

- Network Interface
 - usage models
 - design parameters

- issues
 - latency – important for small messages
 - bandwidth – important for large messages
 - mux/demux
 - buffering
 - notification
 - DMA, VA
 - protocol acceleration

Active Messages

- communication mechanism maps to h/w model:
 - launch message on send, generate interrupt on receive
- messages contain handler pointer (and arguments) for processing to be executed on receive
 - restrictions: non-blocking, to completion, only 1 reply...
 - handler – needs to extract data from network, and integrate it into ongoing computation
 - handler executes on current stack
 - data placed either in user-provided memory, or immediate reply (no buffering)
 - large messages via put and get (write and read)

Remote Queues

- separate queueing of msgs from handler invocation
 - flexibility to implement polling (atomicity cheap), selective interrupts
 - demux messages to appropriate queue, handle immediately system/control msgs
 - queues in host or NIC memory (depending on what's polled where)
 - small msgs in queues, large msgs – address+offset
 - notification mechanism for DMA completion – post a message into appropriate queue

Remote Memory

- ultimate destination in memory
 - addresses can be accessed via bus
- separate data and control
 - data in remote memory, read/write/cas
 - control via notify mechanisms
 - trust assumed, not feasible in general

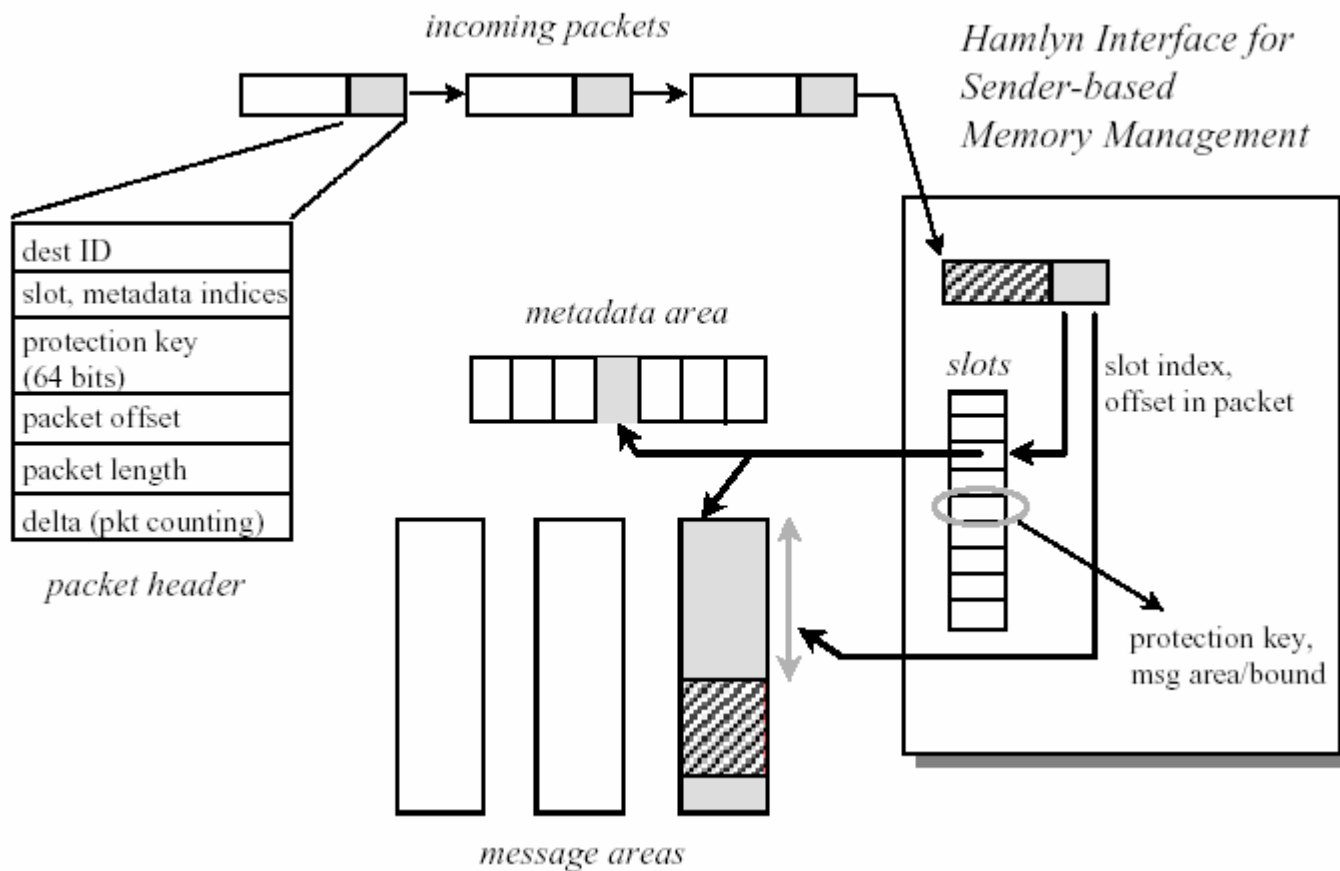
Hybrid Deposit

- source-based
 - random access, cost isolation, synchronized access
- destination-based
 - location implicit, destination manages; general
- hybrid
 - function of sender and destination state

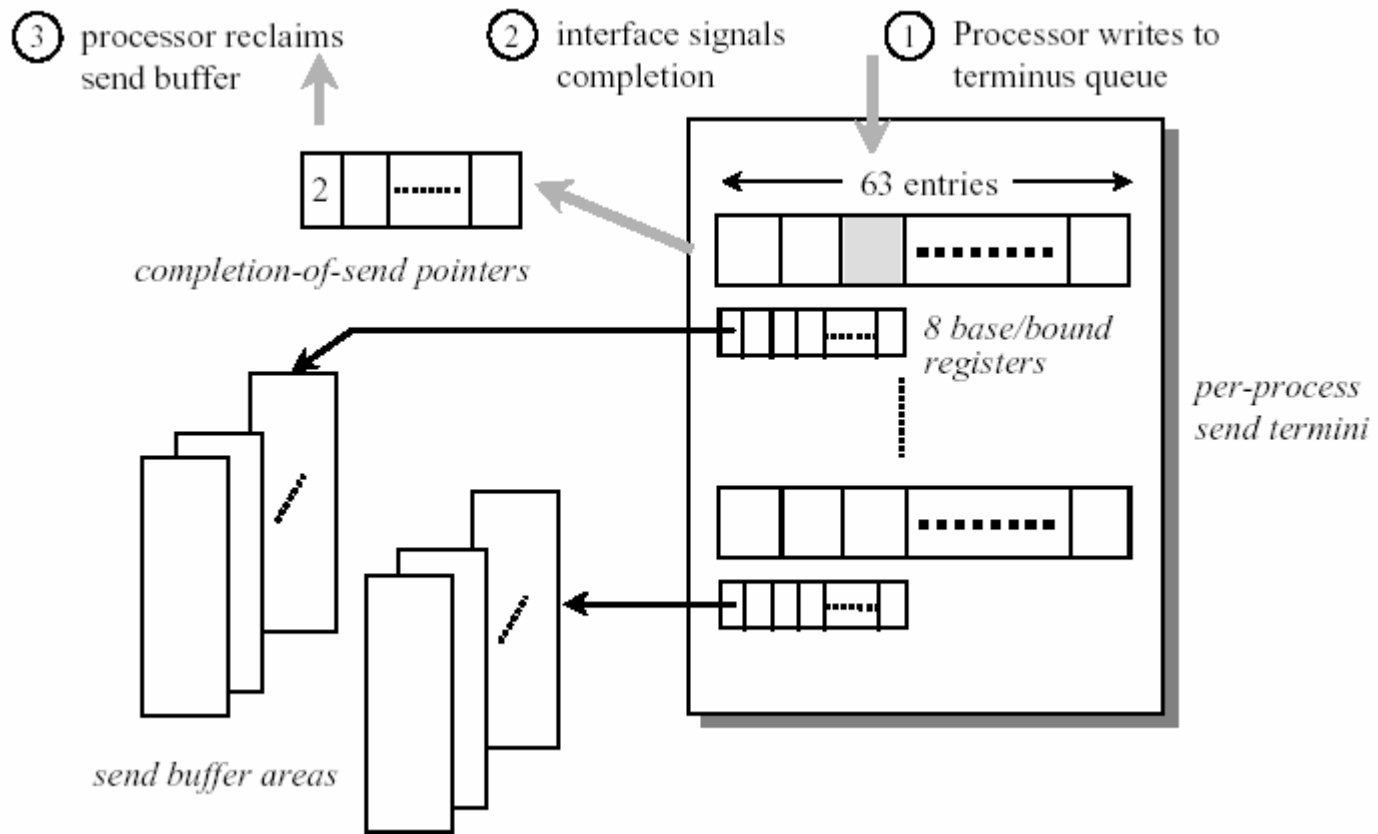
- data movement
 - sender-based (Hamlyn): trust, knowledge...
 - receiver-based (Active Messages): interrupt computation
 - sender&receiver-based (Hybrid Deposit)

Hamlyn I/F: basic idea

- know final destination in receiving host mm before sending a message.
- pinned pages – message areas, referred through *slots*
- sender has workQ (64 entries) for requests
 - short msgs direct IO
 - long msgs DMA from (8) send buffers
- for each msg, entry in metadata area in rcvr VAspace

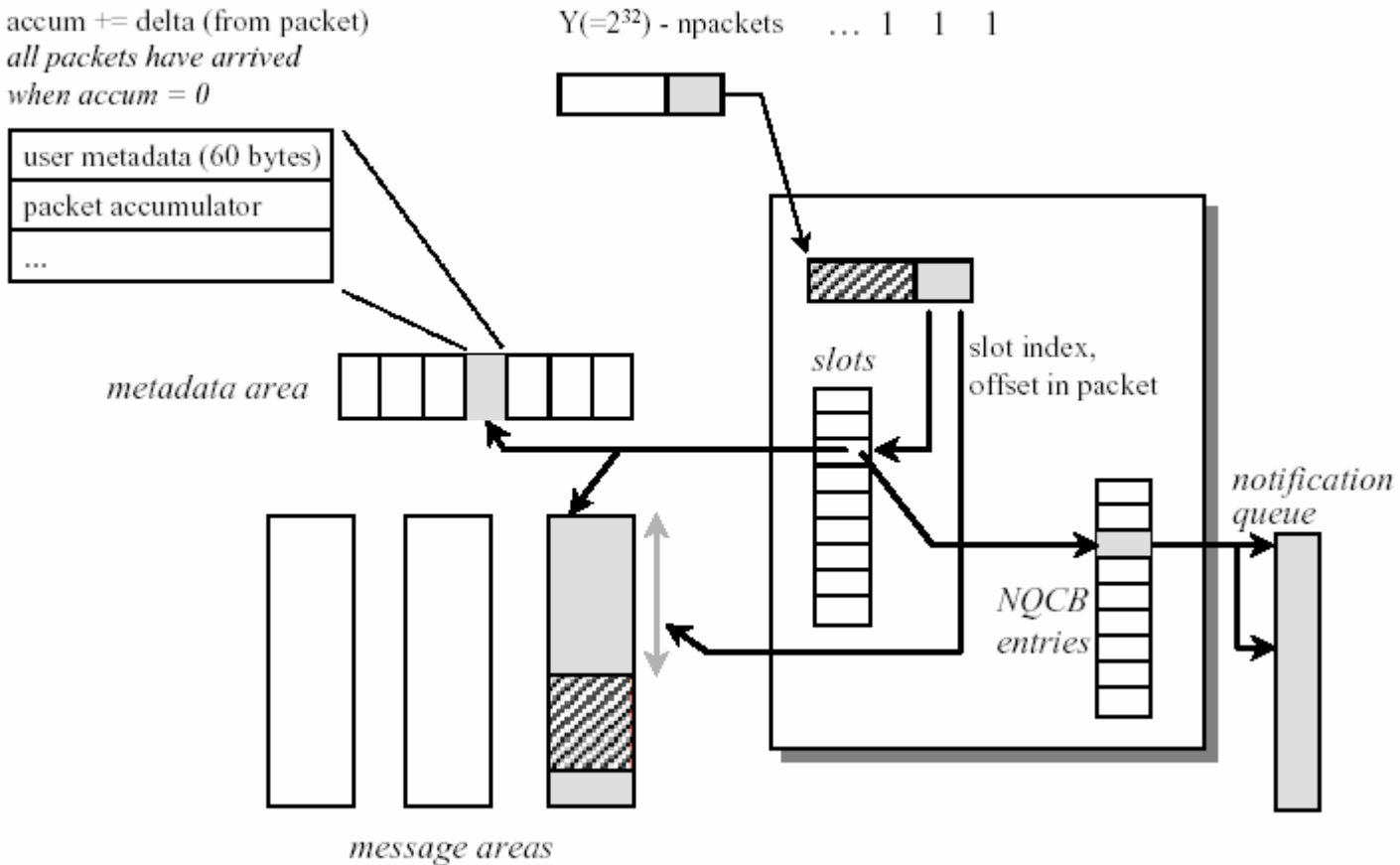


sending a message



receiving a message

accum += delta (from packet)
all packets have arrived
when accum = 0



	AM	RQ	RM
mux/demux	to user process	to queue	address provided
notification	polling/interrupt, multithreaded	buffering,	optional
buffering	avoided	implicit	no
DMA/VA	receiver determines address	small vs. large messages	sender picks address
protocol accel.	handler in h/w	enqueue	yes

- flexibility of interface
 - ActiveMessages – most general
 - RemoteMemory – specialized, cannot expect arbitrary communicating endpoints to share address space knowledge
 - RemoteQueues – share endpoint id, queues of messages...
 - HybridDeposit – use queues to share addresses

Alefiwe

- unified interface for shared memory and message passing communication.
 - large scale shared memory multiprocessor – global virtual address space
 - cache coherency
 - underneath mesh with wormhole routing, assumed reliable!

- so you need message passing interface

- what should it provide?
 - direct access register -> NI -> register
 - special instructions
 - direct access Mm -> NI -> Mm (possible multiple locations)
 - DMA engine
 - gather/scatter
 - ability of user-level access to interface
 - protection – wrt user operations
 - atomic operation – user cannot disable ints during msg handling
 - ability to implement arbitrary mechanisms through interrupt handler code

⇒ on send describe msg, then launch it;

⇒ on receive (interrupt or poll) inspect the msg, then discard, pass to registers, or storeback to memory

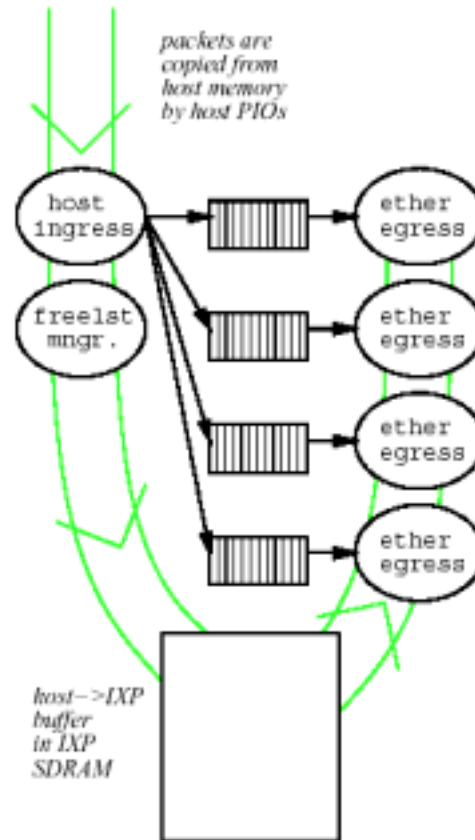
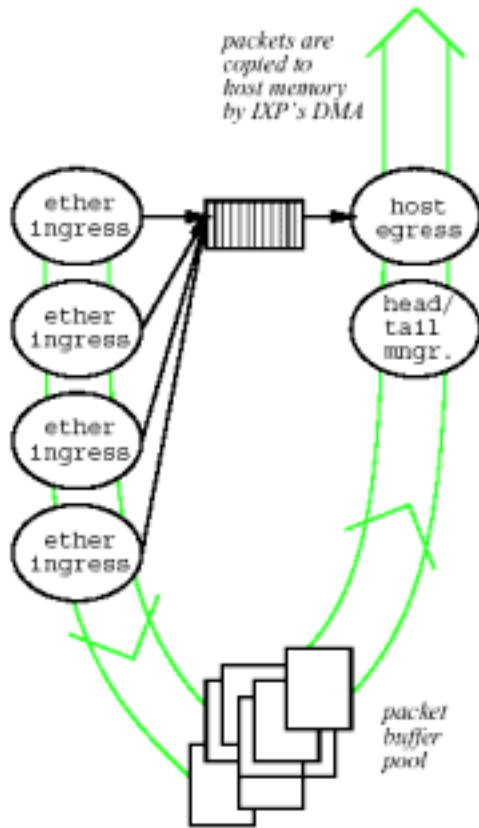
⇒ user, system, cache coherency protocol – all may use messages simultaneously...

- message descriptors
 - 32b header; major opcode for system, user, coherence-protocol; up to 8x64bit long
- on send
 - software writes descriptor from registers, operands and sets addresses and lengths
 - CMMU
 - collects data, commits descriptor to network
 - has space-avail info, which is checked when descriptor is written
- on receive
 - CMMU runs handlers, keeps window into data which can be loaded into registers, can issue storeback operation on all or part of remainder data, even if not avail at i/f
- blocking, polling, and interrupts supported

given that you have a shared memory multiprocessor, can you make the two coexist?

- local vs. global coherence
 - software can send coherence packets -> no hardware limits on directory
- prevent deadlock with high-availability interrupts
- restrictions on handlers wrt global accesses

ixp as a NIC example



Offload vs. Onload

- Is it beneficial? Why? Why not?
- What should be offloaded?
- What shouldn't be offloaded?
- Do multi-core systems change the picture?
- What other capabilities are needed from a NIC?
 - already named few...
 - others...