DMM: Empower Diverse Open Transport Layer Protocol in the Cloud Networking

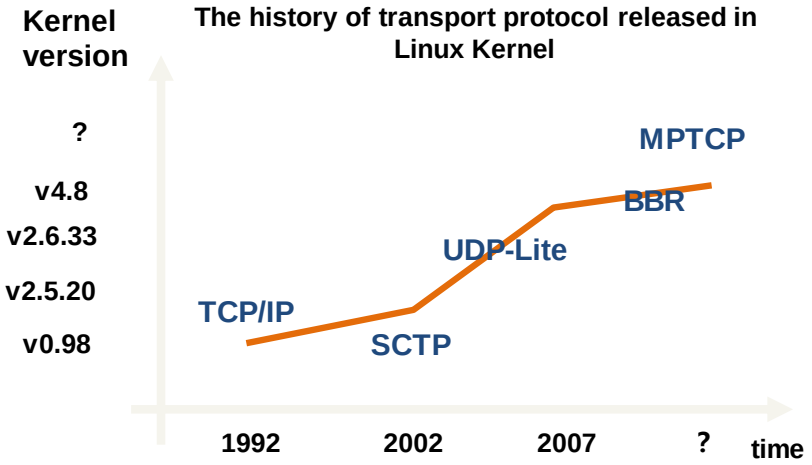
Yalei Wang

Agenda

- **Challenges on protocol stack**
- DMM introduction
- Use cases and stack integration examples
- Roadmap

Challenges on Protocol stack

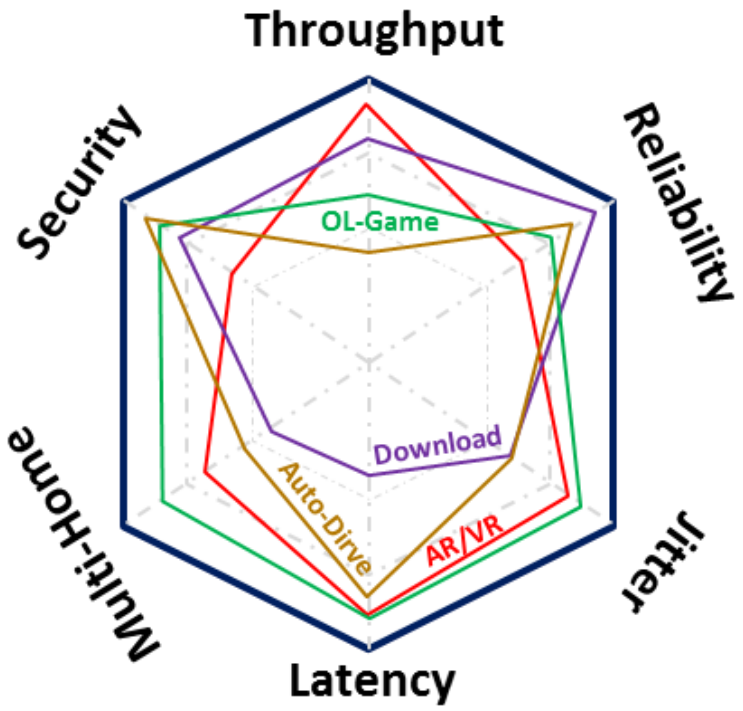
- **TCP Protocol - a performance bottleneck**
  - ✓ Loss sensitive
  - ✓ Poor in larger bandwidth-delay product network
  - ✓ No delivery latency SLA guarantee
  - ✓ Difficult to tune performance
- **Operating System Kernel Stack**
  - ✓ Low performance
  - ✓ Monolithic in design
  - ✓ Hard to customize
  - ✓ Long protocol/algorithm release cycle



- More than 25 years, <5 transport protocols are released in the Linux kernel
- It has takes 8 years after MPTCP was firstly proposed, but MPTCP is still not released in the Linux kernel

Future Transport Protocol Design

- **Ultimate performance**
  - ✓ Video – orders of magnitudes higher bandwidth
  - ✓ VR/AR – very low latency and jitter
  - ✓ IoT – orders of magnitudes more concurrent connections
- **Diversified network QoS/SLA**
  - ✓ Applications with different QoS/SLA requirements exist simultaneously on the same platform
  - ✓ Any optimization is tradeoff between factors
- **Heterogeneous network environments**
  - ✓ Cloud computing and mobile internet turn the network into an extremely complicated system
  - ✓ Network environment might change significantly due to mobility



Challenges on protocol stack - Some Answers

- **Alternative transport protocols**
  - ✓ Google's QUIC
  - ✓ IBM's FASP
- **User-space network stack**
  - ✓ Improving performance: System call, scalability, …
  - ✓ Protecting intellectual property
  - ✓ F-Stack, Seastar, MTCP, vpp-hoststack, …

Challenges

- Diversity is future: One-size-fits-all protocol is not feasible，
- User space protocol stacks need complete solution on：socket layer, fd management，epoll framework，system call, io layer, …
- Completely bypass kernel stack or kernel mode？
  - ✓ Intrusive or non-intrusive for APP
- How to apply the new protocol?
  - ✓ RTC mode or pipeline mode
- APP won’t consider service discovery, LB … but how to avoid considering using what kind of protocol stack

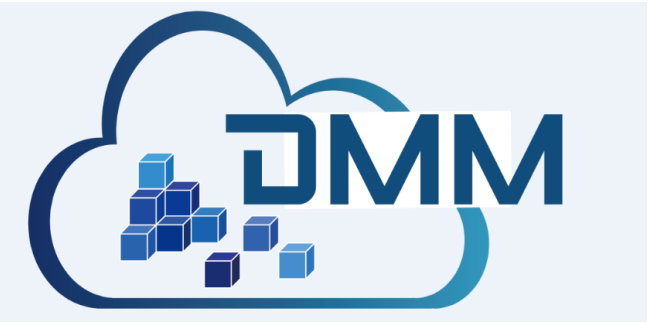
Agenda

- What protocol stack face today
- **DMM introduction**
- Use cases and stack integration example
- Roadmap

DMM: Re-design the Protocol Stack

DMM (Dual modes Multi-protocols Multi-instances) is a network stack framework which enables:

- **Dual mode:** Kernel space and User space
- **Multi-protocols:** Simple new protocol adoptions and integrations
- **Multi-instances:** Enable “Protocol Routing”



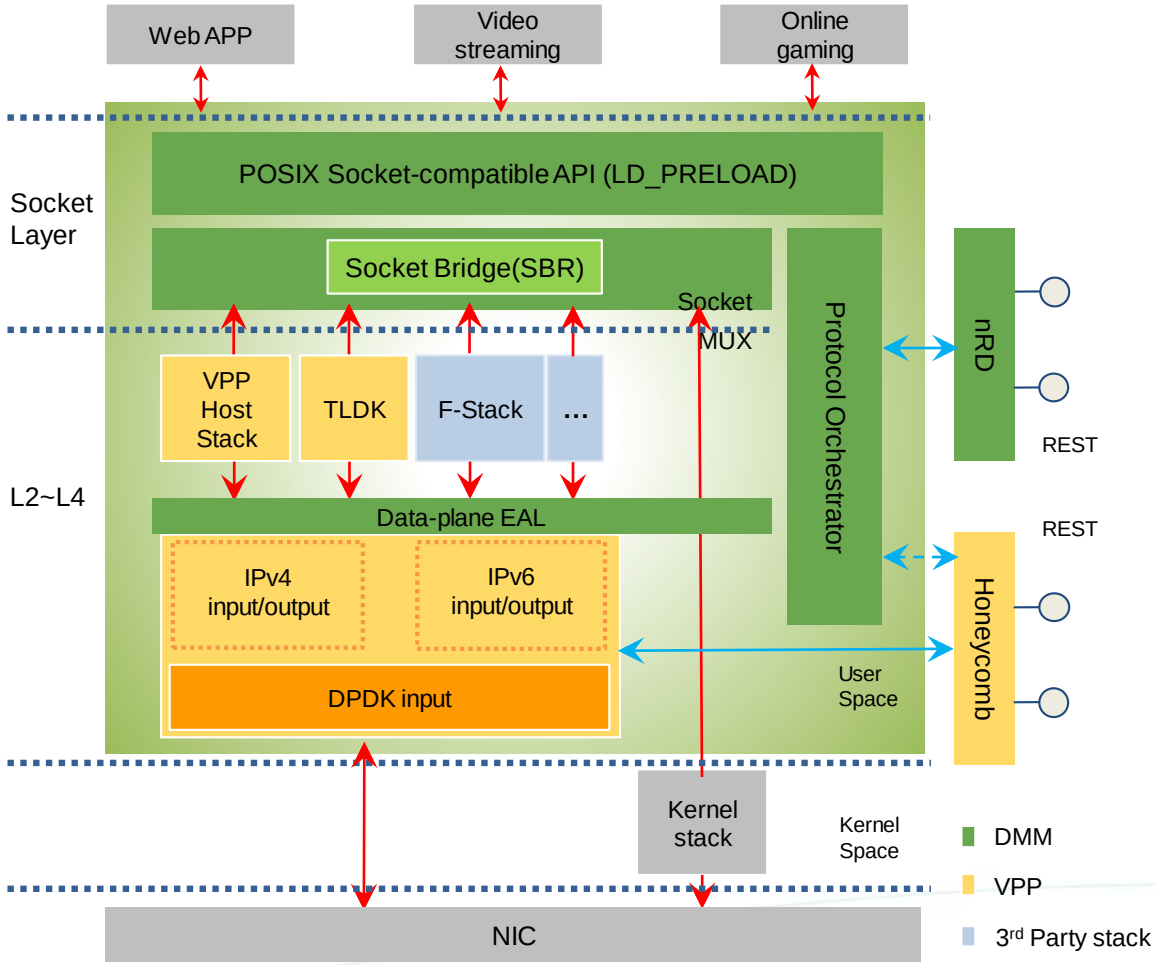
The concept of DMM was proposed by Huawei for the first time in 2015.

Enabling “Protocol Routing”: Revisiting Transport Layer Protocol Design in Internet Communications

DMM: Architecture

Key Techniques

- Distributed and Centralized nRD deployment (LRD & CRD) provide end-to-end protocol orchestration
- Stack-transparent “Protocol Routing” (Stack orchestrator)
- POSIX compatible socket APIs
- Flexible socket API redirection and mapping (SBR)
- Flexible APIs for integration of third party stacks (EAL)
- Multiple stack instances support
- Multiple I/O engines support

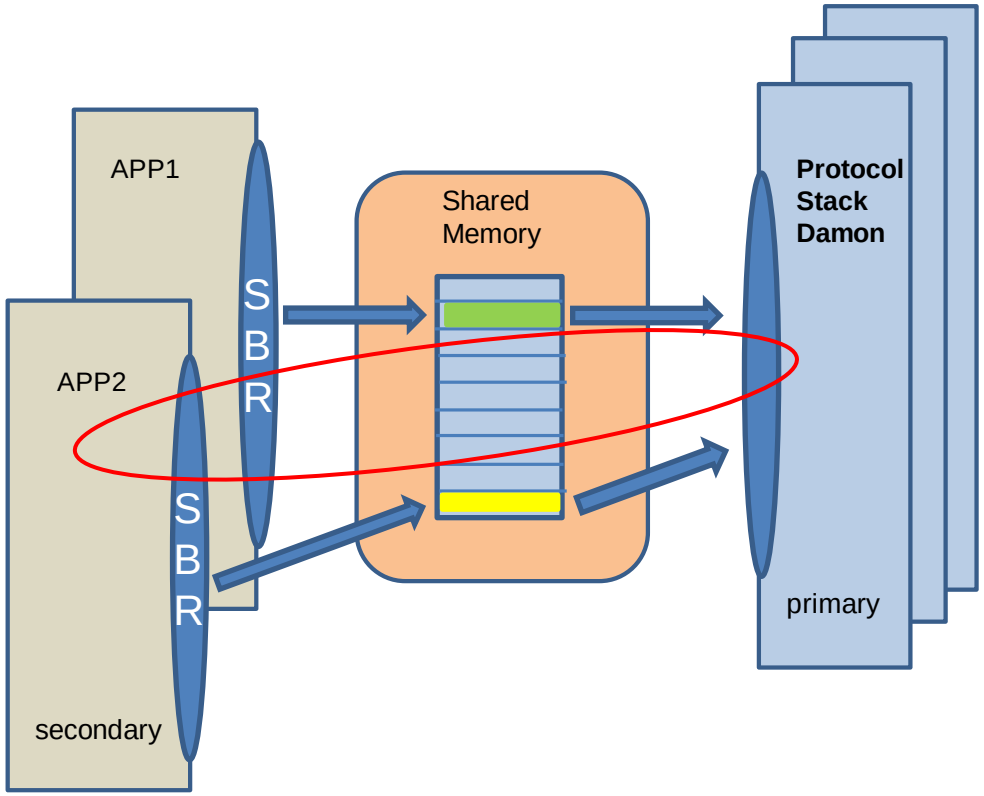


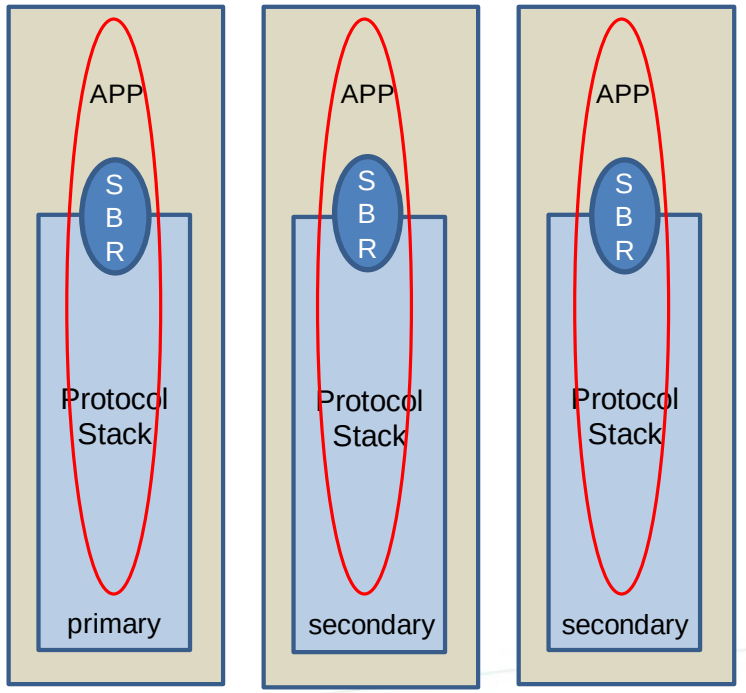
DMM: Distinguishing feature for protocol stack

- Socket layer support both Pipeline mode & RTC mode
- Protocol routing capability
- Fd management
- Epoll framework
- …

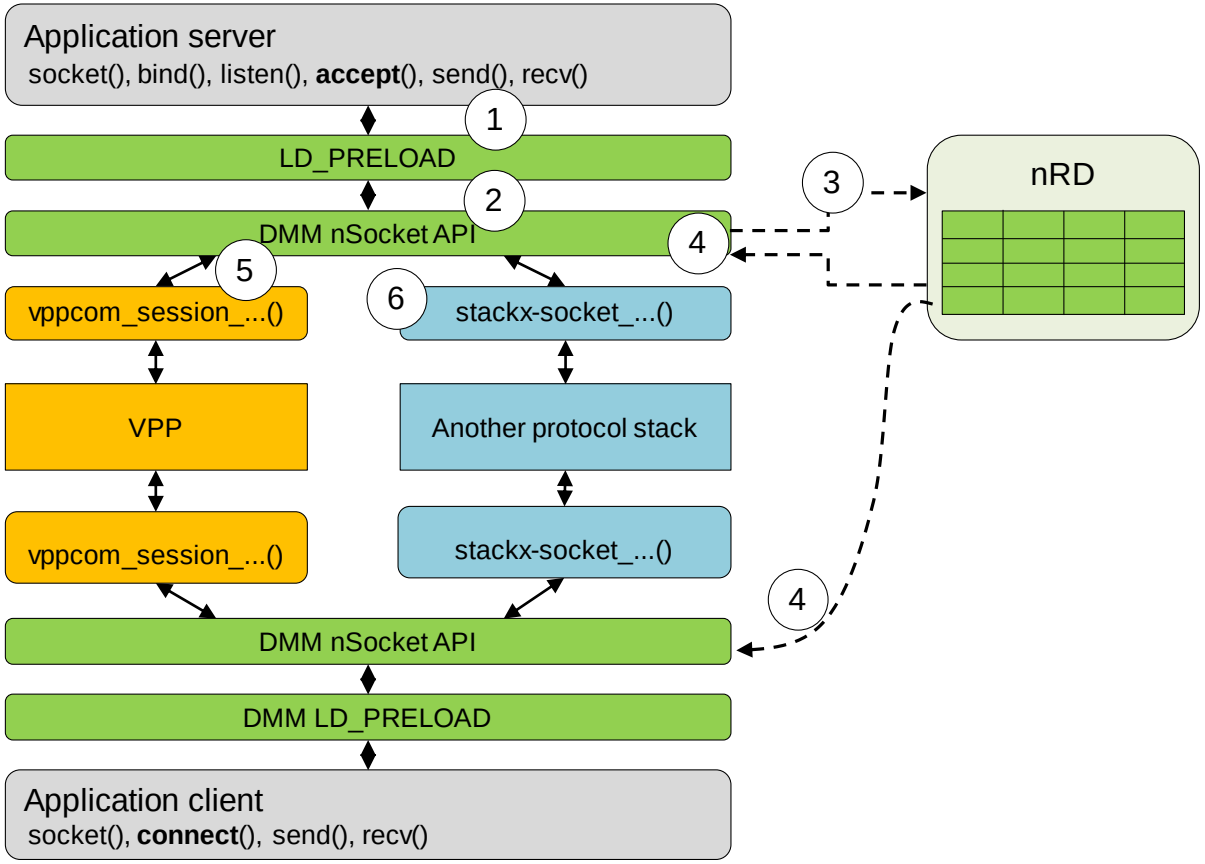
DMM: Pipeline & RTC mode integration

- The flavor is different，socket layer will support both



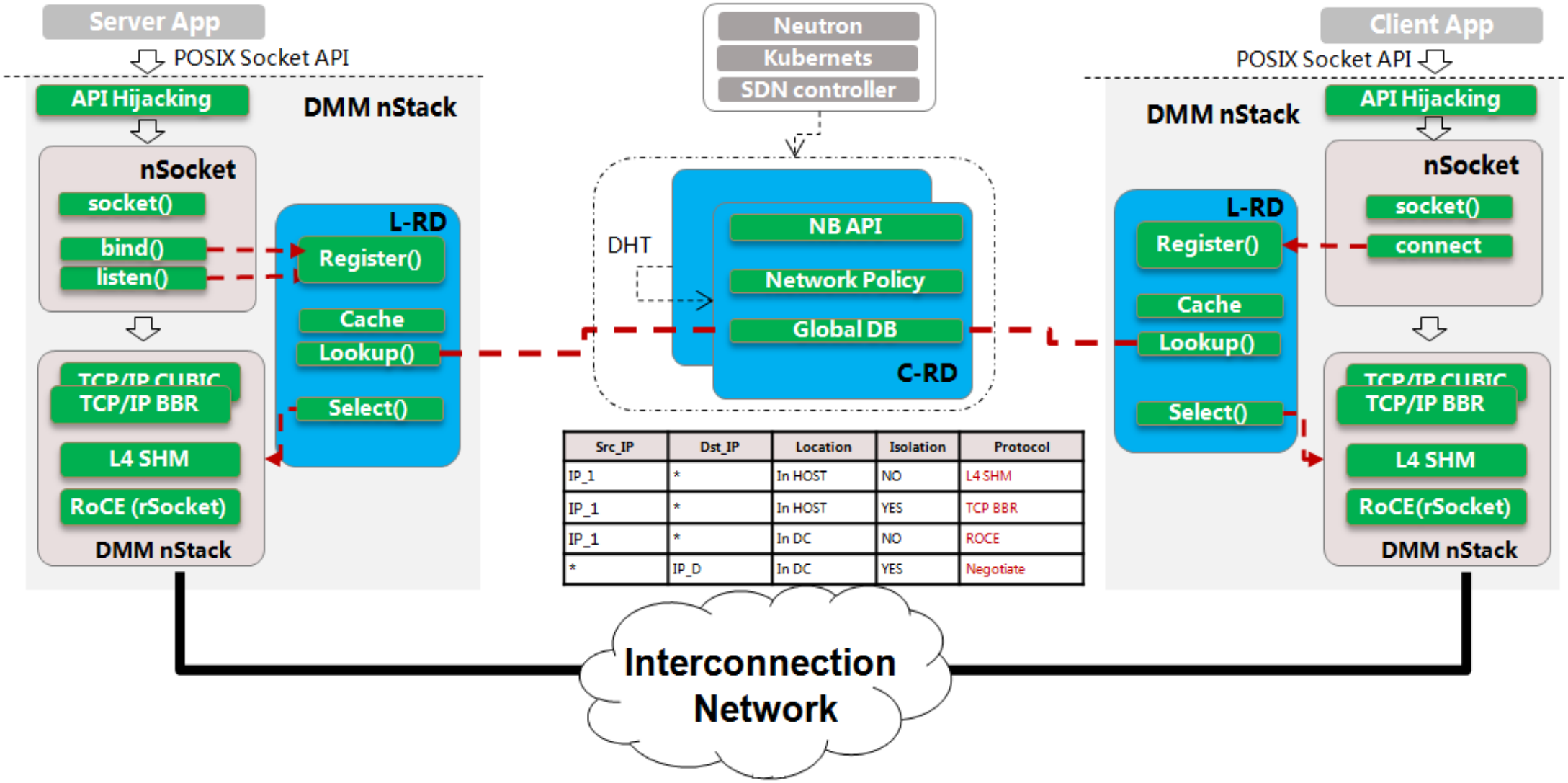


DMM: Protocol Routing



1. Application server and client calls socket interface.
2. Socket APIs are hijacked to DMM nSocket APIs.
3. Server call `listen()` triggers L-RD to publish capacity policies and preference policies to C-RD.
4. Server call `accept()` and client call `connect()` trigger L-RD to retrieve and resolve protocol stack mapping.
5. According to the mapping, the socket is instantiated to one protocol stack or another.
6. Dual mode(kernel or user-space), Multiple protocols, Multiple instances can exist simultaneously.

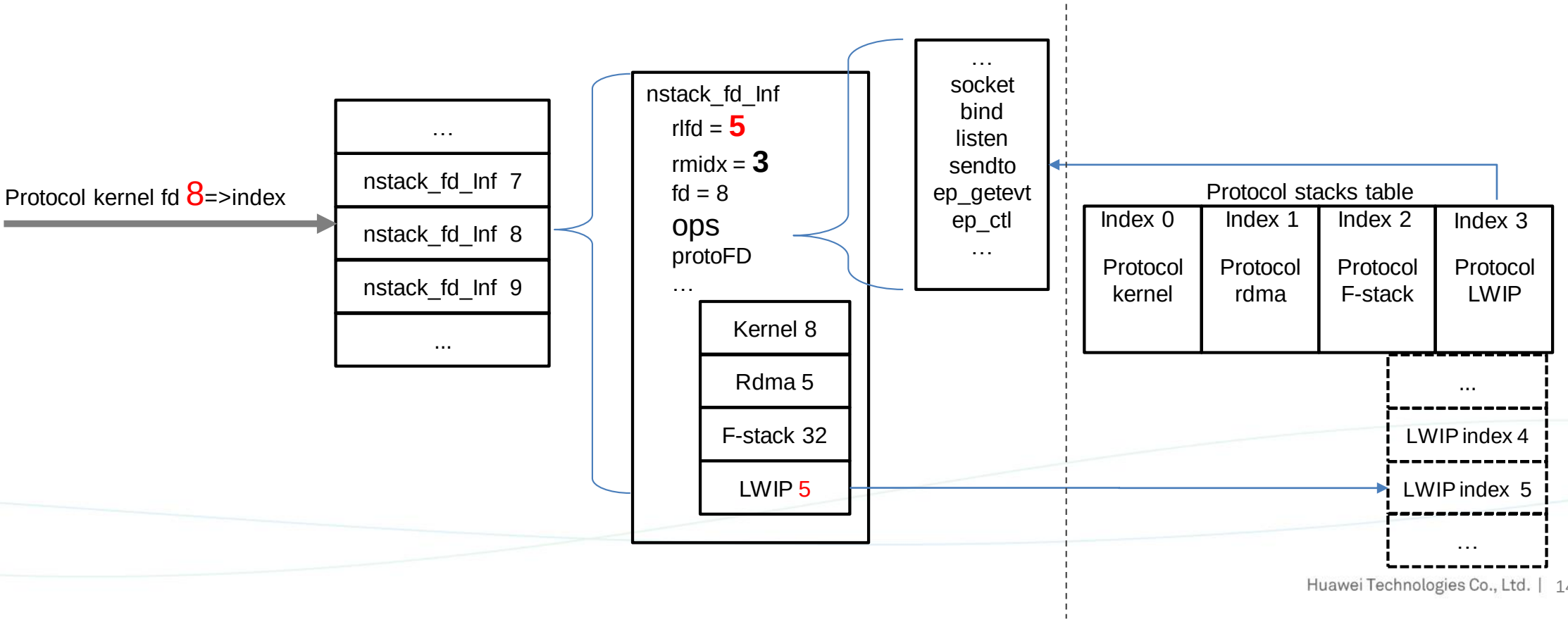
DMM: End to End Network Protocol Orchestration



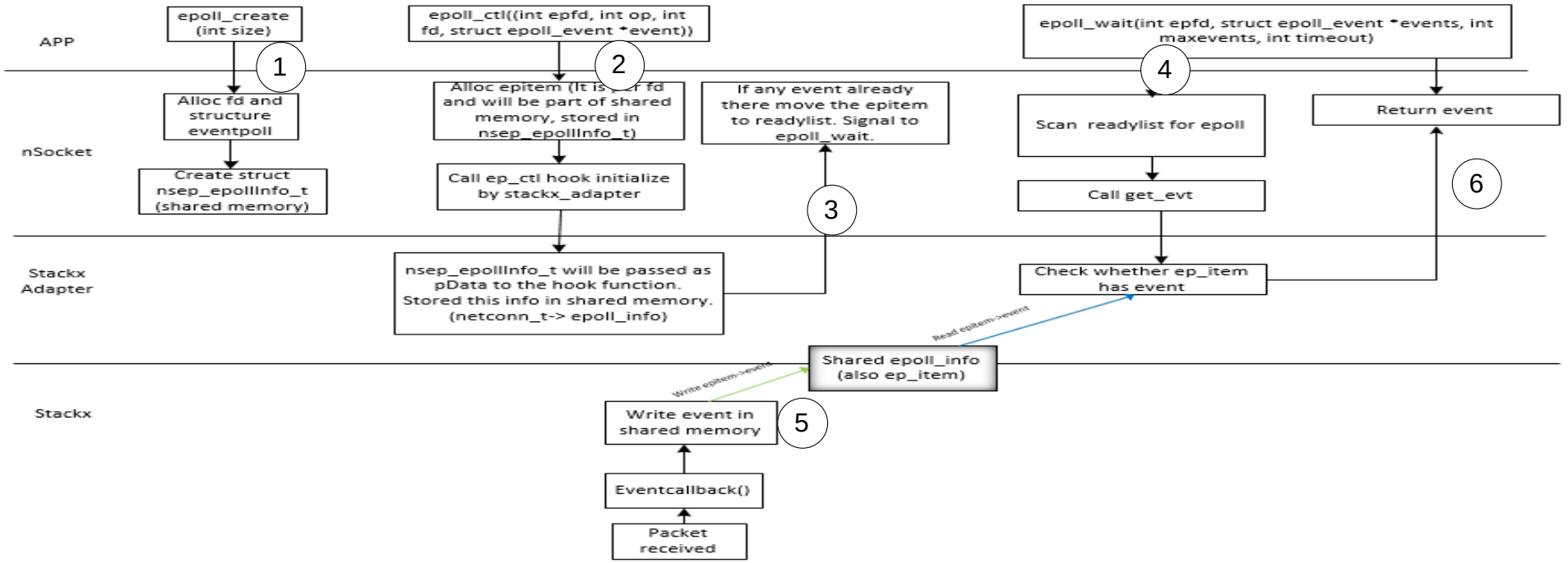
Src_IP	Dst_IP	Location	Isolation	Protocol
IP_1	*	In HOST	NO	L4 SHM
IP_1	*	In HOST	YES	TCP BBR
IP_1	*	In DC	NO	ROCE
*	IP_D	In DC	YES	Negotiate

DMM: Fd Management

- Distinguish fd
- Monitor app lifecycle in pipeline mode integration
- Support epoll/select to loop mixed stack fds



DMM: Epoll Framework



- ✓ APP creates ePoll and nSocket create an epoll info and group file descriptor for this epoll.
- ✓ This epoll info manages the list of socket fd's and its type which this epoll is interested in.
- ✓ Upon epoll_ctl the list of interested fd's are added to this epoll list in nSocket.
- ✓ nSocket inform about this ctl info(shared private data) to the stackpool adapter to monitor these sockets(via shared memory) and notify nsocket on receiving any data.

DMM: More Benefits

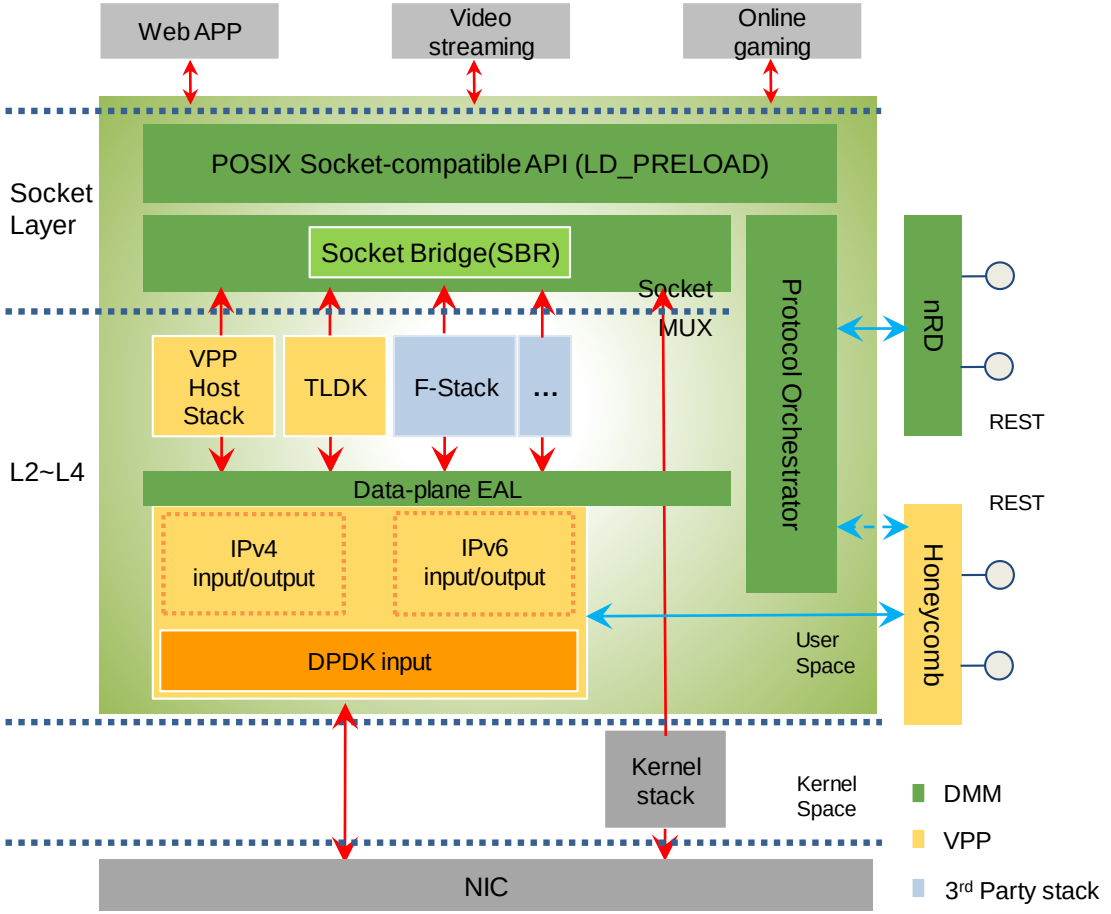
- Help to implement the “fork” semantics in socket layer
- Help to dock various IO requirement (EAL)
- Help to design flexible socket API redirection and mapping (SBR)
- …

DMM: Recap the features

	DMM + protocol	us protocol stack
LD_PRELOAD	yes	yes
Modify source of app	no	Add socket plugin layer Add protocol specific API code
Specify the stack	RD	Specify in source code Configure file
Fork	yes	no
NIC	share	monopoly
Epoll	yes	re-implement
Kernel fd	yes	no
...	...	...

DMM: Key takeaways

- Flexibility to **dynamically choose different protocols** according to performance and/or functional requirements
- **End-to-end orchestration** to maintain stack instances and the app/socket-to-stack mappings
- Extendable transport protocol plug-in framework to host **multiple stack instances** simultaneously
- Let stack developers **concentrate** on user space protocol innovation



Agenda

- What protocol stack face today
- DMM introduction
- **Use cases and stack integration example**
- Roadmap

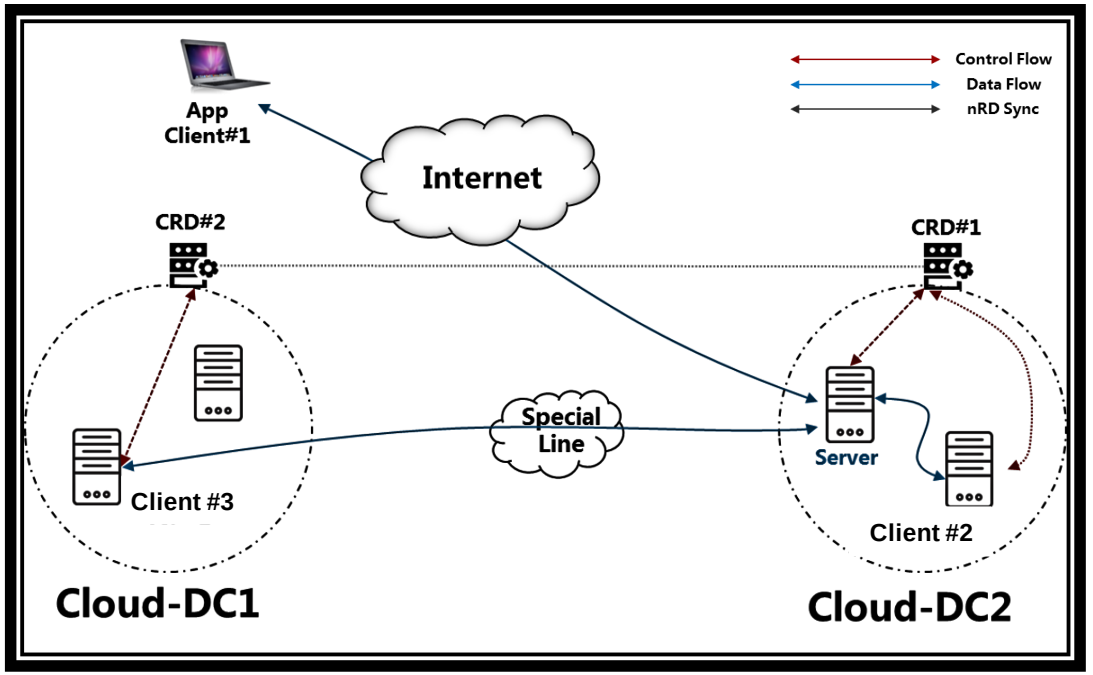
Booth Demo1: Protocol Routing for Multi-network Client-Server Application

File Sync Application

- 3 Clients --> Server

Network Setting

- Internet (Client #1)
- Intra DataCenter (Client #2)
- Inter DataCenter (Client #3)

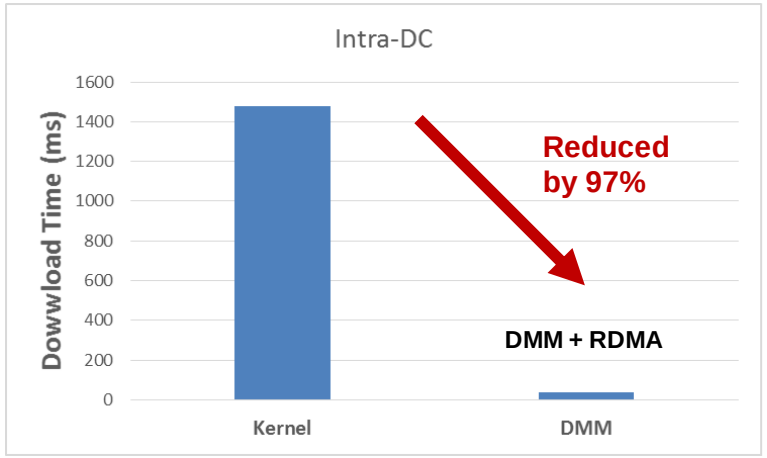


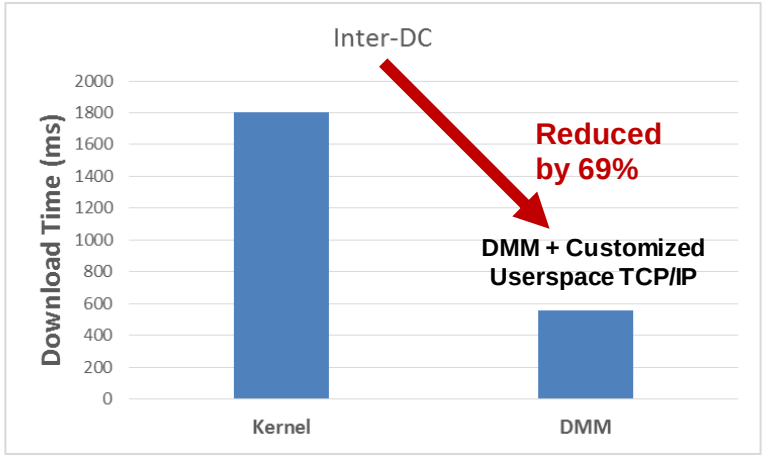
Comparison scheme

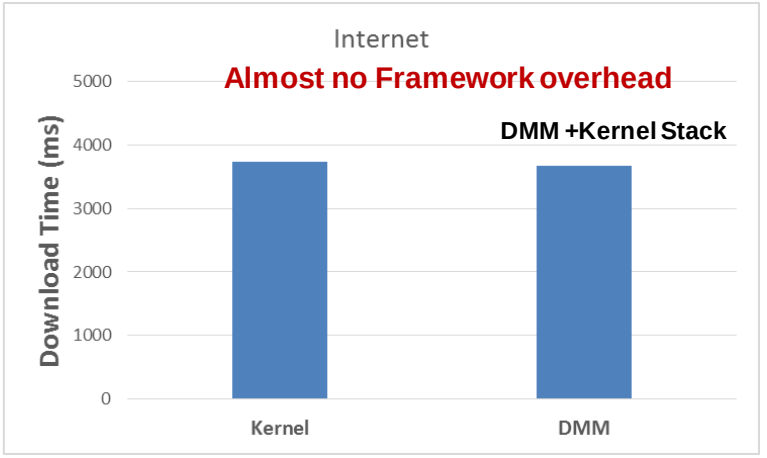
- Default: the kernel TCP/IP stack
- DMM: automatically negotiate appropriate stacks (kernel TCP/IP, customized user-space TCP/IP, RDMA) on different network

Booth Demo1: Protocol Routing for Multi-network Client-Server Application

No one stack/protocol fits all scenario, but by adaptively negotiating stack according to the network environment, DMM achieves significant performance improvement.







Booth Demo2 : Dual mode support for Nginx Server

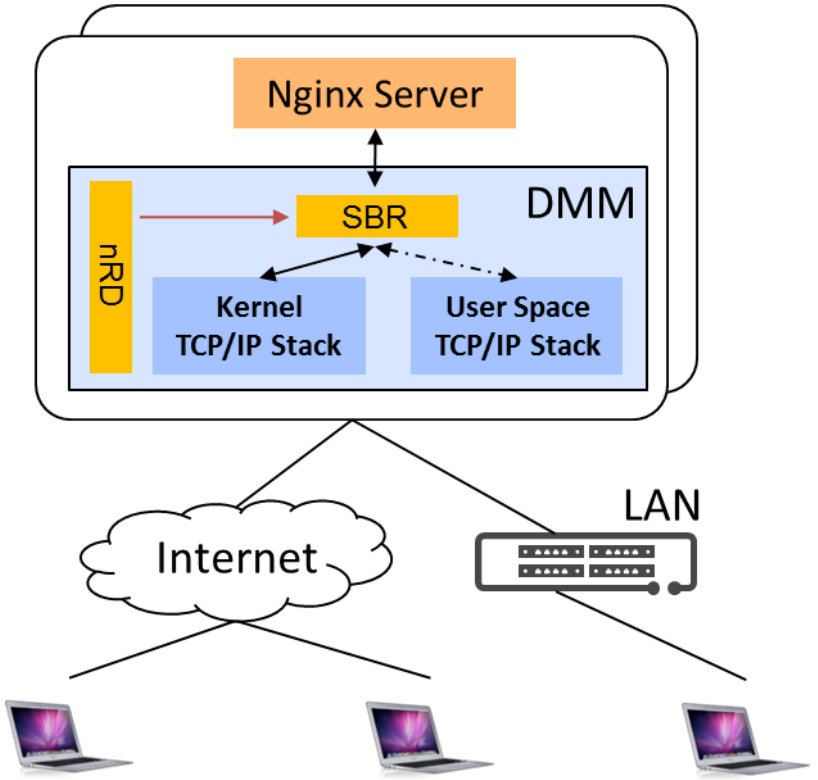
Nginx application

- kernel stack vs user-space stack ?

DMM nRD Policy (Example):

- Internet connection ---> kernel stack
- LAN connection ---> user-space stack

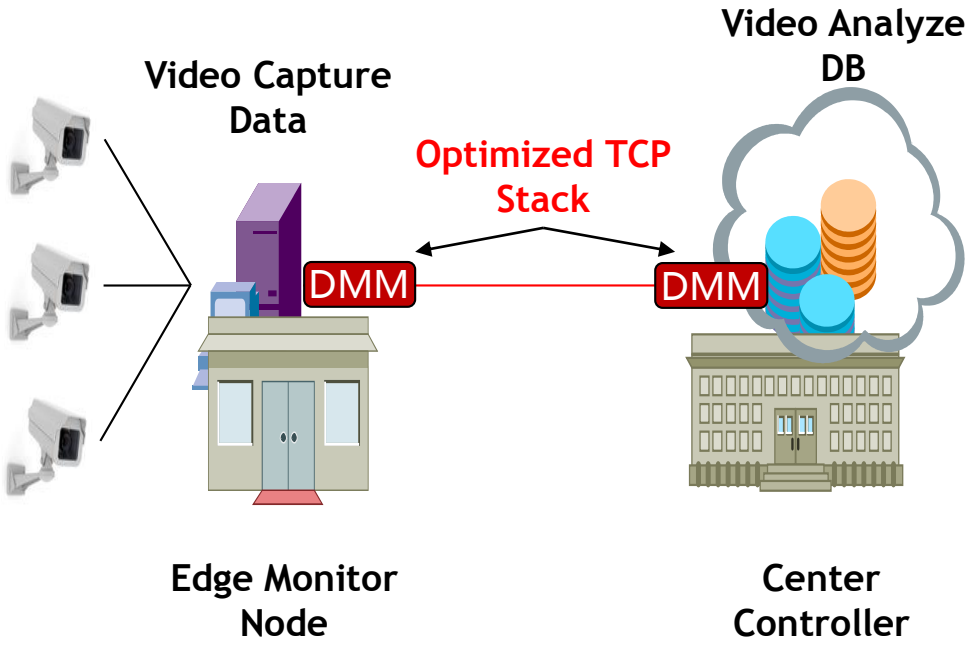
	Kernel	NSUE	DMM
Robustness	✓	×	✓
Performance	×	✓	✓
Customizability	×	✓	✓
Reliability	✓	×	✓



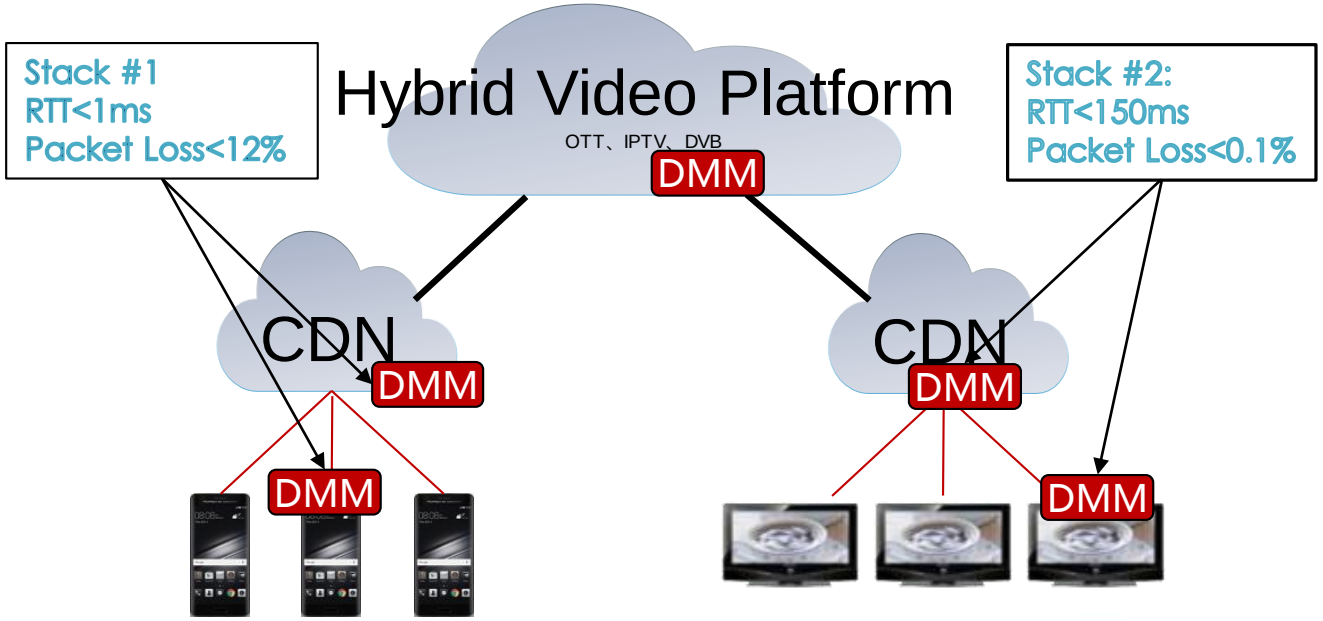
Using DMM Nginx application server could switch between kernel stack and user-space stack adaptively to use their advantages respectively under different scenario; and “D” mode help to process the kernel fd and event.

How DMM applied into production environment

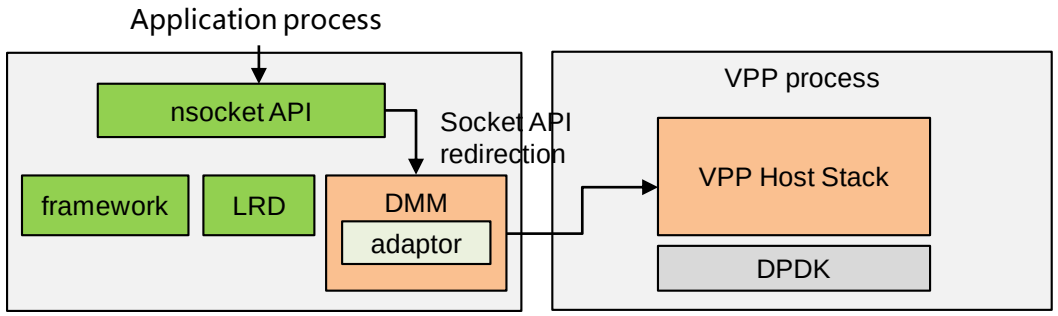
Optimized TCP stack in DMM can achieve 90% bandwidth usage and low latency, easy to deploy.



Optimized TCP stack in DMM can support mass concurrent connections and achieve smooth user experience even 12% packet loss rate.



Stack integration example 1: vpp hoststack



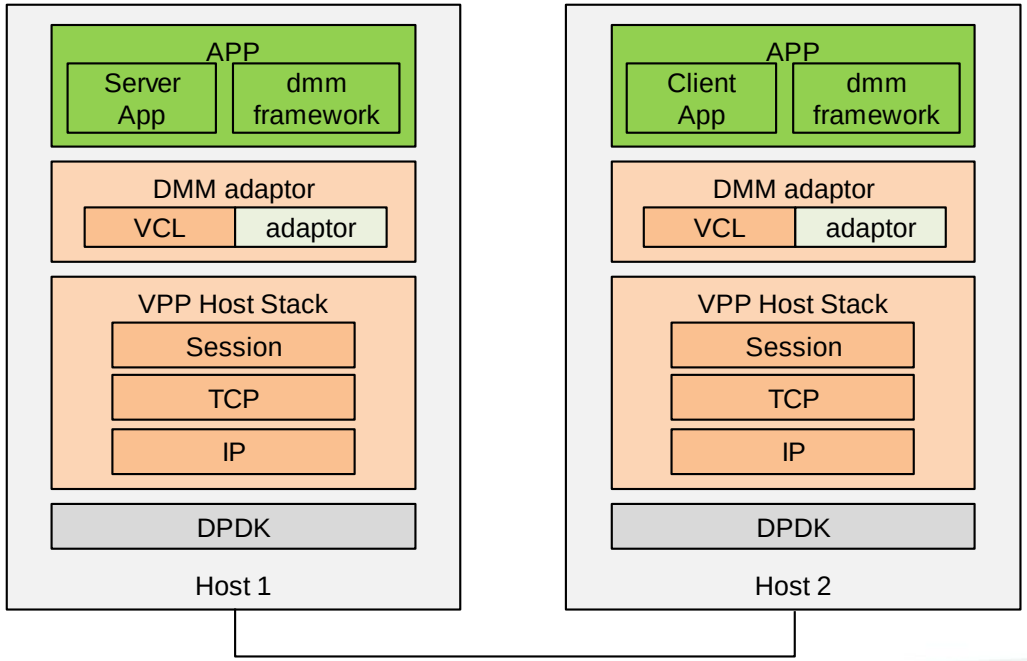
- module_config.json:

```
{
    ...
    {
        "stack_name": "vpp_host_stack",
        "function_name": "vpphs_stack_register",
        "libname": "libdmm_vcl.so",
        ...
    }
}
```

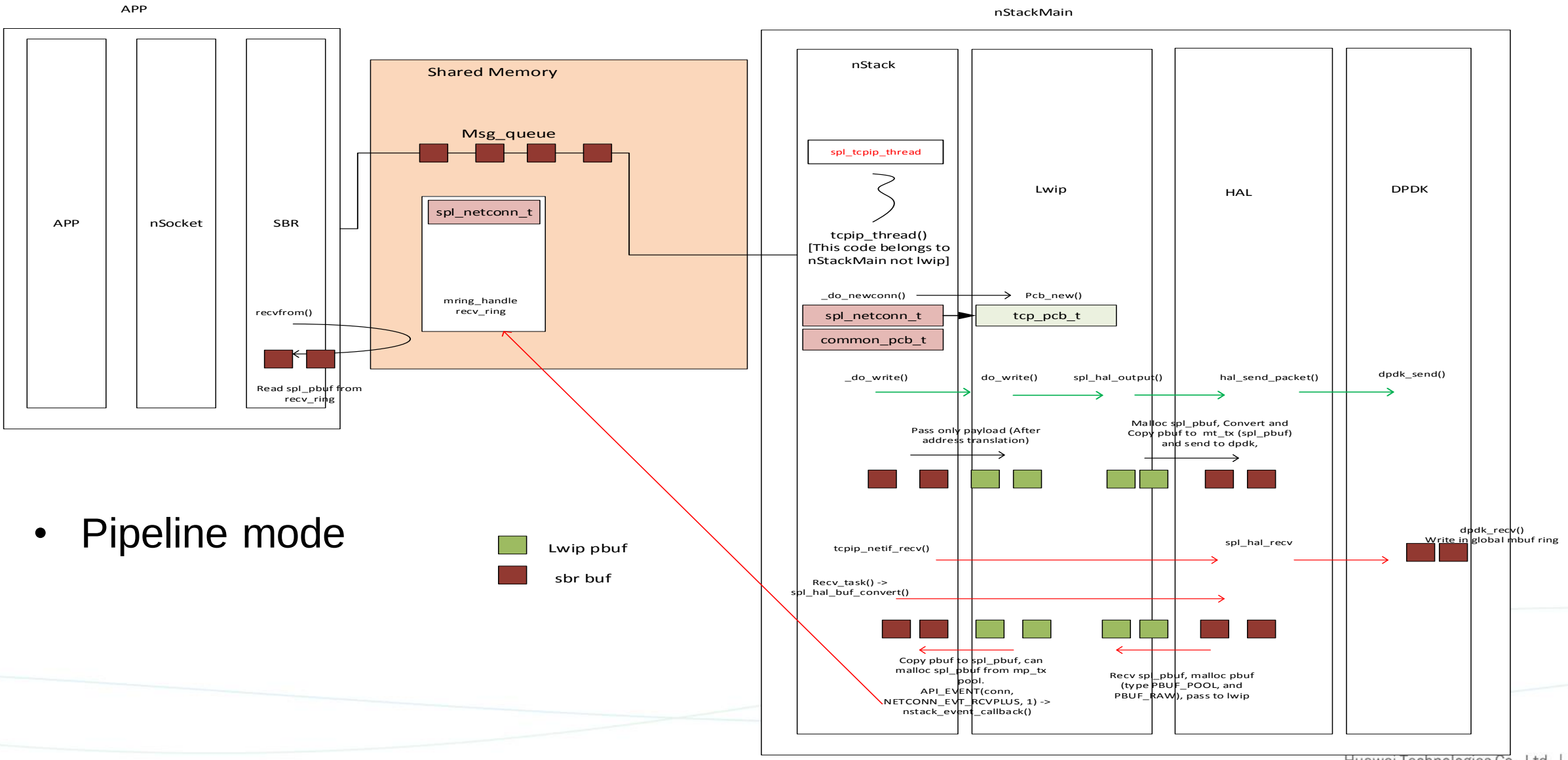
- dmm_vcl_adpt.c is built into libdmm_vcl.so.

```
vpphs_stack_register()
{
    // load symbol from "libname"
    nstack_proc_cb->socket_ops.pf##fn = dlsym(val->handle, #fn);

    nstack_proc_cb->extern_ops.ep_ctl = vpphs_fd_ep_trigger;
    nstack_proc_cb->extern_ops.ep_getEvt = vpphs_fd_ep_getEvt;
    ...
}
```



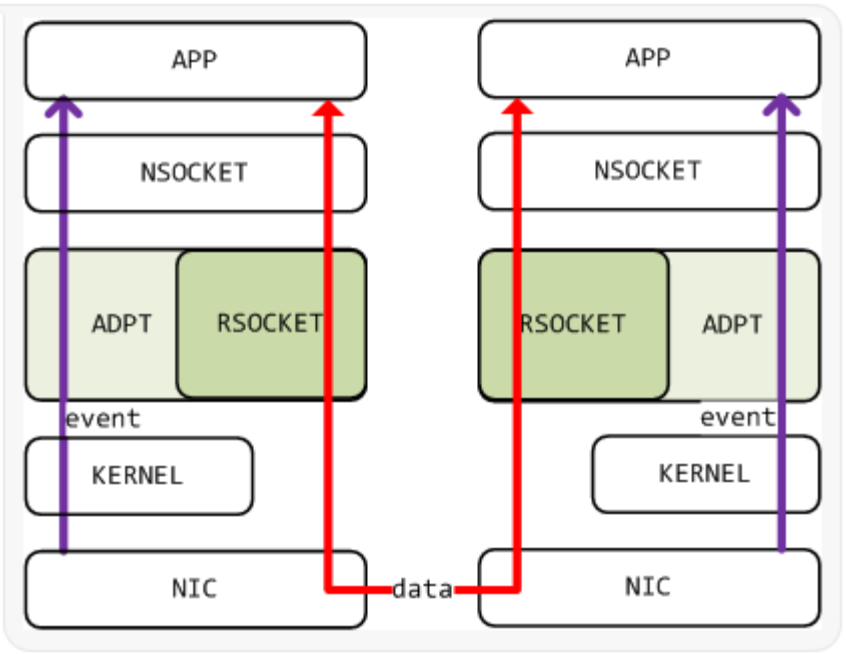
Stack integration example 2: LWIP+dpdk

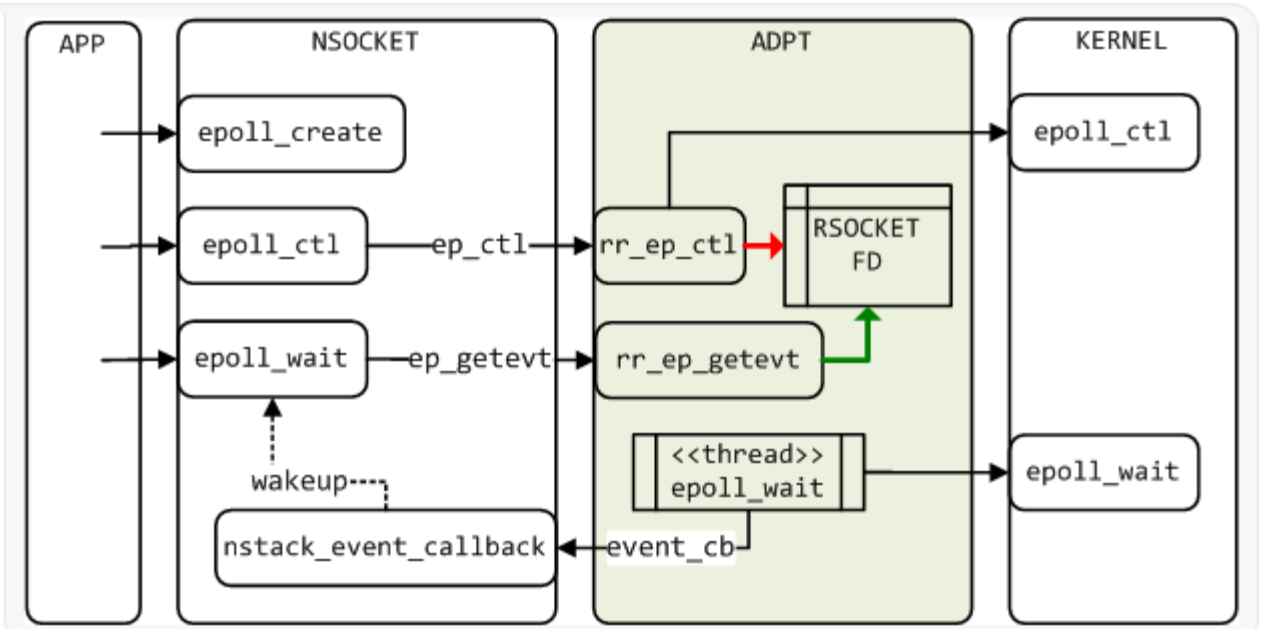


- Pipeline mode

Stack integration example 3: rsocket

- RTC mode





Agenda

- What protocol stack face today
- DMM introduction
- Use cases and stack integration example
- **Roadmap**

DMM: Roadmap

- v18.07
  - ✓ Integrate rsocket stack
  - ✓ Initial support for pipeline userspace LWIP(DPDK) and vpp hoststack
  - ✓ Manual on how to integrate a stack into DMM framework
  - ✓ DMM pkg release in rpm and deb

- v18.10
  - ✓ RTC F-Stack、vpp hoststack、LWIP example into DMM
  - ✓ Support “Fork” example
  - ✓ provide “SBR” example
  - ✓ DMM Performance optimization
  - ✓ Enhance the “contactless” for APP

Join us

FD.io DMM project

- https://wiki.fd.io/view/DMM

Repo

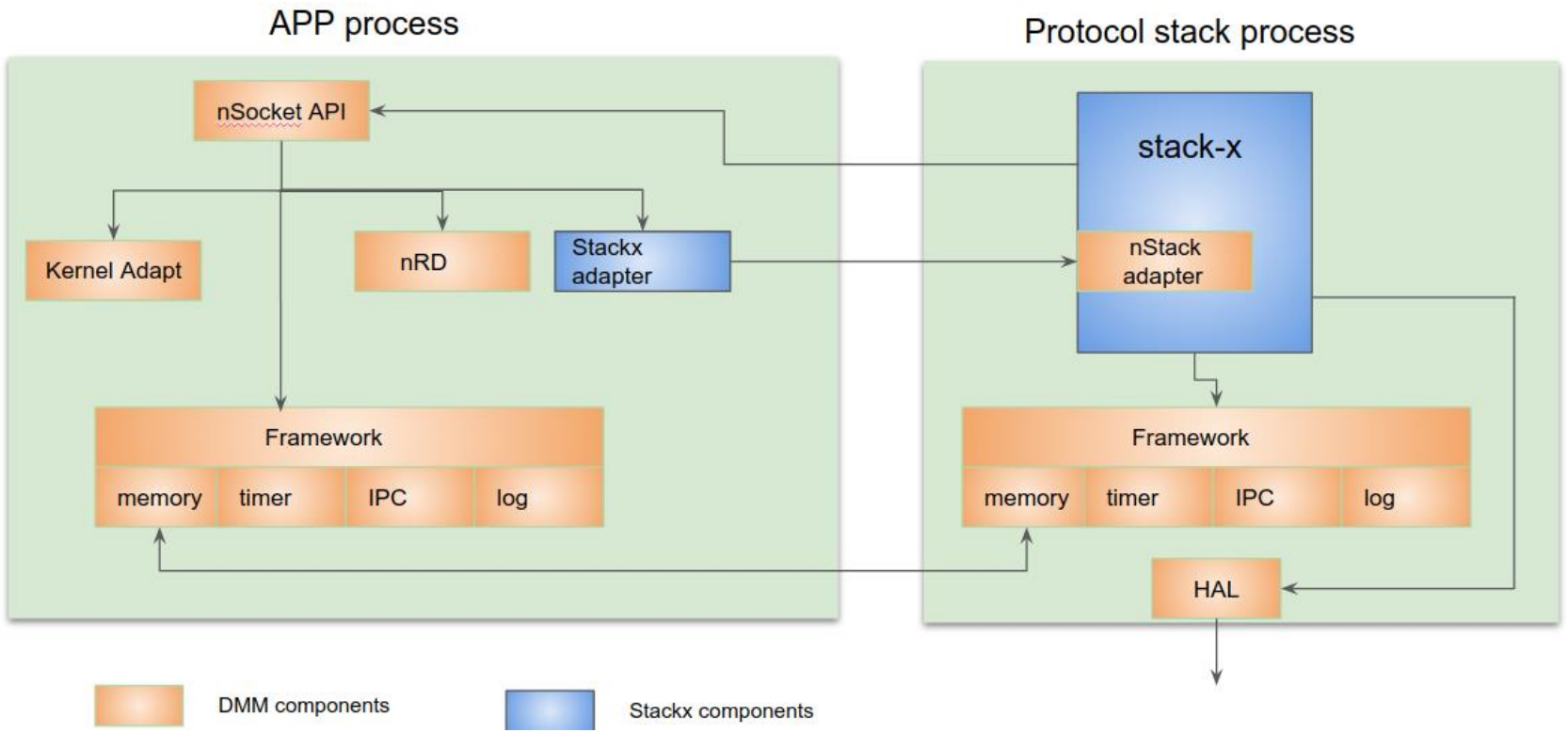
- https://git.fd.io/dmm

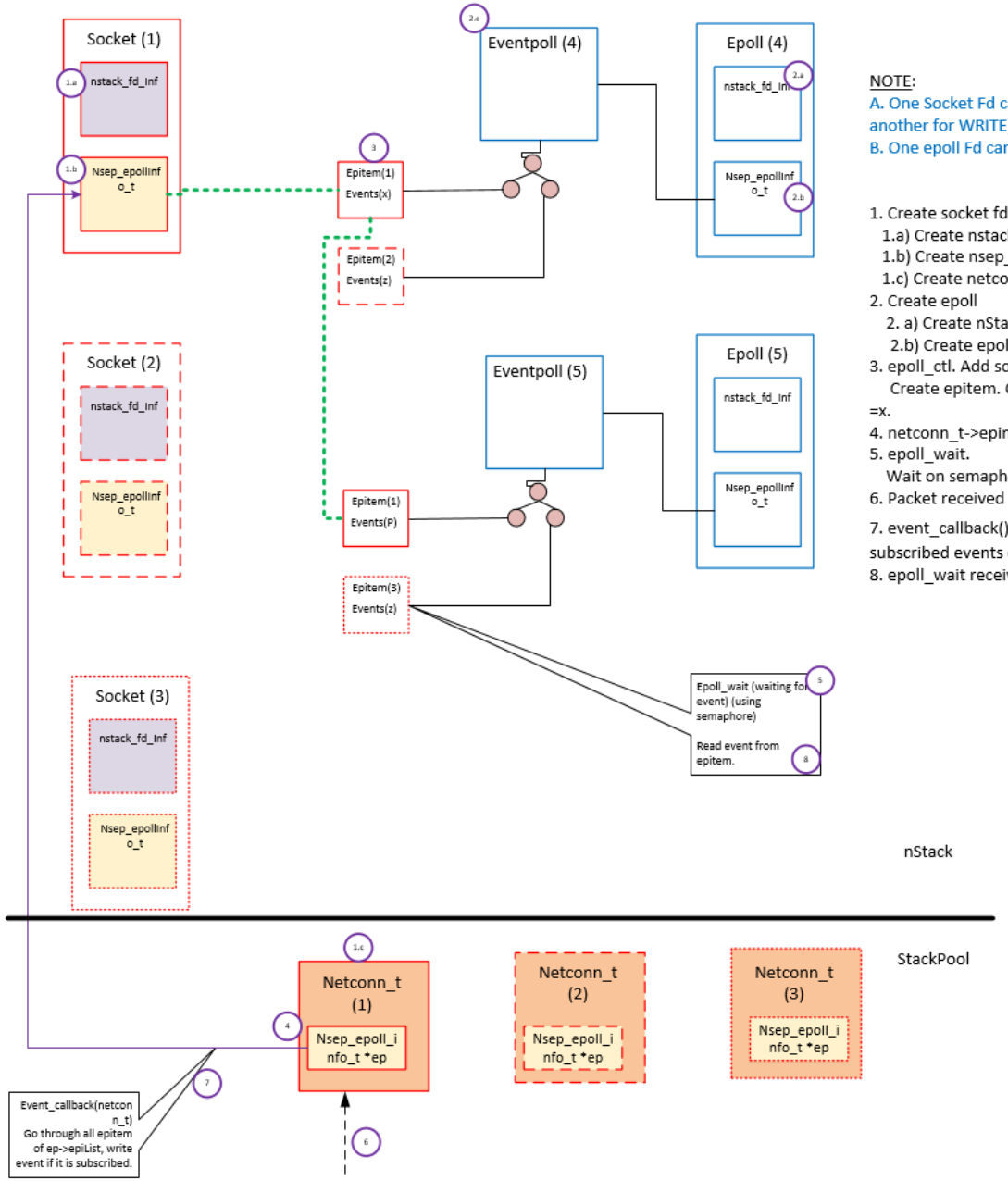
Involved in

- Maillist: dmm-dev@lists.fd.io
- IRC: #fdio-dmm

Thank You.

Copyright©2016 Huawei Technologies Co., Ltd. All Rights Reserved.
The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.


APP process
Protocol stack process
nSocket API
Kernel Adapt
nRD
Stackx adapter
Framework
memory
timer
IPC
log
stack-x
nStack adapter
Framework
memory
timer
IPC
log
HAL
DMM components
Stackx components



NOTE:

A. One Socket Fd can be added to multiple epoll Fd. (Example scenario: one epoll fd for READ EVENT another for WRITE)

B. One epoll Fd can have multiple socket Fd.

1. Create socket fd = 1
   1.a) Create nstack_fd_Inf fd = 1
   1.b) Create nsep_epollInfo_t fd = 1
   1.c) Create netconn_t which will be used for communication between nStack and Stackpool. Fd = 1.
2. Create epoll
   2. a) Create nStack_fd_inf fd = 4
   2.b) Create epoll_info_t fd = 4
3. epoll_ctl. Add socket fd 1 to epoll fd 4.
   Create epitem. One epitem represents **one socket and one epoll fd** relation. Subscribed events =x.
4. netconn_t->epinfo now points to soctes epoll info.
5. epoll_wait.
   Wait on semaphore. Check any events are there.
6. Packet received for netconn_t. (stackpool).
7. event_callback() function: It will go through all the epitem (---) inf epinfo->epiList, and **write** if subscribed events occurred. Epitems may belong to different epoll fd. Signal the semaphore.
8. epoll_wait received semaphore signal. It **reads** the event, move the epitem to readylist.

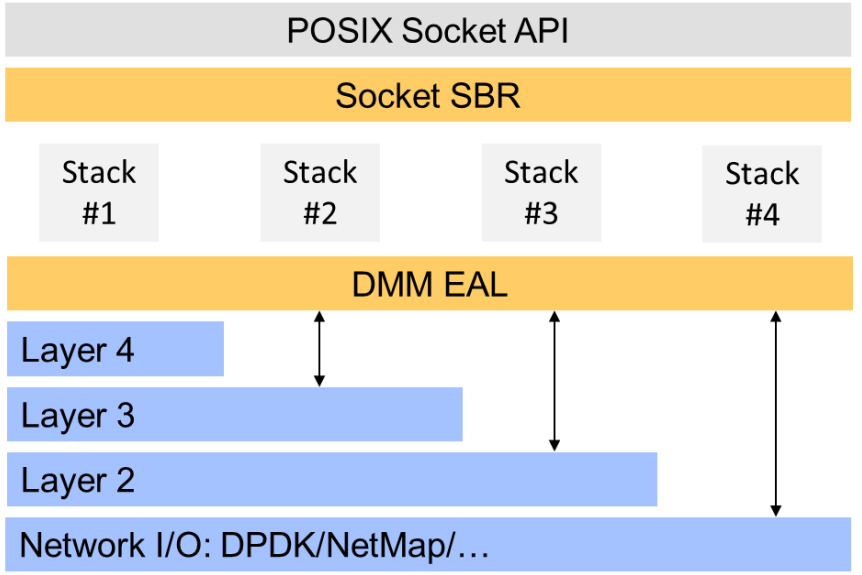
DMM: Benefit To Stack/Protocol Developers

Friendly interfaces to integrate

- ✓ Mechanisms for socket redirect
- ✓ Flexible I/O APIs including NIC/L2/L3/L4 APIs

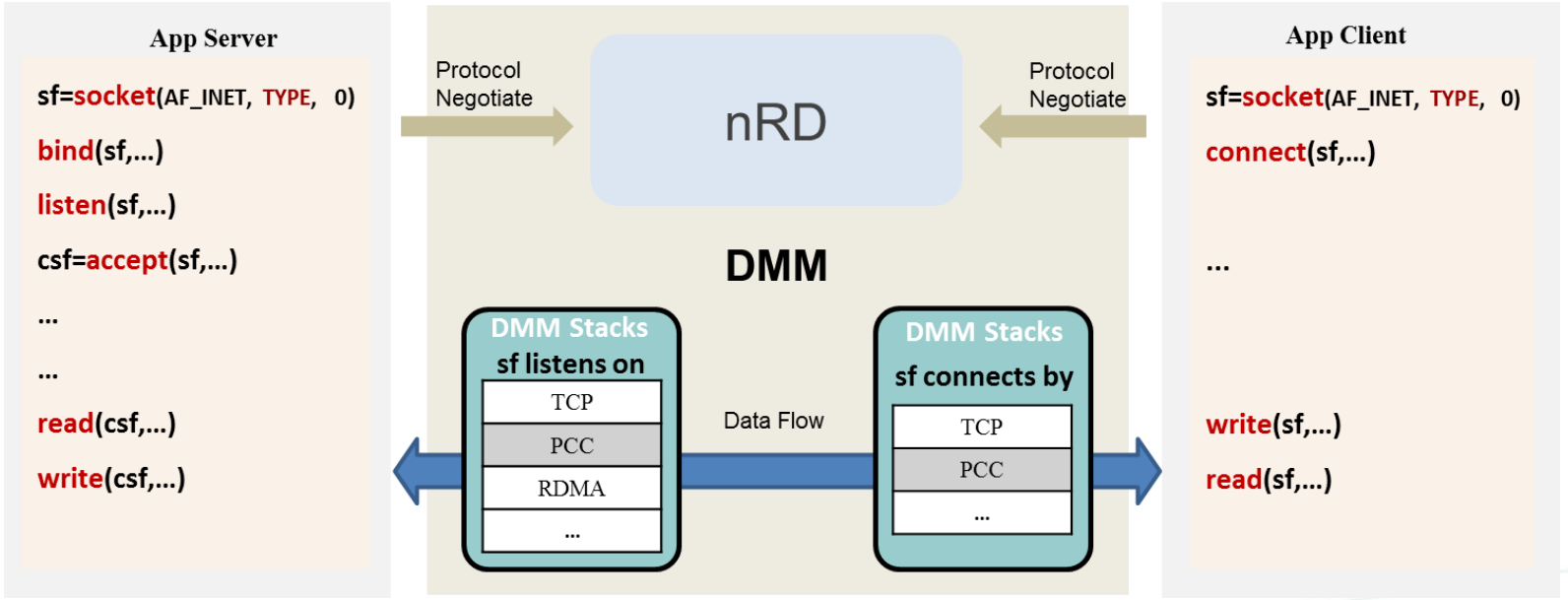
Accelerate innovation of new stacks

- ✓ Concentrate on the core function of the stack
- ✓ Easy to be deployed
- ✓ Easy to be adopted/tested by applications



DMM: Benefit To Application Developers

- ✓ Extended POSIX socket APIs which is backward compatible for legacy applications
- ✓ ‘Protocol Routing’ based on network env, application requirements and host information
- ✓ Rapidly benefit from the new stacks/protocols without any changes on the application code



‘Protocol Routing’ workflow

DMM-Demo-2: Dual mode support for Nginx Server

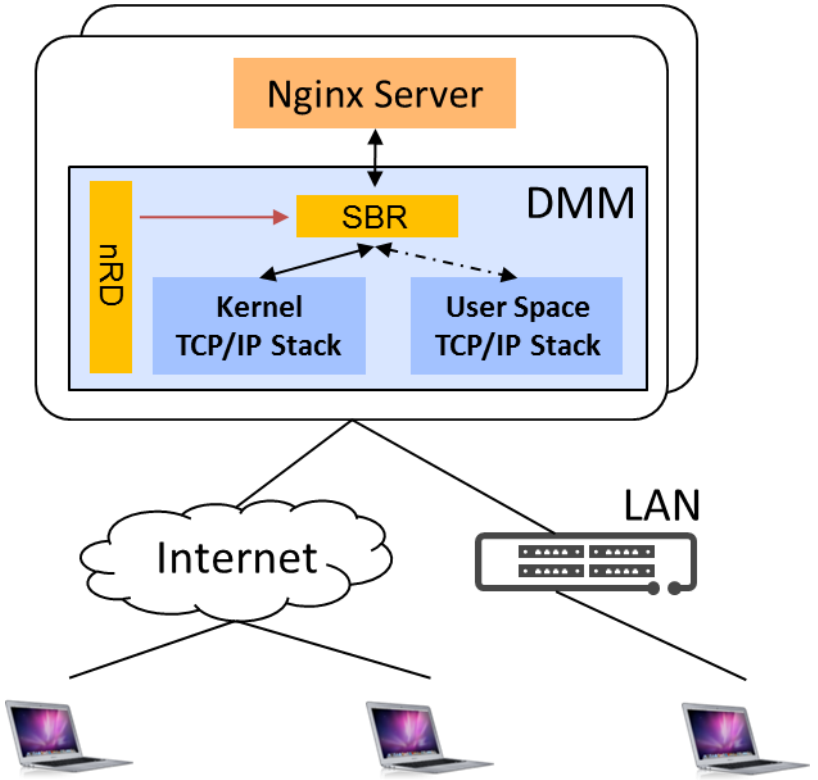
Nginx application

- kernel stack vs user-space stack ?

DMM nRD Policy (Example):

- Internet connection ---> kernel stack
- LAN connection ---> user-space stack

	Kernel	NSUE	DMM
Robustness	✓	×	✓
Performance	×	✓	✓
Customizability	×	✓	✓
Reliability	✓	×	✓



Using DMM Nginx application server could switch between kernel stack and user-space stack adaptively to use their advantages respectively under different scenario

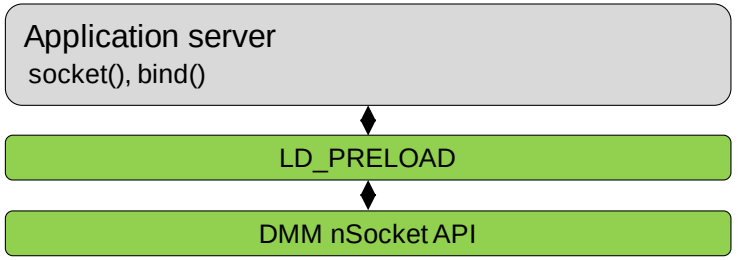
DMM: Protocol Routing Workflow

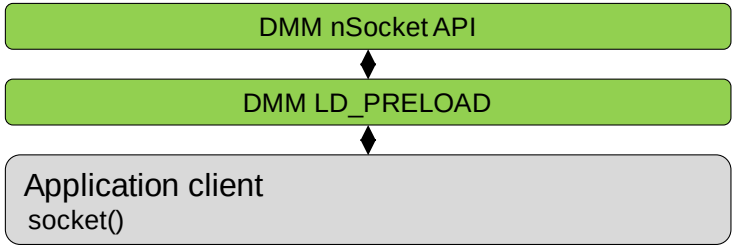
Application server
socket(), bind()

Application client
socket()

1 Application server and client calls socket interface.

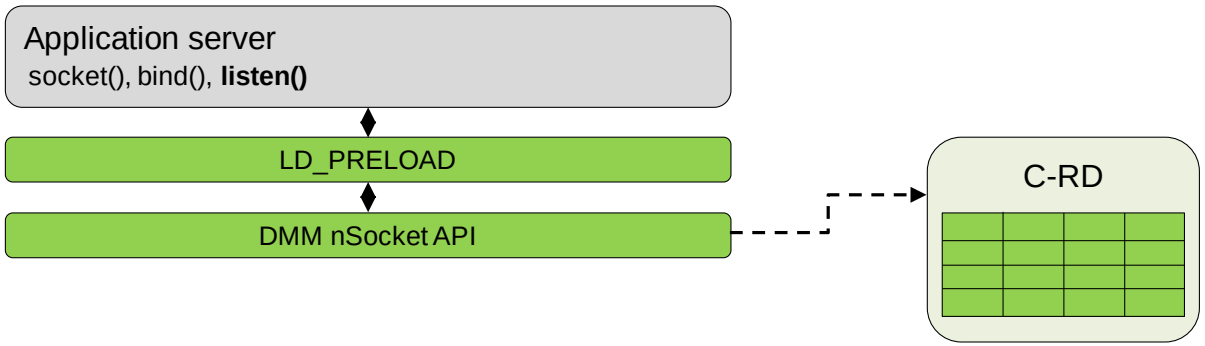
DMM: Protocol Routing Workflow



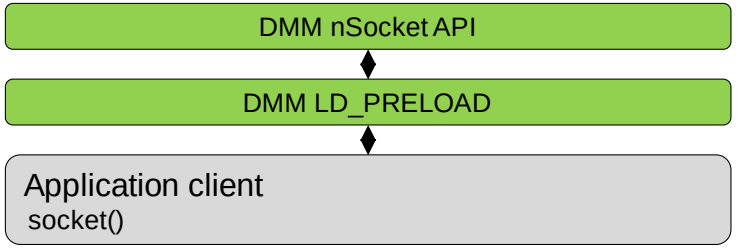


1. Application server and client calls socket interface.
2. Socket APIs are hijacked to DMM nSocket APIs.

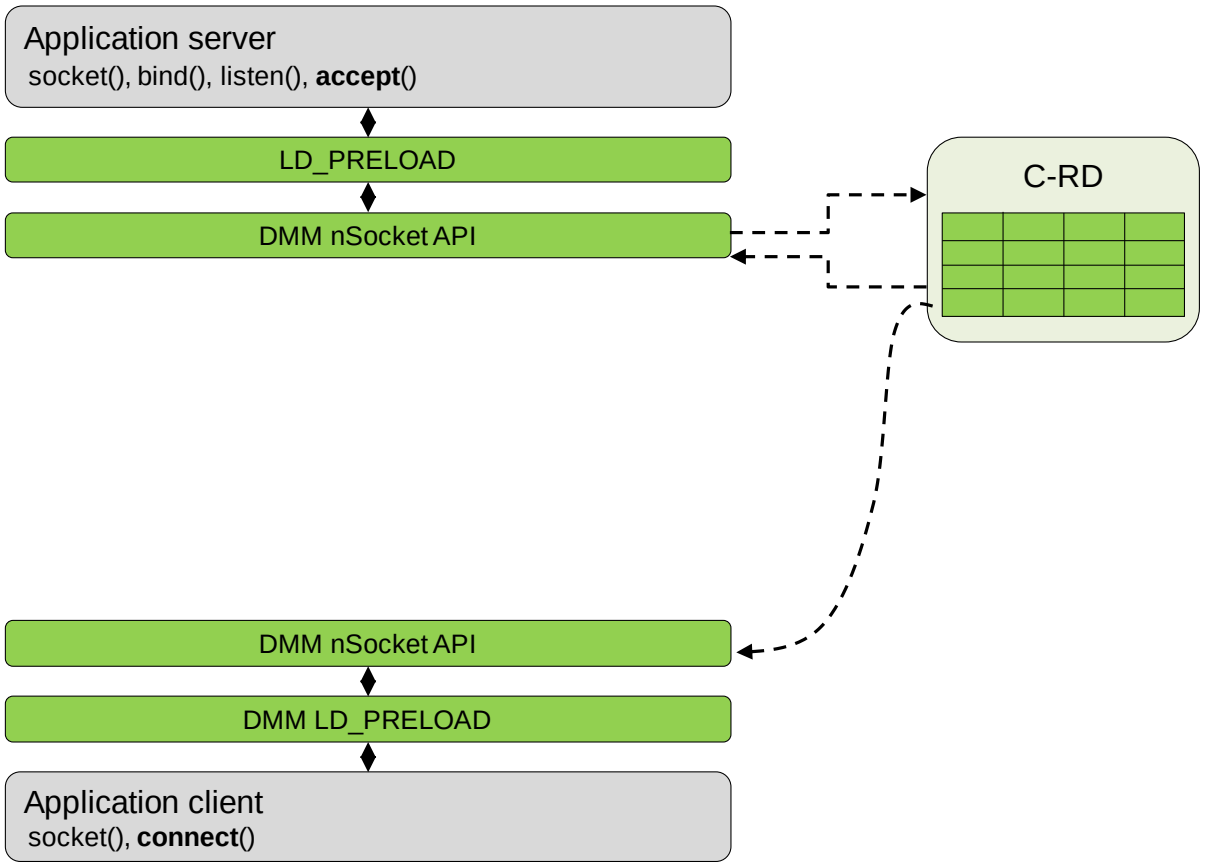
DMM: Protocol Routing Workflow



1. Application server and client calls socket interface.
2. Socket APIs are hijacked to DMM nSocket APIs.
3. Server call `listen()` triggers L-RD to publish capacity policies and preference policies to C-RD.

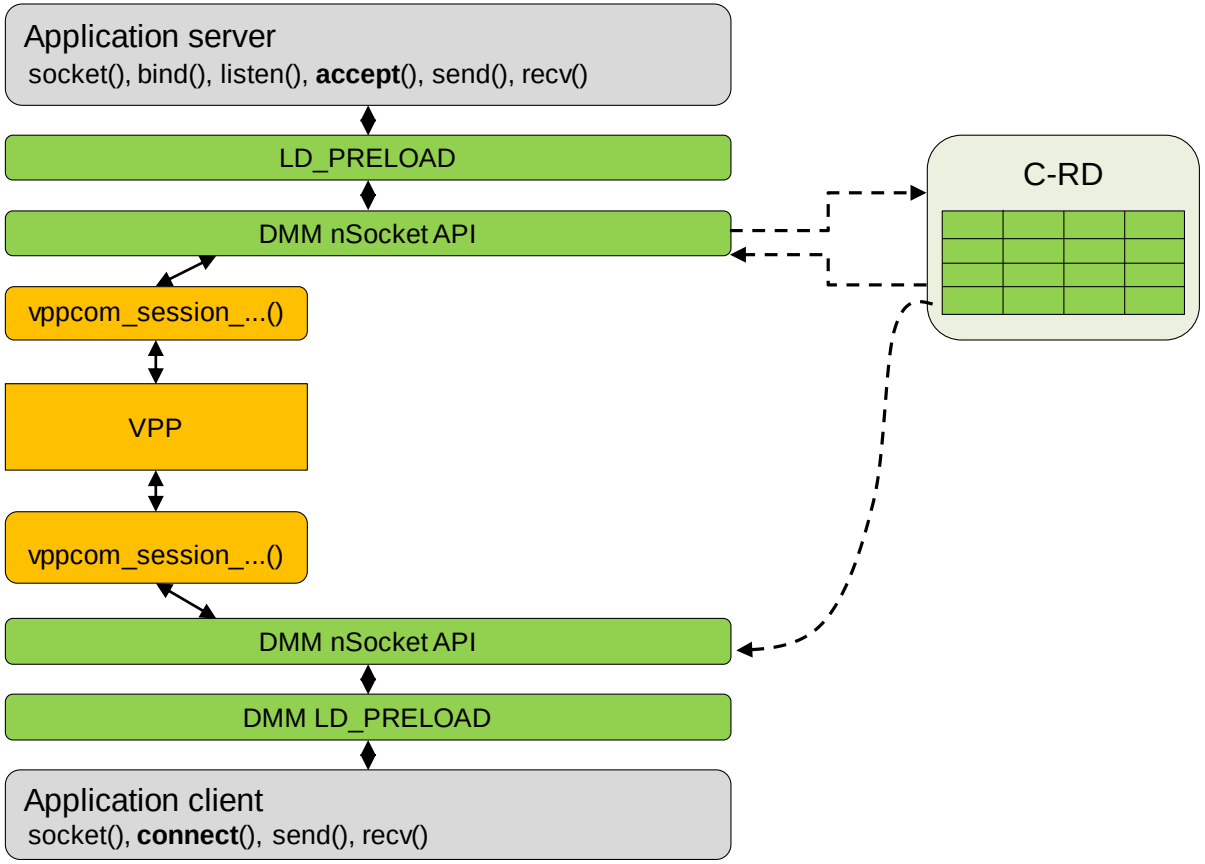


DMM: Protocol Routing Workflow



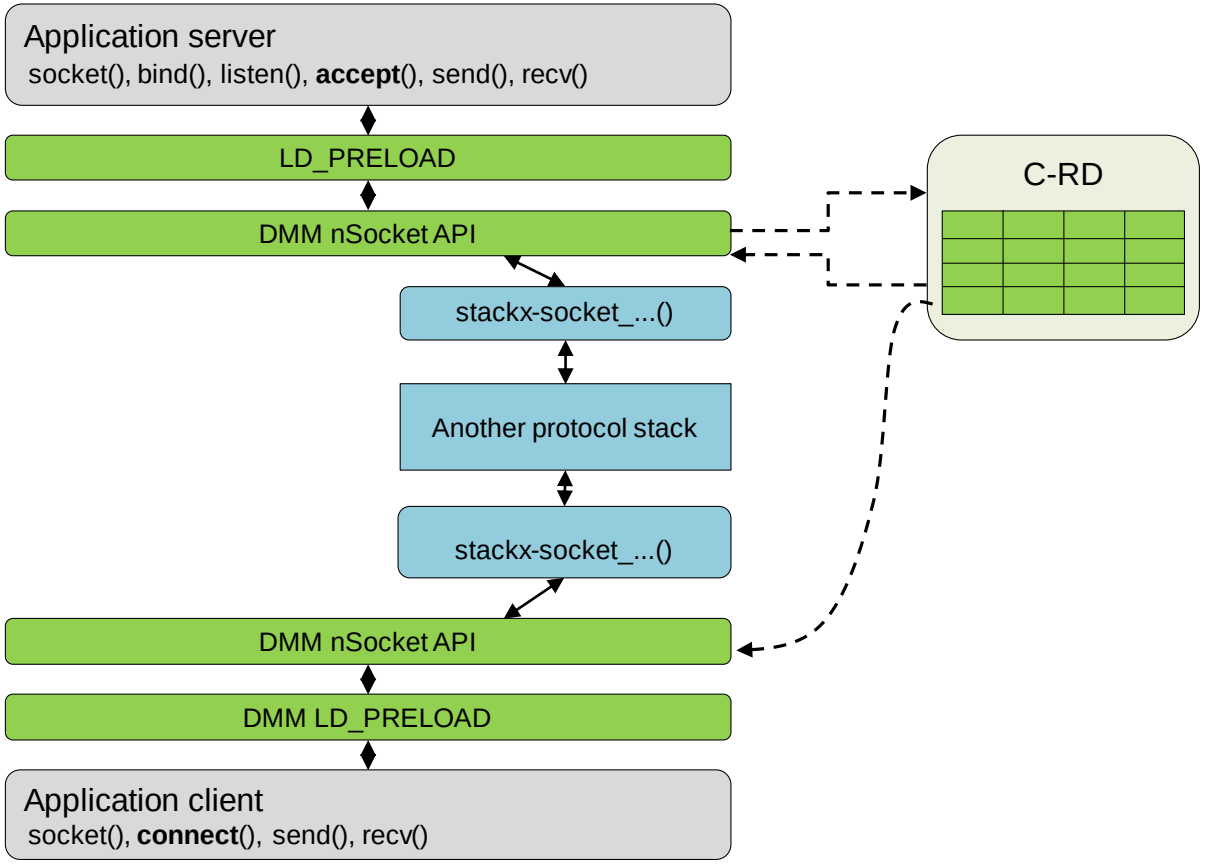
1. Application server and client calls socket interface.
2. Socket APIs are hijacked to DMM nSocket APIs.
3. Server call `listen()` triggers L-RD to publish capacity policies and preference policies to C-RD.
4. Server call `accept()` and client call `connect()` trigger L-RD to retrieve and resolve protocol stack mapping.

DMM: Protocol Routing Workflow



1. Application server and client calls socket interface.
2. Socket APIs are hijacked to DMM nSocket APIs.
3. Server call `listen()` triggers L-RD to publish capacity policies and preference policies to C-RD.
4. Server call `accept()` and client call `connect()` trigger L-RD to retrieve and resolve protocol stack mapping.
5. According to the mapping, the socket is instantiated to one protocol stack

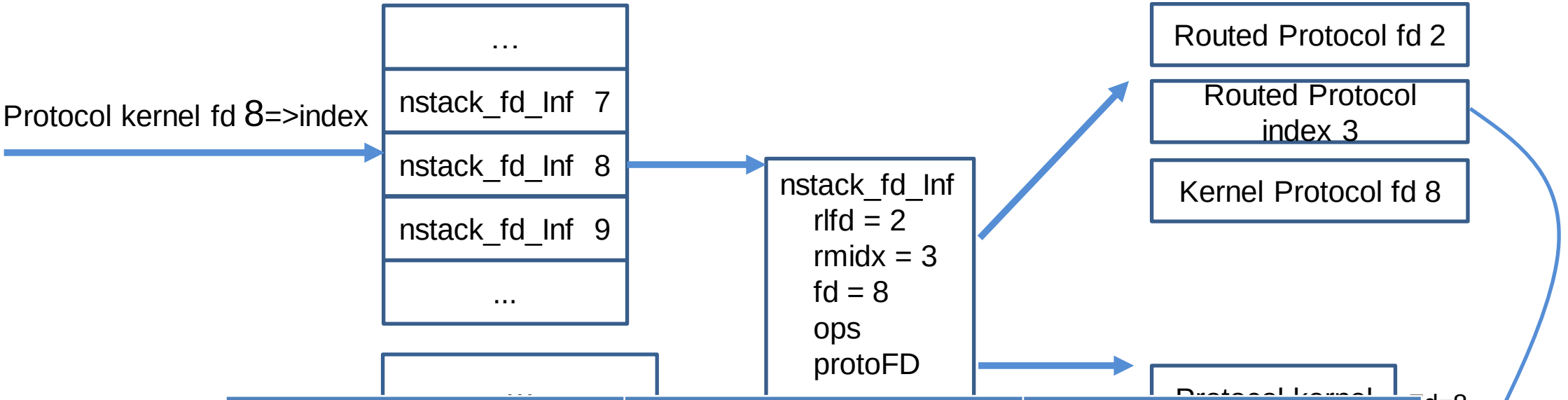
DMM: Protocol Routing Workflow



1. Application server and client calls socket interface.
2. Socket APIs are hijacked to DMM nSocket APIs.
3. Server call `listen()` triggers L-RD to publish capacity policies and preference policies to C-RD.
4. Server call `accept()` and client call `connect()` trigger L-RD to retrieve and resolve protocol stack mapping.
5. According to the mapping, the socket is instantiated to one protocol stack or another.

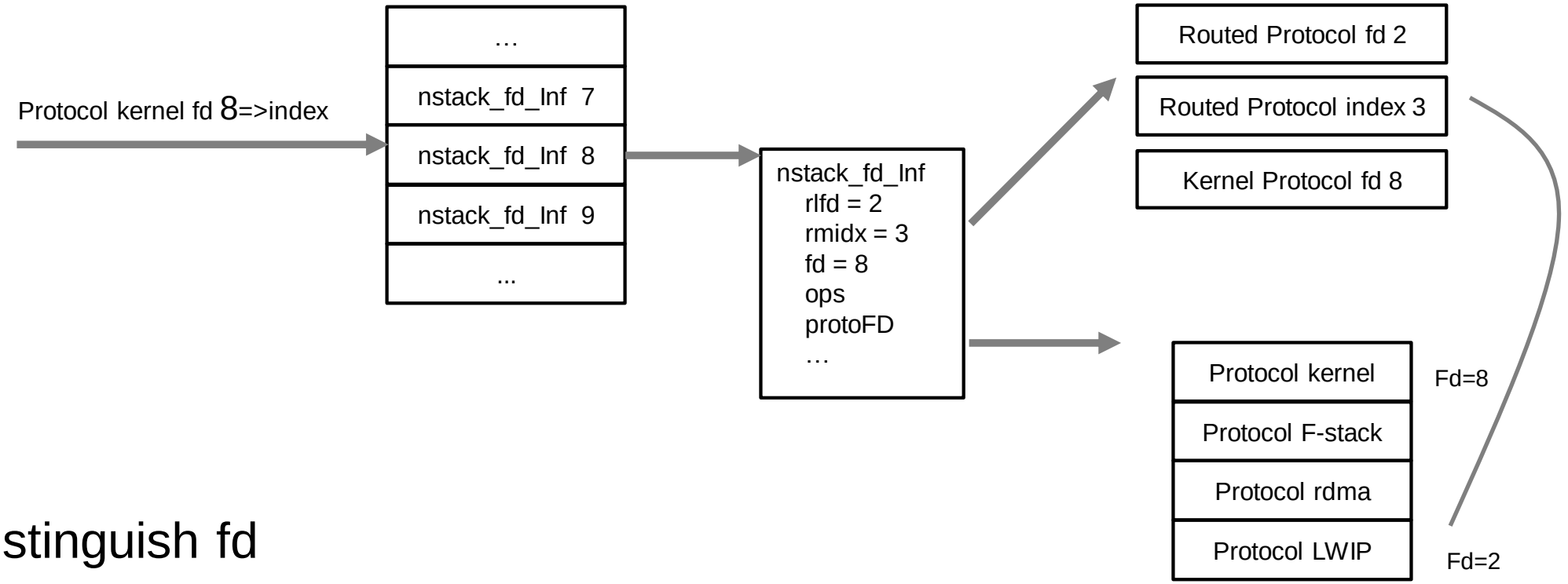
DMM: provide fd management

- Every userspace protocol stack got FD management problem



	DMM + protocol	protocol stack
Modify the APP	no	yes
LD_PRELOAD	yes	yes
Fork	yes	no
Nic	share	grab

DMM: Fd Management



- Distinguish fd
- Monitor app lifecycle
- Support poll/select both on protocol fd and normal fd