Context & motivations

- Internet keeps growing, in terms of
  - Number of users
  - Links bandwidths (xDSL,10GEth)

⇒ Internet servers must face increasing traffics

- CPU speed does not increase as fast as network speed

⇒ Internet servers must scale with the #CPUs

Outline

- Background
- Analysis
- KNET network subsystem
- Results

BSD-style networking subsystem

Inbound pkt flow

- CPU takes a rcv interrupt
- ISR enqueues the pkt(s) in the backlog queue and schedules a bottom-half
- The bottom-half pulls pkts off the backlog and takes them through the stack

☞Linux works this way

☞Optimization: "Interrupt coalescing"

NAPI networking subsystem

Inbound pkt flow

- CPU takes a rcv interrupt
- ISR deactivates RX intr, enqueues the device, and schedules a bottom-half
- The bottom-half dequeues the device, pulls the first pkt off the RX DMA queue, takes it through the stack, and repeats this while it finds new pkts in the RX DMA queue
- The bottom-half eventually reactivates RX intr

☞Linux-2.4.20 and above support NAPI

Linux Softnet for SMP

- Three types of BH: old-style, tasklet, softirq
- Incoming pkts are processed in the context of a softirq (NET_RX_SOFTIRQ)
- Each interrupt vector is distributed among CPUs in a round-robin fashion

$\Rightarrow$ Load-balancing

$\Rightarrow$ Parallelism

NAPI Pros and Cons

- Pros:
  - Rationale: not prone to Receive Livelock [Mogul]
  - "Adaptative Interrupt Coalescing"
  - No TCP segments reordering (fast-path)
  - Better data & instr. Locality
- Cons:
  - No parallelism for traffic coming in from a single network interface

Analysis (1)

- Latency + Zero-copy sends ⇒ most data transfers occur due to received ACKs

⇒ Parallel receives are necessary

- No packet reordering
- Less lock contentions
- Scalability

⇒ Connection-level parallelism (classification)

Analysis (2)

- NAPI Benefits
  - Adaptative Interrupt Coalescing

$\Rightarrow$ Classify packets out of the kernel, in the Network Interface Adapter

KNET

Design

- Per-processor worker threads
- Per-processor RX queues
- Each thread is attached to a particular CPU
- Incoming pkts are classified by the NIC
- All packets of the same connection are processed by the same CPU

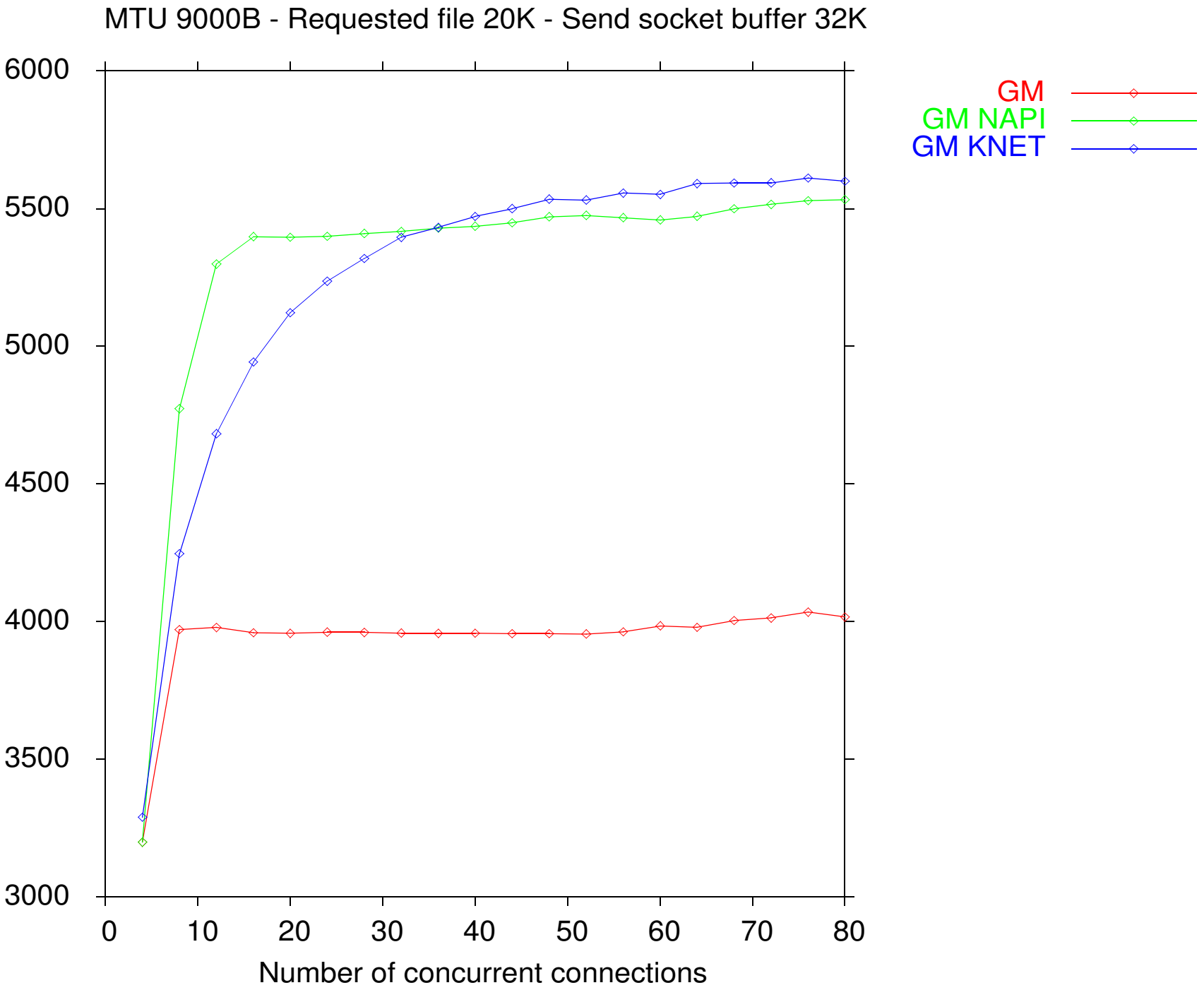
KNET

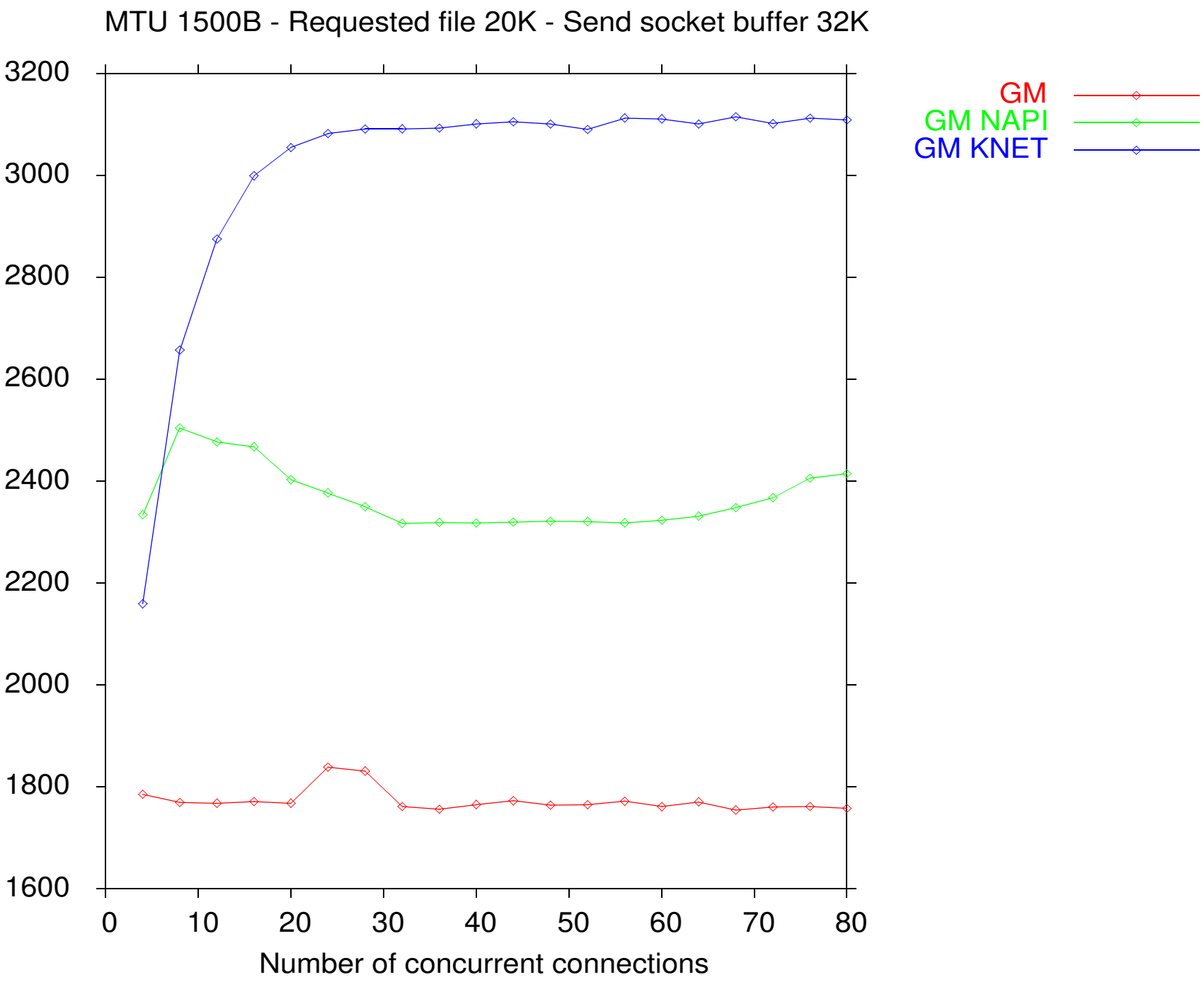
Implementation

- Linux-2.4.x module
  - Per-CPU kernel threads
  - Events loop
  - Define a new interface to drivers
- Modifications to Myricom/GM NIC
  - Driver (implement new iface, per-CPU RX queue)
  - Firmware: (classification)

Experimental environment

- Four HTTP traffic generators
  - ServerWorks-based bi-PIII 600Mhz/256MB RAM
  - Modified version of Sclient traffic generator
- One HTTP web server
  - Webfs (event-driven, multithreaded, sendfile)
  - ServerWorks-based quad-PIII 550Mhz/512MB RAM
- Myrinet 2x2Gbps - L9 200Mhz


MTU 9000B - Requested file 20K - Send socket buffer 32K
6000
5500
5000
4500
4000
3500
3000
0
10
20
30
40
50
60
70
80
Number of concurrent connections
GM
GM NAPI
GM KNET


MTU 1500B - Requested file 20K - Send socket buffer 32K
3200
3000
2800
2600
2400
2200
2000
1800
1600
0
10
20
30
40
50
60
70
80
Number of concurrent connections
GM
GM NAPI
GM KNET

Bibliography

- [Mogul, 1995] Eliminating Receive Livelock in an Interrupt-driven Kernel
- [Hadi Salim, 2001] Beyond Softnet