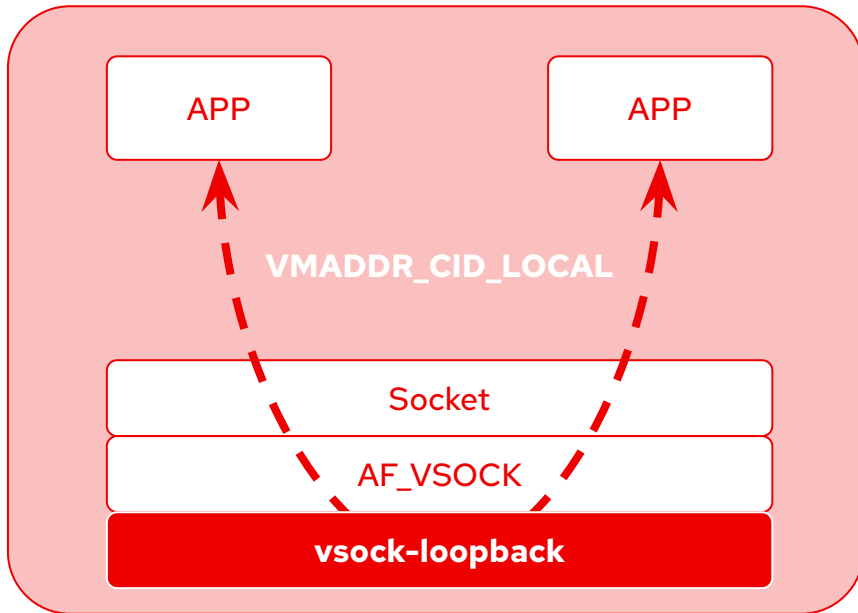


AF_VSOCK changes



- CID space in the host
 - Single registry shared with H2G transports
 - vhost-vsock and VSOCK_PROTO_GUEST both registering into one table
 - Namespace support (vsock supports netns since Linux 7.0)
 - GLOBAL mode: CIDs must be globally unique
 - LOCAL mode: CIDs are scoped, so they can be reused across namespaces
 - CID reservation/allocation
 - Proposed: `bind()` reserves available CID (like port binding)
 - Semantic: "verify available, claim it"
 - Alternative: explicit `ioctl()` on `/dev/vsock`
- Wildcard-port listener (`VMADDR_PORT_ANY`):
 - one listener per CID catches all guest ports
 - accepted socket carries the `connect()` destination port (`getsockname`)

vsock_loopback reuse



- **VSOCK_TRANSPORT_F_LOCAL**
 - handles local communication (CID=1) mainly for testing/debugging
- **VSOCK_PROTO_GUEST** communication is LOCAL
 - both endpoints on the same host kernel
- **vsock_loopback** changes
 - per-socket source CID
 - instead of VMADDR_CID_LOCAL (1)
 - **VSOCK_PROTO_GUEST** support
 - `bind()` on a guest CID
 - restricts `connect()` to CID 2 (vhost-vsock enforces the same)
 - rename to `vsock_local.ko` ?
 - with a `MODULE_ALIAS`
- Alternative: abstract it and create a new `vsock_local.ko`

Thank you!

Stefano Garzarella <sgarzare@redhat.com>

<https://stefano-garzarella.github.io/>

 [linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)

 facebook.com/redhatinc

 youtube.com/user/RedHatVideos

 twitter.com/RedHat

