



A NIC-offload Implementation of Portals for Quadrics QsNet

Kevin Pedretti & Ron Brightwell
Scalable Computing Systems Department
Sandia National Laboratories
Albuquerque, NM
{ktpedre, rbbrigh}@sandia.gov



Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy's National Nuclear Security Administration
under contract DE-AC04-94AL85000.



Outline

- Background
- Different approaches: Portals vs. QsNet
- Design and performance of Portals on QsNet
- Conclusion and future directions





Motivation

- The Ideal ---->
 - Balanced
 - Linux
 - Light Weight Kernel
 - Purpose Built



- Would like to approximate with COTS components
 - Need a fast Portals implementation
- Gain more NIC offload experience



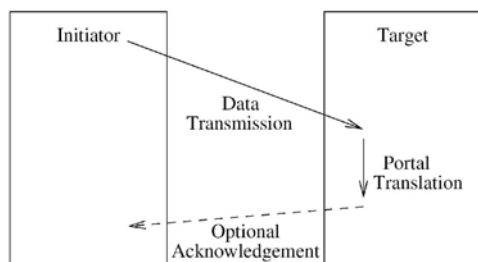
Portals – What is it?

- Sandia/UNM developed API for data movement
 - Portals 2.x had no functional interface (ASCI Red)
 - Portals 3.0 introduced a functional interface (Cplant)
 - Portals 3.3 being used for Red Storm
- Embedded matching (like Tports, MX)
- Designed for offloading
- Designed with light weight kernel in mind
- External users: Lustre

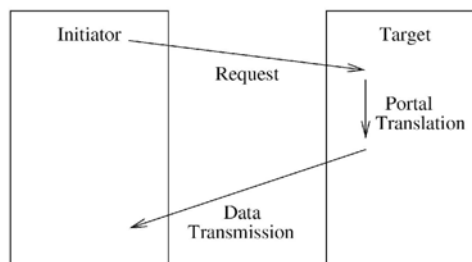




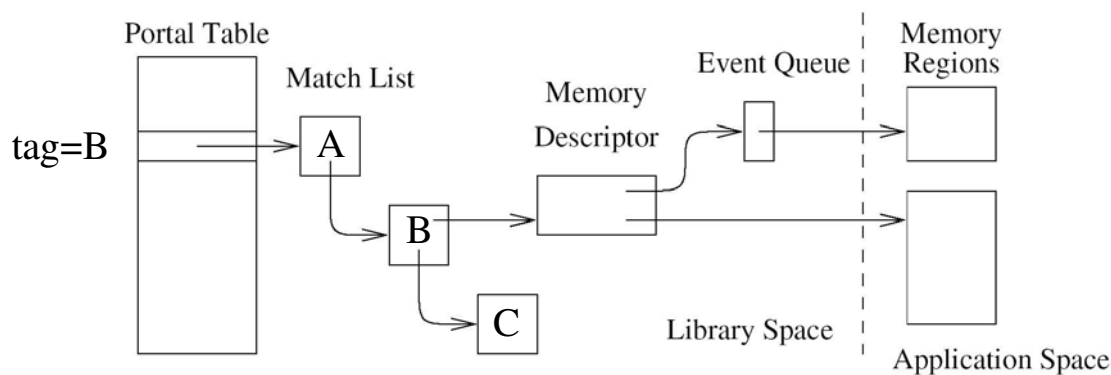
Portals Data Movement



Matching Put

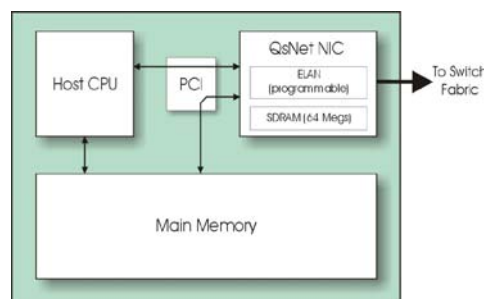


Matching Get

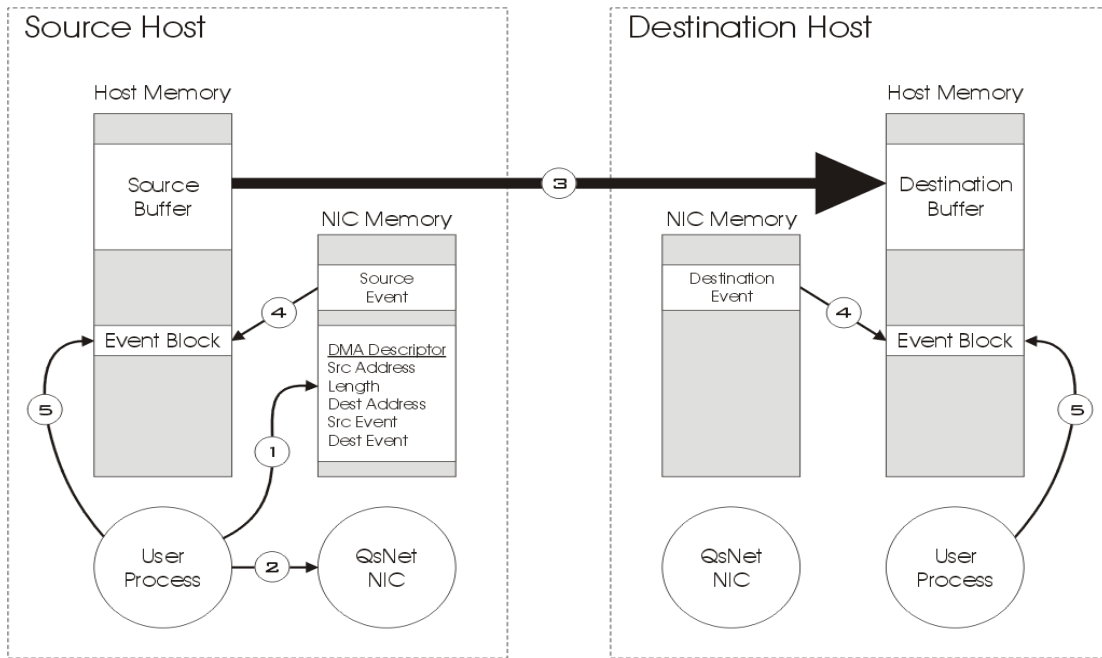


Quadrics QsNet

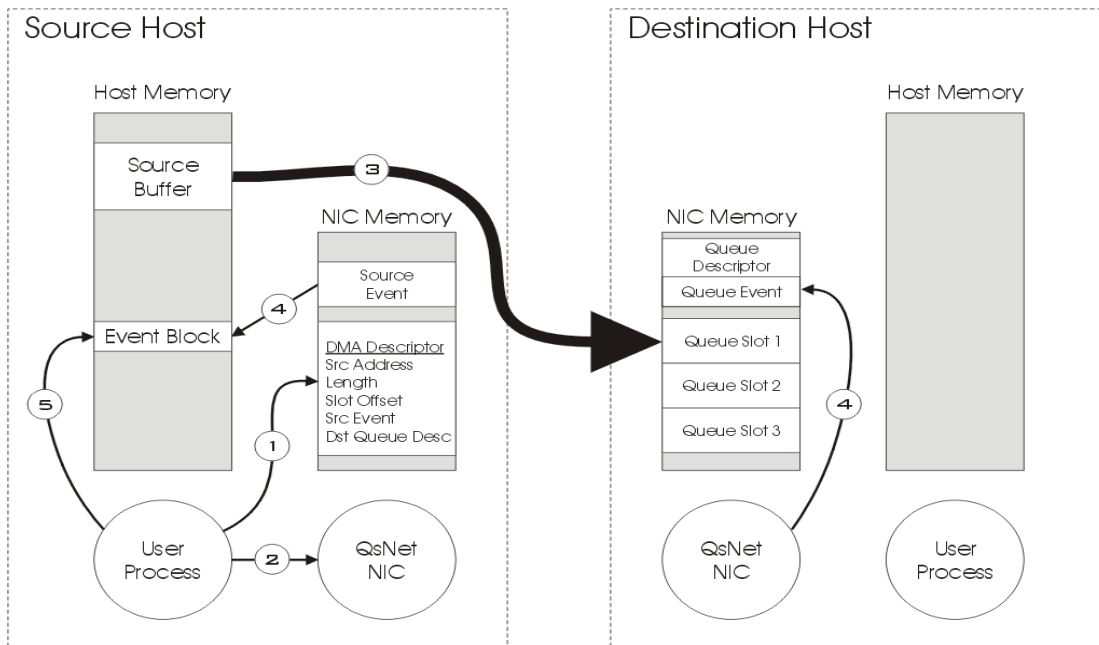
- Elan NIC
 - Programmable processor
 - Memory management unit
 - 64 MB SDRAM
 - RDMA engine
- Supplied with extensive software stack
 - IP stack
 - Kernel comms
 - MPICH, SHMEM
 - Tports (in Libelan)
 - Libelan, Libelan3



QsNet RDMA Operation



QsNet Queued RDMA Operation





Multi-Protocol Support

- Portals: Portals table abstraction
 - Each protocol allocates a set of Portals table entries
 - Example: MPI entries 1-5, Lustre entries 5-8
 - Match entries and memory descriptors then attached
 - Drawback: possible overkill
- QsNet: NIC thread per protocol
 - Hardware primitives provide minimal functionality
 - NIC threads used to implement more complex semantics
 - Drawbacks: complicated hardware, limited portability



User-Level Network Protection

- Portals: Receiver-side checks
 - All network end-points can communicate by default
 - End-points addressed by (nid, pid)
 - File systems and runtime systems need this
 - Access control lists setup to filter unwanted messages
- QsNet: Capabilities
 - Only processes in a parallel application can communicate
 - End-points addressed by rank
 - Drawback: relies on runtime to distribute capabilities





Network Failures

- Portals: Exposes network failures
 - Completion events can specify operation failure
 - File systems and runtime systems need this
- QsNet: Failures not exposed
 - Runtime handles failures (kills application)



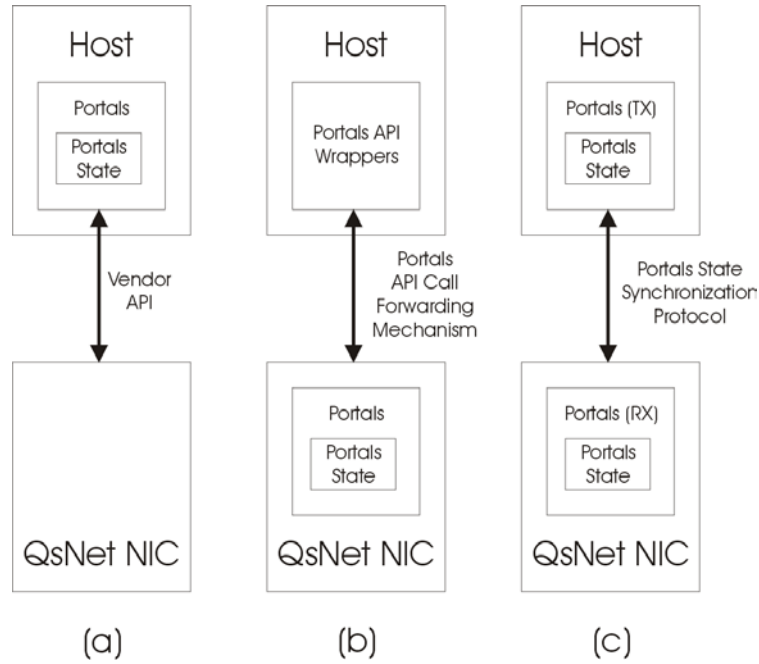
Portals vs. Other APIs

	Portals	Tports	GM	MX	SHMEM
Embedded Matching	Green	Green	Red	Green	Red
Independent processes	Green	Red	Green	Green	Red
Connectionless	Green	Green	Green	Green	Green
Reliable	Green	Green	Green	Green	Green
Ordered	Green	Green	Green	Green	Green
No alignment restrictions	Green	Green	Green	Green	Green
Independent MPI progress	Green	Green	Red	Green	Red
Allows overlap	Green	Green	Red	Green	Green
Asynchronous completion	Green	Green	Red	Green	Green
One sided operations	Green	Green	Green	Red	Green
Exposes failures	Green	Red	Green	Red	Red
Allows reviews to be canceled	Green	Green	Red	Green	Red
Collective operations	Red	*	Red	Green	Green
Embeds unexpected message handling	Red	Green	Red	Green	N/A

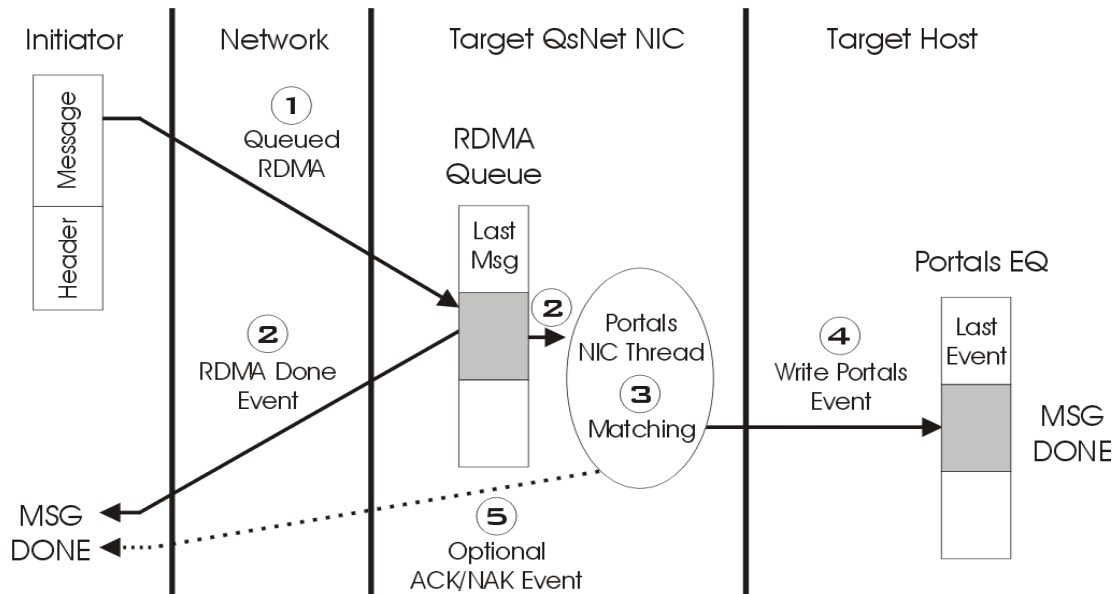




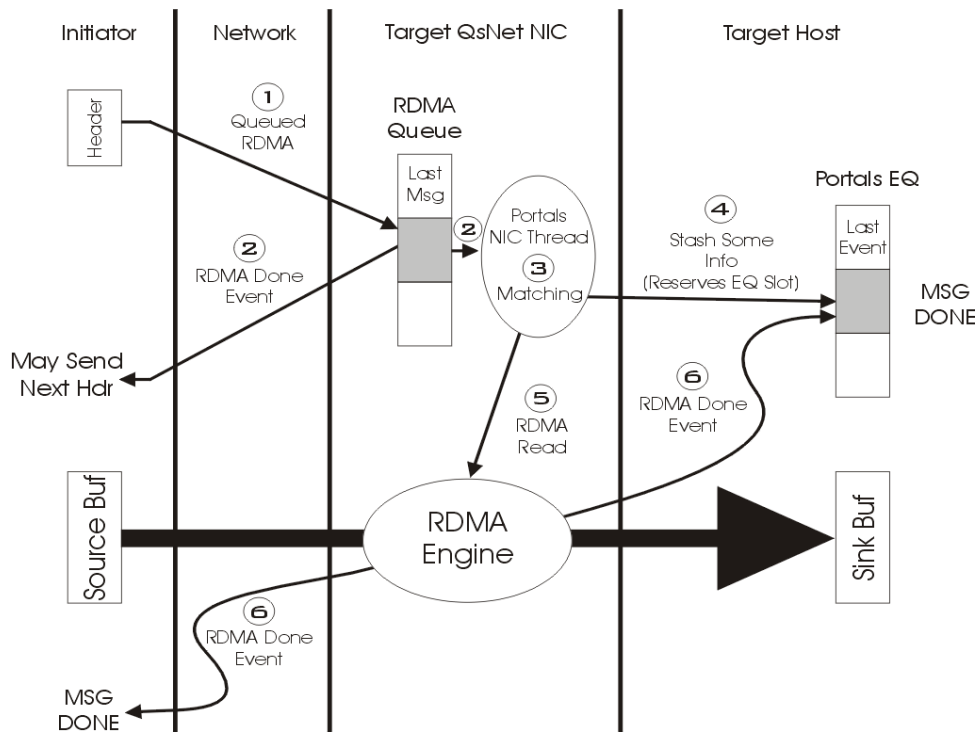
Design Options



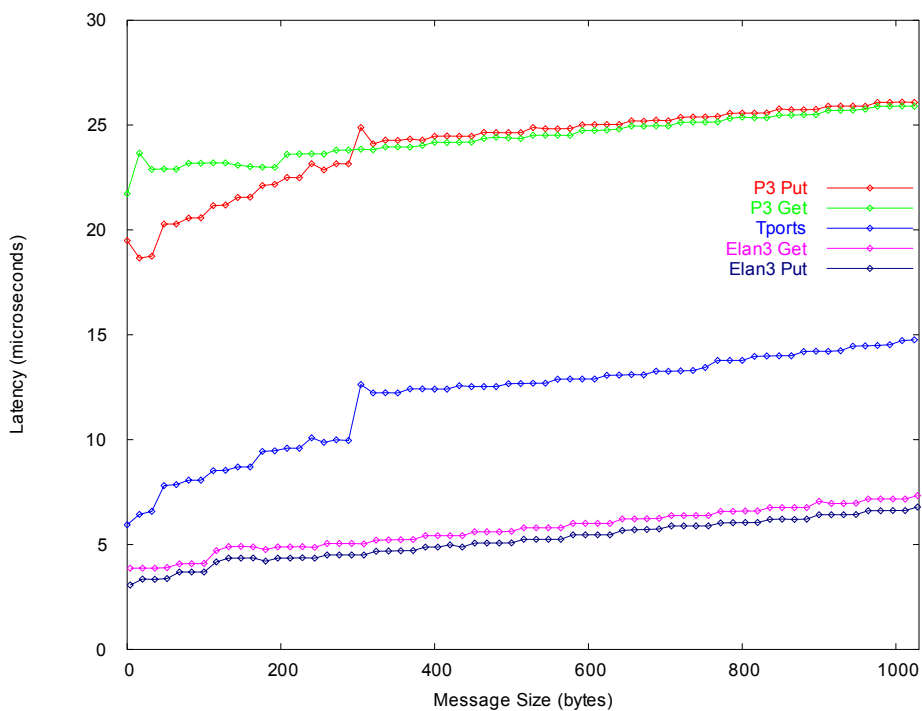
Small Message Protocol



Large Message Protocol

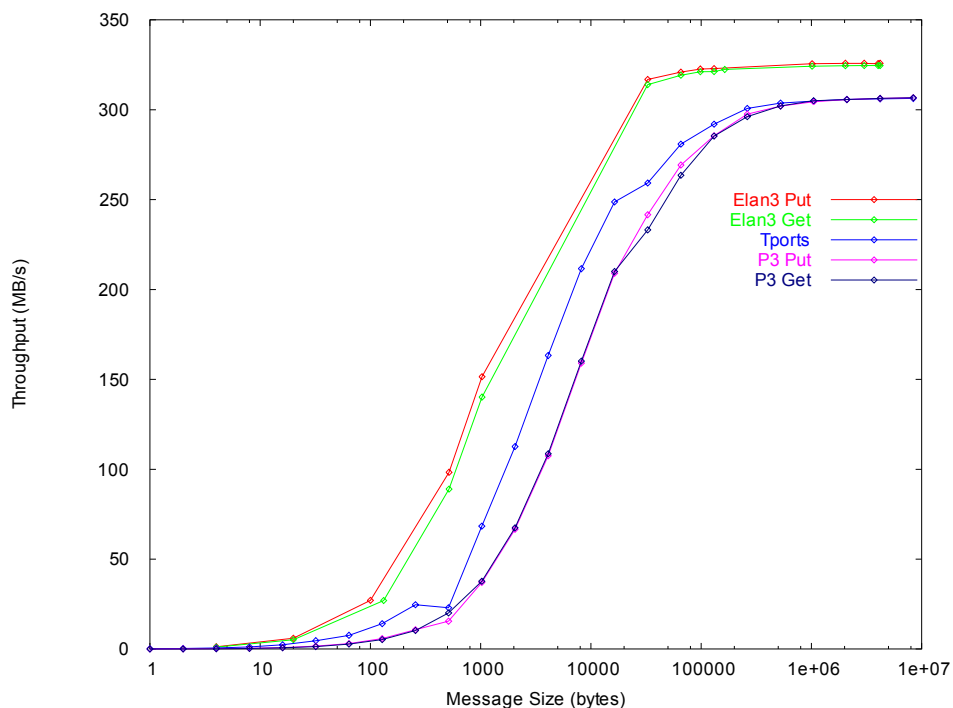


Ping-Pong Latency





Ping-Pong Bandwidth



Ping-Pong Host Level Timing

	Portals	Tports
Post receive for pong	6.4	1.7
Send ping	2.2	1.0
Wait for send to complete	2.8	0.1
Wait for pong to arrive	24.7	8.5
Total:	36.1	11.3

(microseconds)

- Post receive takes two API calls (ME Attach, MD Attach)
 - Implemented separately (2x PCI traffic)
- Portals events are large (128 bytes)
 - API copies events by value





Ping-pong NIC Thread Timing

	Portals	Tports
Take lock	0.2	0.1
Inspect header	0.4	0.1
Match	1.4	0.3
Hosekeeping	0.7	0.2
Write completion event	0.7	0.5
Unlink matched buffer	1.3	0.1
Drop lock	0.1	0.1
Total:	4.9	1.5

(microseconds)

- Portals matching more complex
- Supporting conditional unlinking adds overhead
 - Tports buffers are always unlinked, allows optimization



Conclusion

- Portals and QsNet take different approaches
 - Portals “all-in-one”, QsNet many specialized
 - Portals useful for file system, runtime
- Portals is implementable on QsNet
 - Not a perfect match
 - Rich semantics of Portals costs performance





Next Steps

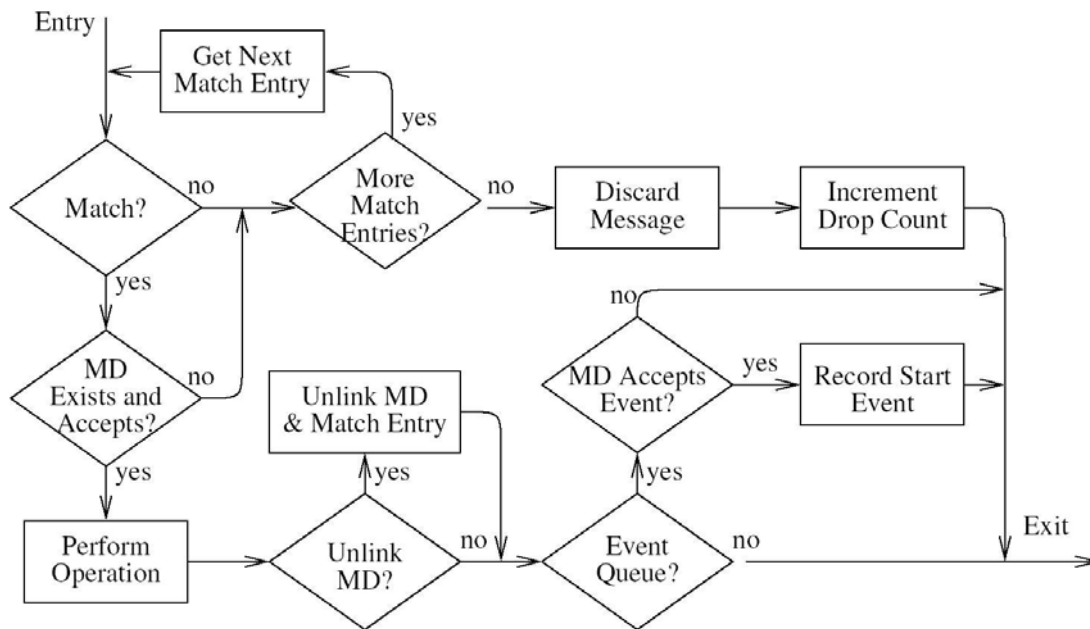
- QsNetII (Elan4) looks promising
 - PCI-X based
 - Faster thread processor (2x)
 - Possibly write completion events into queue
- 64 node QsNetII development cluster in house
- Possible Portals changes



Backup Slides



Portals Receive Processing



Main Portals API Functions

- Match Entries (MEs)
 - PtlMEAttach, PtlMEInsert, PtlMEUnlink
- Memory Descriptors (MDs)
 - PtlMDAttach, PtlMDBind, PtlMDUnlink
- Event Queues (EQs)
 - PtlEQAlloc, PtlEQFree
- Data Movement
 - PtlPut, PtlGet



Main Portals API Structures

```
typedef struct {
    void *          start;
    ptl_size_t     length;
    int            threshold;
    unsigned int   max_size;
    unsigned int   options;
    void *         user_ptr;
    ptl_handle_eq_t eq_handle;
} ptl_md_t;
```



Event Queue

```
typedef struct {
    ptl_event_kind_t type;
    ptl_process_id_t initiator;
    ptl_uid_t        uid;
    ptl_jid_t        jid;
    ptl_pt_index_t  pt_index;
    ptl_match_bits_t match_bits;
    ptl_size_t      rlength;
    ptl_size_t      mlength;
    ptl_size_t      offset;
    ptl_handle_md_t md_handle;
    ptl_md_t        md;
    ptl_hdr_data_t  hdr_data;
    ptl_seq_t       link;
    ptl_ni_fail_t   ni_fail_type;
    ptl_seq_t       sequence;
} ptl_event_t;
```

```
int PtlMEAttach(ni, portal, match_id,
               match_bits, ignore_bits,
               unlink, position, &me_handle);
```

```
int PtlMDAttach(me_handle, md, unlink_op, &md_handle);
```



QsNet RDMA Descriptor

```
typedef volatile struct e3_dma {
    E3_DmaType      dma_u;
    E3_uint32       dma_size;
    E3_Addr         dma_source;
    E3_Addr         dma_dest;
    E3_Addr         dma_destEvent;
    E3_CookieVproc dma_destCookieProc;
    E3_Addr         dma_src_Event;
    E3_CookieVproc dma_srcCookieProc;
} E3_DMA;
```

```
elan3_initevent_blk(s dramaddr_t);
elan3_waitevent(s dramaddr_t);
elan3_putdma(E3_DMA);
elan3_getdma(E3_DMA);
```



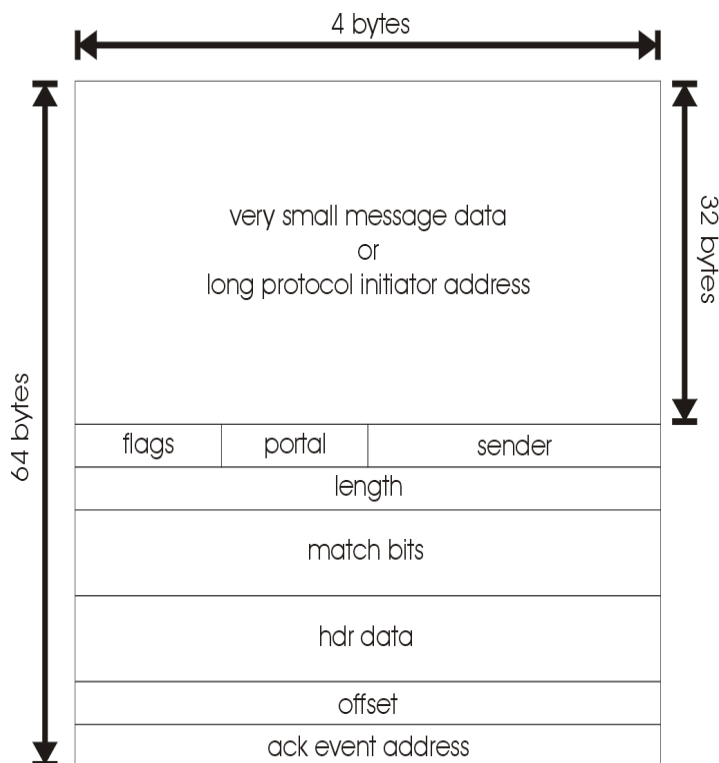


Matching Semantics

- Portals: Embedded Complex Matching Semantics
 - Match entry list traversal general enough for all network services
 - Complex memory descriptors
 - Can be persistent (e.g., threshold = infinity)
 - Offset management (local vs. remote offset management)
 - Drawback: possible overkill
- QsNet: NIC Threads Perform Matching
 - Tports has simple matching semantics
 - Drawbacks: hardware complexity, portability



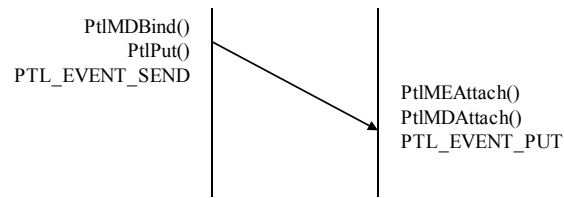
Portals Header



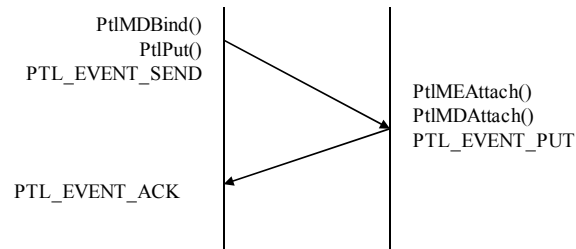


MPI over Portals

Standard Short Send

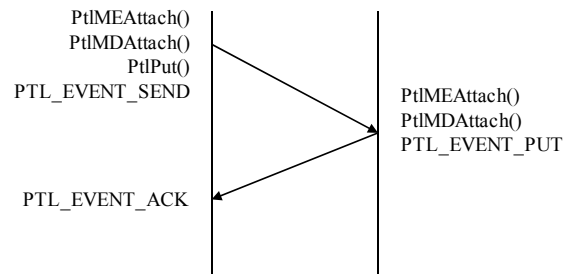


Synchronous Short Send



MPI Over Portals (cont.)

Long Send Pre-Posted



Long Send Unexpected

