Transparent Network Services via a Virtual Traffic Layer for Virtual Machines

Motivation

- HPC community more interested in using VMs
- Particularly because of
  - VM migration capabilities
    - Much more cleaner than process migration
  - Transparent addition of new services without changing the VM or the applications.
- Monitoring and controlling
  - Execution of VMs
  - Network communication of VMs

Problem being addressed

- How to provide new class of network services to applications in the VMs?
  - Without modifying either the VM or the Apps.
- Goals:
  - Monitor traffic
  - Control routing
  - Interpose and modify data and signaling

Contribution

- VTL: Virtual traffic later toolset
  - Packet capture
  - Packet inspection
  - Connection state maintenance and modification
  - And combination of the above
- Basically a fancy NAT for VMs

Service Examples

- Tor-VTL
- Subnet Tunneling
- Local Acks
- Split TCP
- Protocol transformations
- Stateful firewall
- TCP Keep-alives
- Traffic wormholing for IDS

VTL Toolset

- Library of packet monitoring and manipulation functions
- User-level code that relies on libpcap or winpcap
- Assumption: All traffic goes through a virtual network interface (“host-only” NIC) via domain 0 or equialent.

Relationship to VNET

- VNET
  - Allows nodes on disjoint IP subnets to pretend as if they are on the same subnet.
  - Relies on a set of proxying hosts that perform MAC address transformations in packets while keeping IP addresses the same.
- VTL can work either standalone or with VNET
  - VTL-VNET communication via a local channel.

VTL Example

```
RawEthernetPacket pkt;

iface_t * src_if = if_connect("src_device");
iface_t * dst_if = if_connect("dst_device");

while (if_read(src_if, &pkt)) {
    if_write(dst_if, &pkt);
}
```

Figure 1: Simple one-way VTL bridge.

```
RawEthernetPacket pkt;
unsigned long dst, new_dst;

dst = *(uint32 *)IP_DST(pkt.data);
*(uint32 *)IP_DST(pkt.data) = new_dst;

dst = GET_IP_DST(&pkt);
SET_IP_DST(&pkt, new_dst);
```

Figure 3: Example of basic packet access.

```
int create_data_pkt(vtl_model_t * model,
                    char * data,
                    int data_len) {

  RawEthernetPacket data_pkt;

  create_empty_pkt(&model,
                   &data_pkt,
                   OUTBOUND_PKT);

  memcpy(TCP_DATA(data_pkt.data),
         data, data_len);

  ip_len = GET_IP_TOTAL_LEN(data_pkt.data);
  ip_len += data_len;
  SET_IP_TOTAL_LEN(data_pkt.data, ip_len);

  compute_ip_checksum(&data_pkt);
  compute_tcp_checksum(&data_pkt);

  sync_model(&model, &data_pkt);

  pkt_len = data_pkt.get_size() + data_len;
  data_pkt.set_size(pkt_len);

  queue_pkt(&data_pkt);
  return 0;
}
```

Figure 4: Creating a data packet.

User space overhead

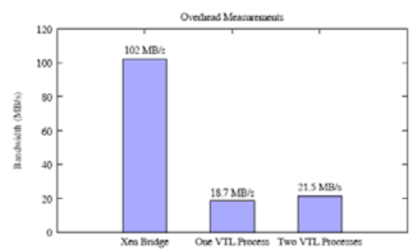


Figure 2: Performance overhead of VTL.

State Model

- Basically connection state information for individual network connections in VM
- For TCP
  - Sequence and ACK numbers
  - Timestamps
  - Receive window size?
  - IP state (?) etc
- Initialization
  - Manually
  - Supplying an example packet

Anonymous TOR Service

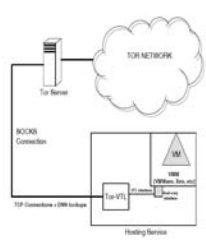


Figure 5: Network configuration of Tor-VTL.

- Use of SOCKS standard for proxies
- Four states:
  - Open, Established, close, Error
- DNS Lookups
  - SOCKS4a
- ARP interception

Subnet Tunneling

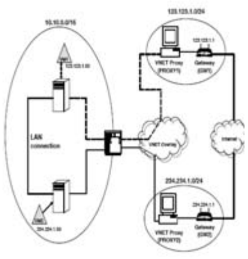
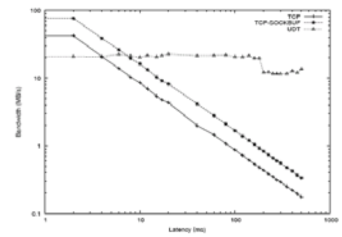


Figure 6: Subnet tunneling example.

- Optimize communication between two VM on the same LAN but different VNET sIP subnets
- Basically MAC address remapping

Enhancing Network Performance

- For high bandwidth-delay product networks
- Local ACKs
- Split TCP
- Protocol transformations



Connection persistence during VM migration

- Over wide-area, we have longer VM migration time during which VM does not run.
- Routing changes: Handled by VNET
- Timeouts: handled by VTL
- TCP keep-alive packets
- Advertising a receive window size of zero to peer
- Respond to periodic probes from peer

Coorperative wormhole scheduling and Vortex