SpliceNP: A TCP Splicer using a Network Processor

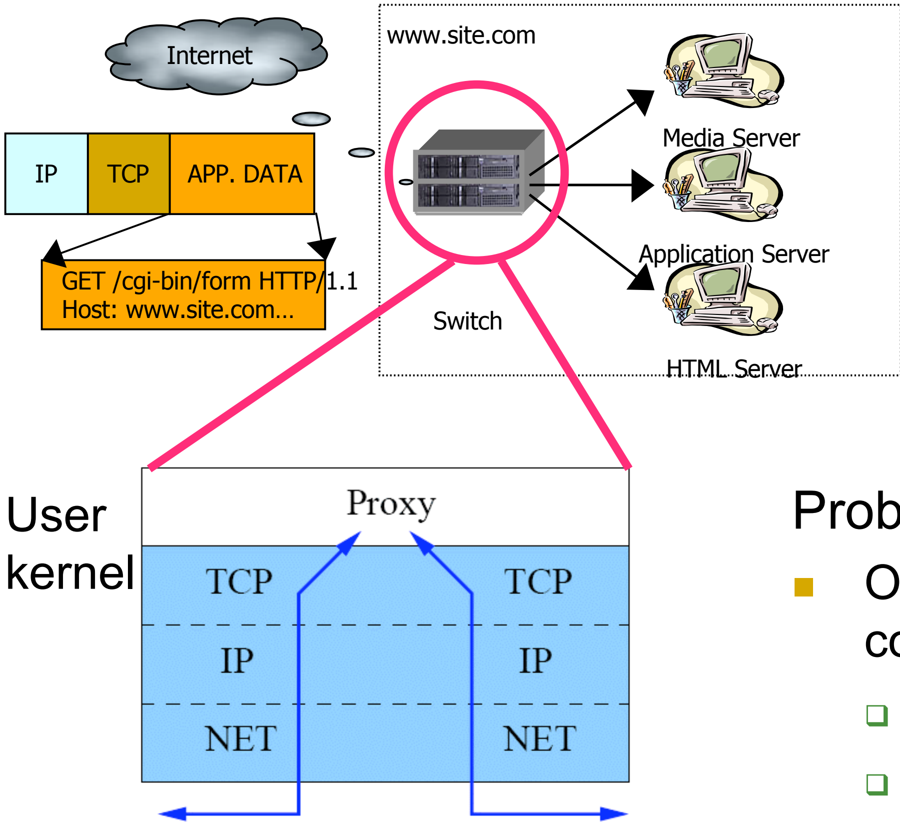
Li Zhao[+], Yan Luo[*], Laxmi Bhuyan
University of California Riverside
Ravi Iyer
Intel Corporation

[+] Now with Intel Corporation
[*] Now with University of Massachusetts Lowell

Outline

- Background and Motivation
- Design and Implementation
- Performance Evaluation
- Conclusions

Background



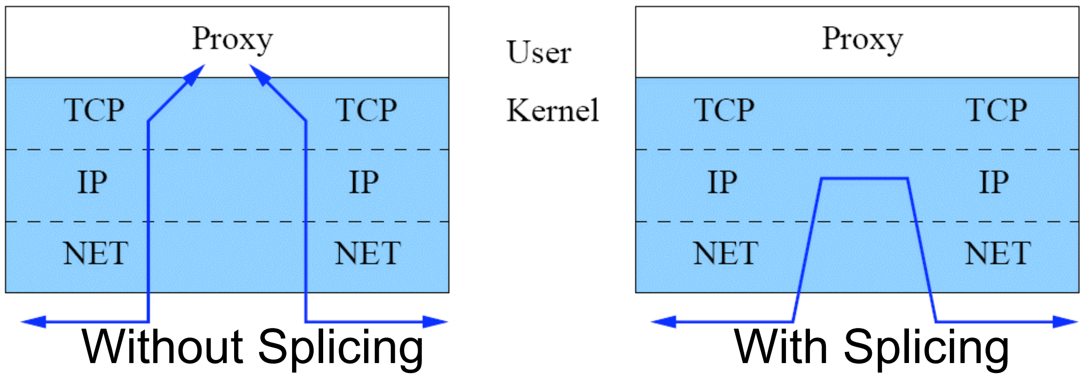
HTTP proxy

- Front-end of a web cluster, one VIP
- Route packets based on layer 5 information
  - Examine application data in addition to IP& TCP

Problem with HTTP Proxy

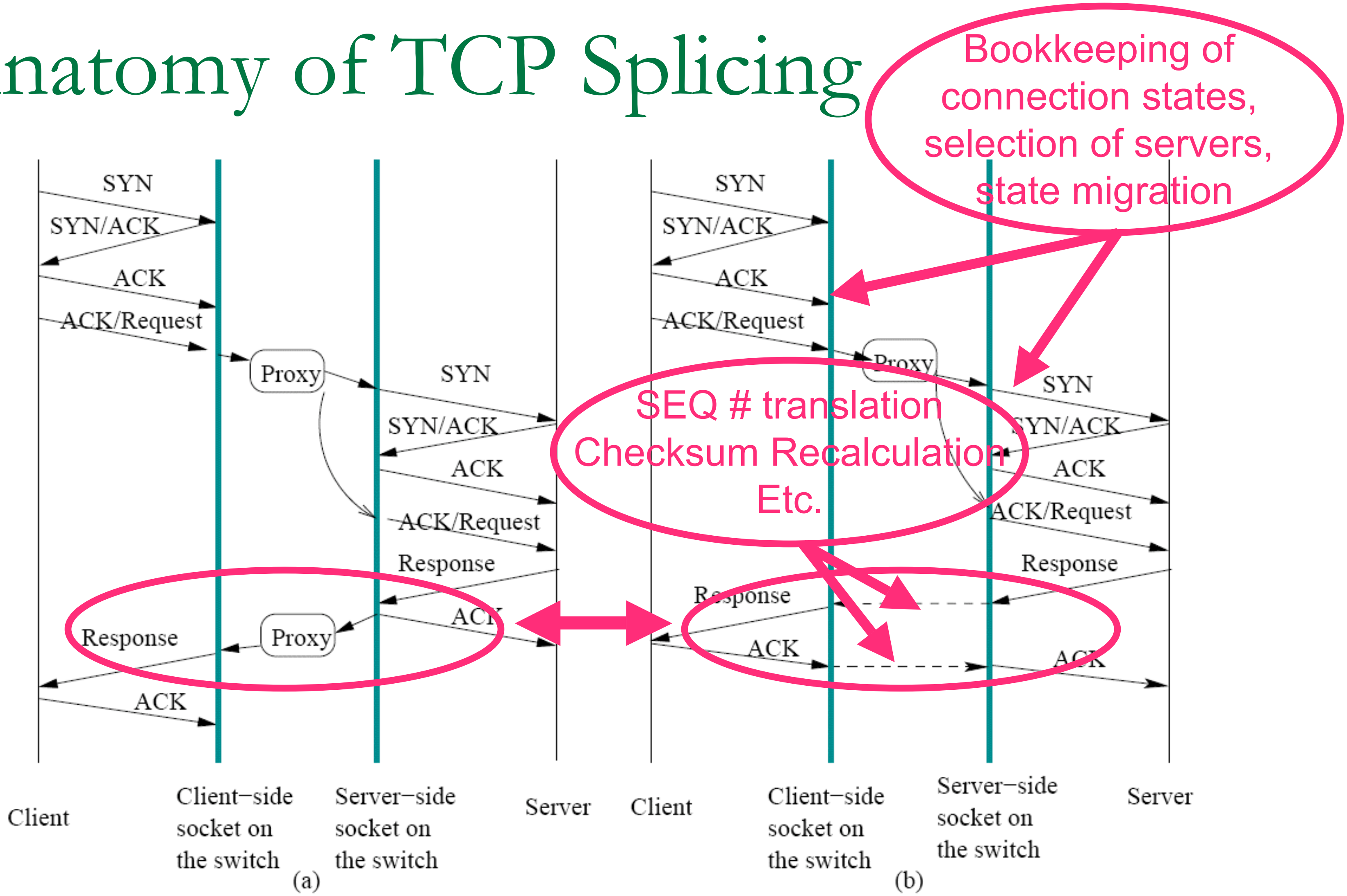
- Overhead to copy data between two connections is high
  - Data goes up through protocol stack
  - Data is copied from kernel space to user space
  - Data is copied back to kernel and goes down through protocol stack

TCP Splicing



- A technique to splice two TCP connections inside the kernel
- Data relaying between the two connections can be speeded up
- Can be used to speed up layer-5 switching, web proxy and firewall running in the user space
- Example: content-aware switch

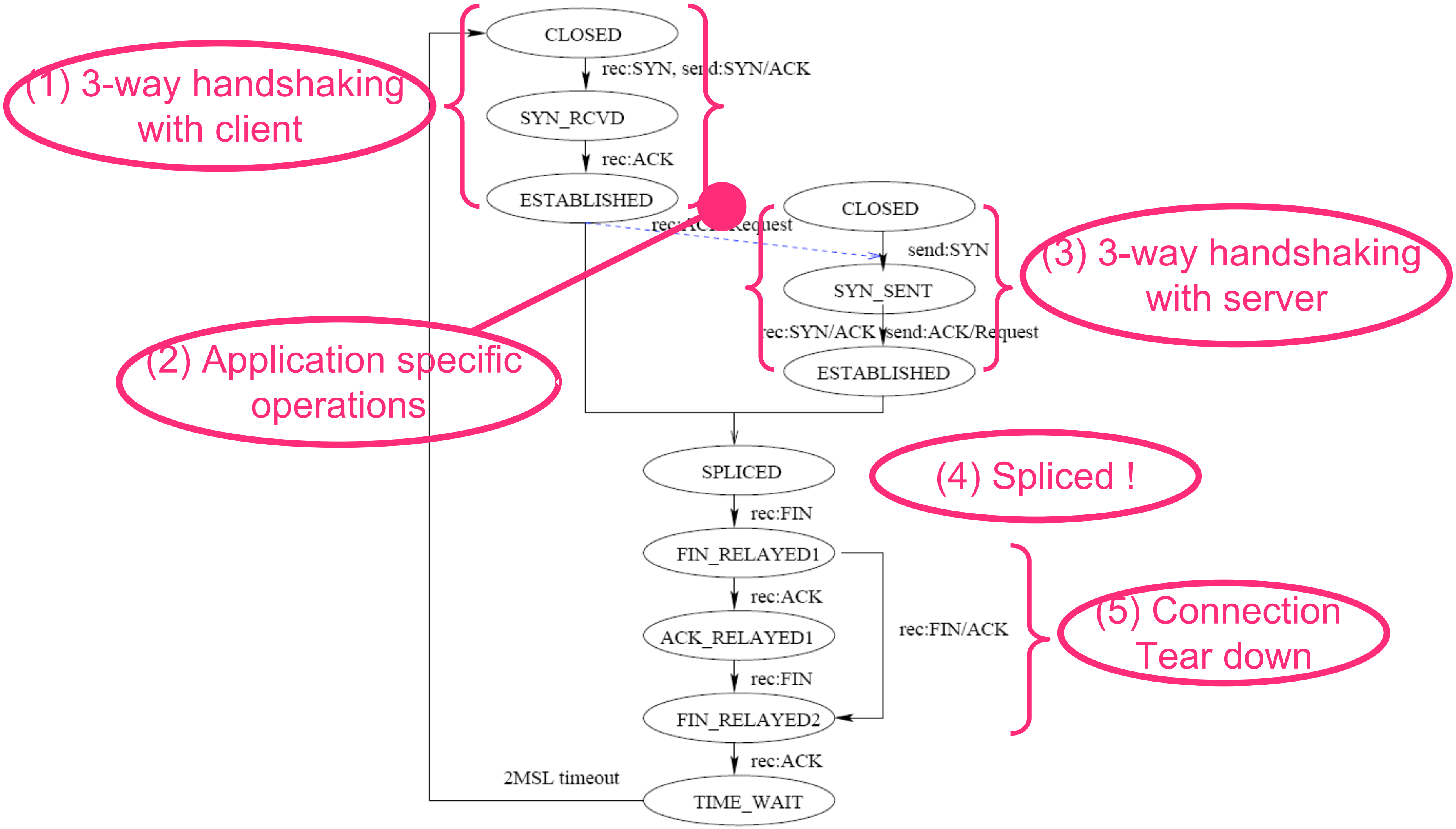
Anatomy of TCP Splicing



Without TCP Splicing

With TCP Splicing

State Transition in TCP Splicing



Outline

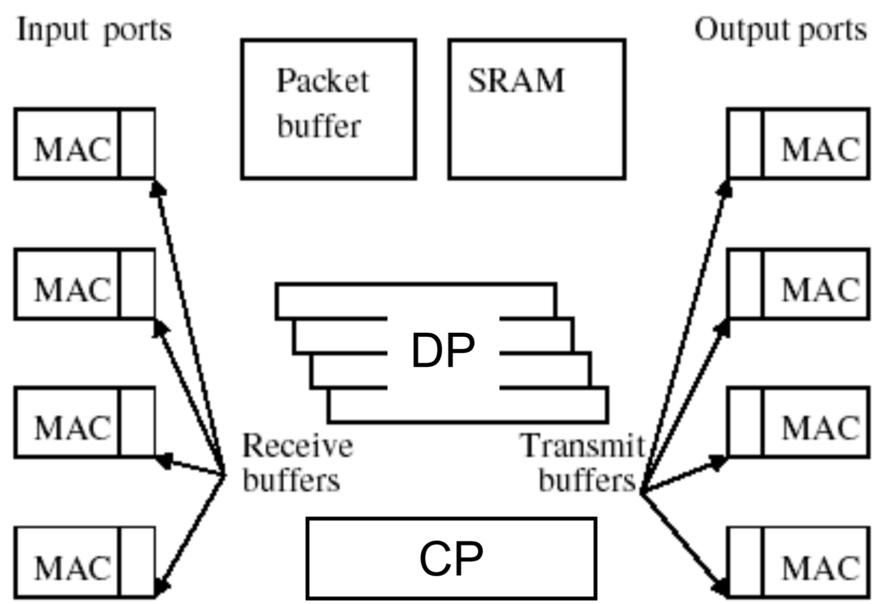
- Background and Motivation
- Design and Implementation
  - Design Options
  - Implementation on IXP2400
- Performance Evaluation
- Conclusions

Processing Elements for TCP Splicing

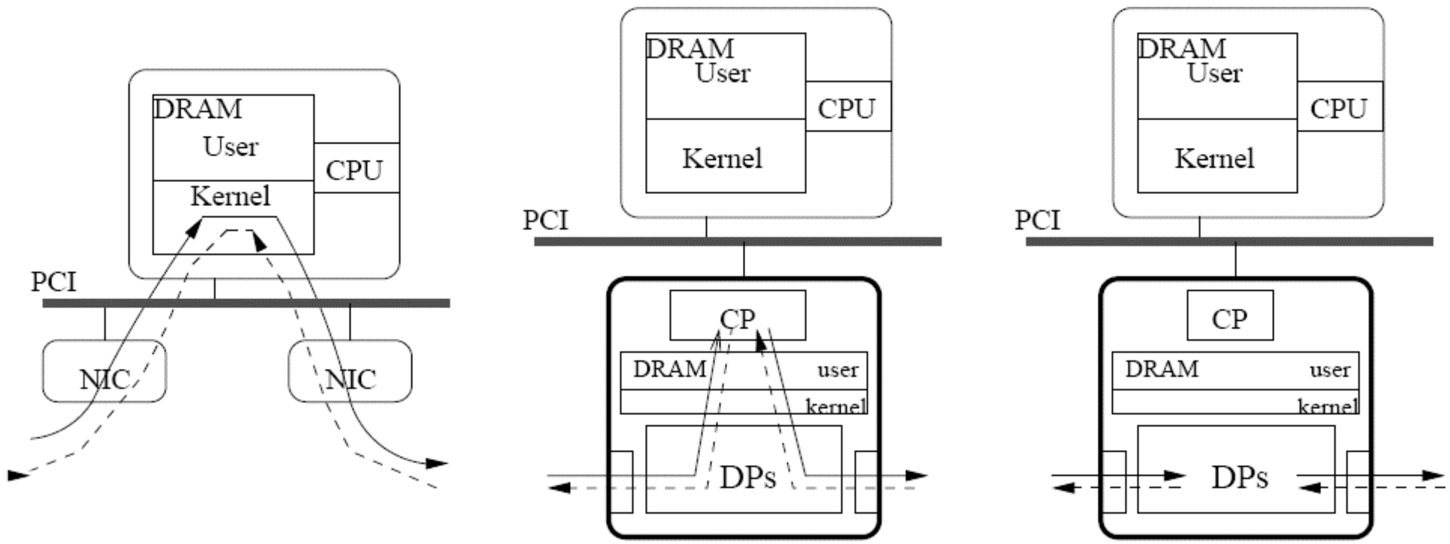
- ASIC (Application Specific Integrated Circuit)
  - High processing capacity
  - Long time to develop
  - Lack the flexibility
- GP (General-purpose Processor)
  - Programmable
  - Overheads:
    - interrupt, moving packets over PCI bus, ISA not optimized
- NP (Network Processor)
  - optimized ISA for packet processing, multiprocessing and multithreading → high performance
  - Programmable

Background on NP

- Architecture
  - Control processor (CP): embedded processor, maintain control information, process exception packets
  - Data processors (DPs): tuned specifically for packet processing
  - Communicate through shared SRAM and DRAM
- NP operation
  - Packet arrives in receive buffer
  - Packet Processing by CP or DPs
  - Transfer the packet onto wire after processing



Design Options



(a) (b) (c)

- Option (a): GP-based (Linux-based) switch
  - Overhead of moving data across PCI bus
- Option (b): CP creates and splices connections, DPs process packets sent after splicing
  - Connection setup & splicing is more complex than data forwarding
  - However, control packets need to be passed through DRAM queues
- Option (c): DPs handle connection setup, splicing & forwarding

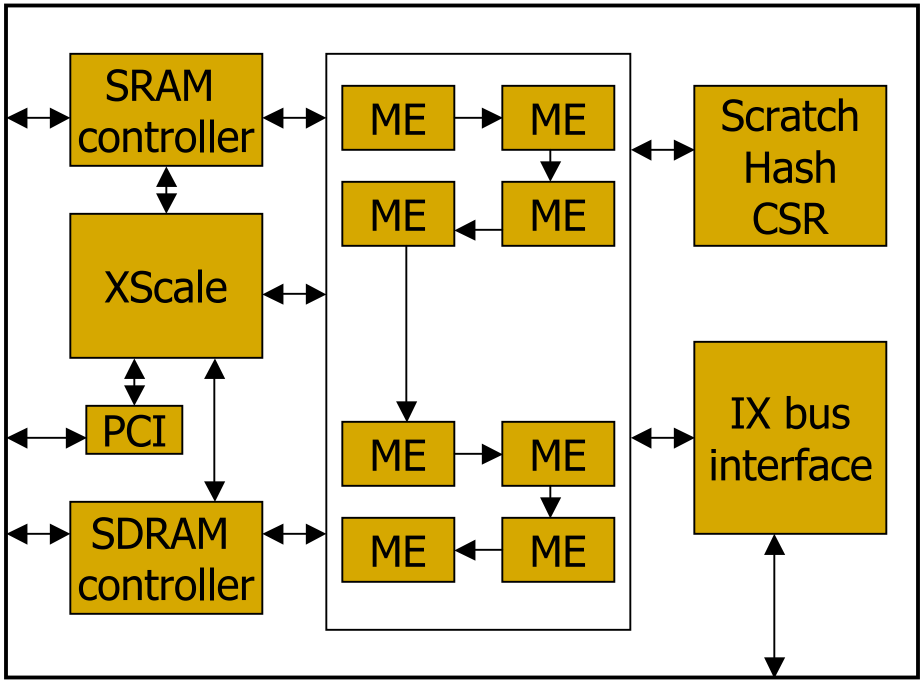
Outline

- Background and Motivation
- Design and Implementation
  - Design Options
  - Implementation on IXP2400
- Performance Evaluation
- Conclusions

IXP 2400 Architecture and Development Challenge

Architecture

- XScale core
- 8 Microengines(MEs)
- Each ME
  - run up to 8 threads
  - 4K instruction store
  - Local memory
- Scratchpad memory, SRAM & DRAM controllers



Challenges

- Limited size of instruction memory
- Distribute data to versatile memory units
- C compiler not available for IXP2400

Resource Allocation

SRAM (8MB)

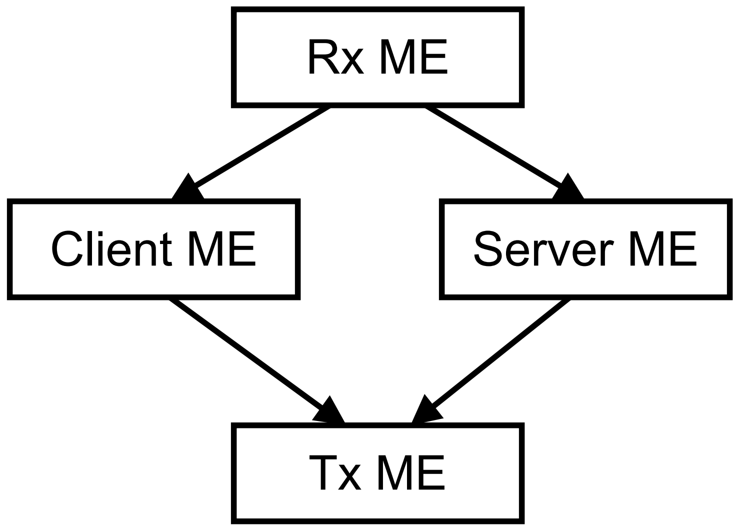
- Client side CB list
- Server side CB list
- server selection table
- Locks

- Client-side control block list
  - record states for connections between clients and SpliceNP, states after splicing
- Server-side control block list
  - record states for connections between server and SpliceNP

DRAM (256MB)

Packet buffer

Microengines



Scratchpad (16KB)

Packet queues

Comparison of Functionality (I)

- *A lite version of TCP due to the limited instruction size of microengines.*

Processing a SYN packet

Step	Functionality	TCP	Linux Splicer	SpliceNP
1	Dequeue packet	Y	Y	Y
2	IP header verification	Y	Y	Y
3	IP option processing	Y	Y	N
4	TCP header verification	Y	Y	Y
5	Control block lookup	Y	Y	Y
6	Create new socket and set state to LISTEN	Y	Y	No socket, only control block
7	Initialize TCP and IP header template	Y	Y	N
8	Reset idle time and keep-alive timer	Y	Y	N
9	Process TCP option	Y	Y	Only MSS option
10	Send ACK packet, change state to SYN_RCVD	Y	Y	Y

Comparison of Functionality (II)

Processing a SYN/ACK packet

Step	Functionality	TCP	Linux Splicer	SpliceNP
1-5	Same as processing a SYN packet			
2	Reset idle time and keep-alive timer	Y	Y	N
3	Process TCP option	Y	Y	Only MSS option
4	Verify ACK numbers and flags	Y	Y	Y
5	Connection establishment timer	Y	Y	N
6	Initialize receive sequence number	Y	Y	Y
7	Set state to ESTABLISHED	Y	Y	Y
8	Send ACK packet	Y	Y	Y

Comparison of Functionality (III)

Processing a data packet

Step	Functionality	TCP	Linux Splicer	SpliceNP
1-5	Same as processing a SYN packet			
6	Reset idle time and keep-alive timer	Y	Y	N
7	Process TCP option	Y	Y	Only MSS option
8a	Wake up receiving process	Y	Direct forwarding	Direct forwarding
	Copy data to application	Y	N	N
8b	Delete acknowledged data from send buffer	Y	Direct forwarding	Direct forwarding
	Wake up waiting process	Y	N	N
9	Flow control processing	Y	Y	N

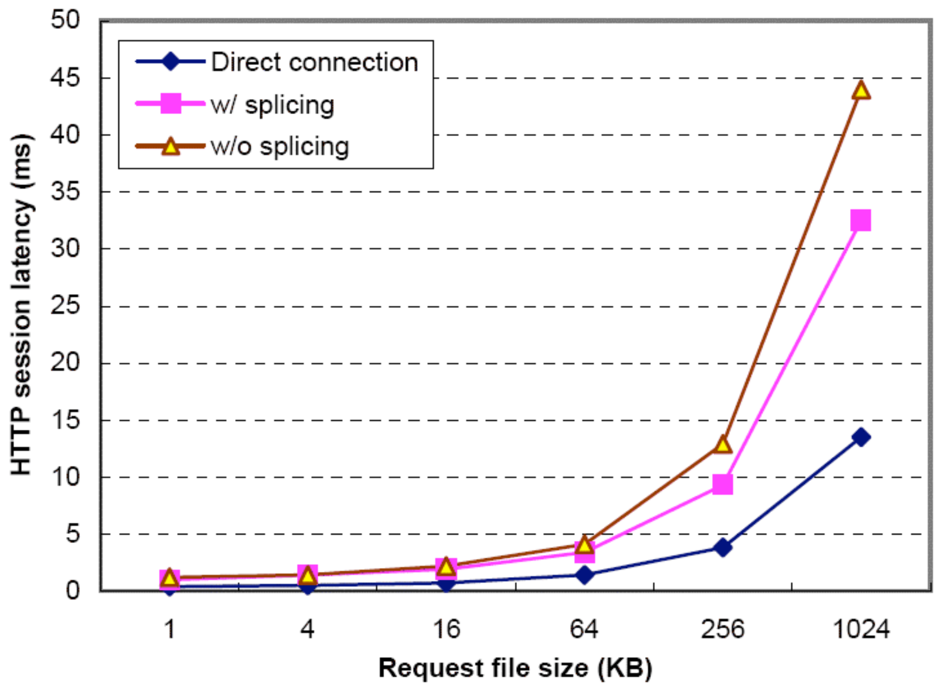
Outline

- Background and Motivation
- Design and Implementation
- Performance Evaluation
- Conclusions

Experimental Setup

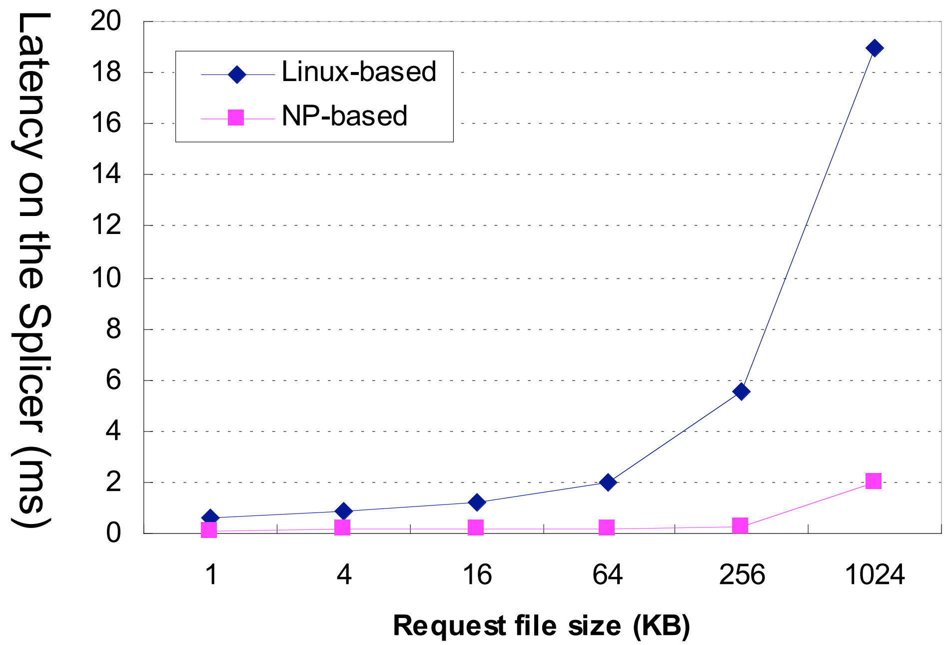
- Radisys ENP2611 containing an IXP2400
  - XScale & ME: 600MHz
  - 8MB SRAM and 128MB DRAM
  - Three 1Gbps Ethernet ports: 1 for Client port and 2 for Server ports
- Server: Apache web server on an Intel 3.0GHz Xeon processor
- Client: Httperf on a 2.5GHz Intel P4 processor
- Linux-based switch
  - Loadable kernel module
  - 2.5GHz P4, two 1Gbps Ethernet NICs

Latency on a Linux-based TCP Splicer



- Latency is reduced by TCP splicing

Latency vs Request File Size



- Latency reduced significantly
  - 83.3% (0.6ms → 0.1ms) @ 1KB
- The larger the file size, the higher the reduction
  - 89.5% @ 1MB file

Comparison of Packet Processing Latency

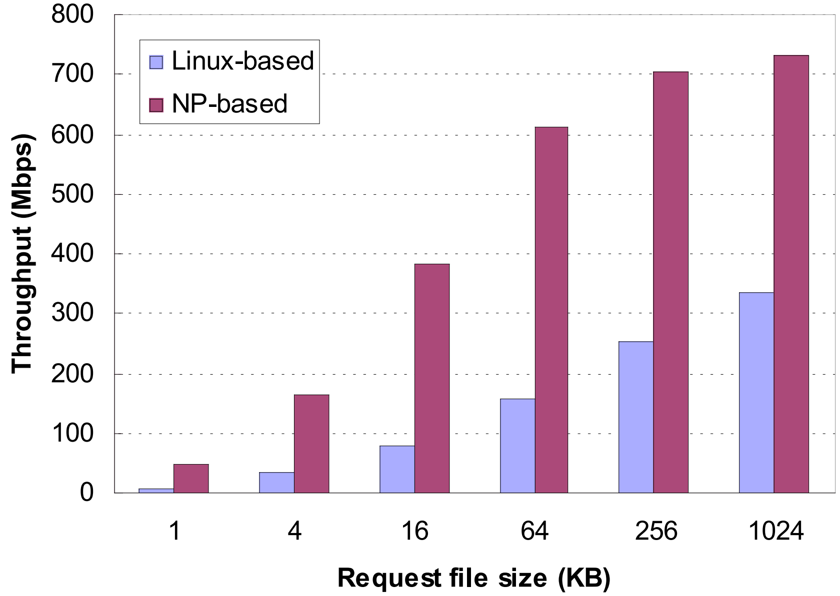
Table 5: Processing latency for control and data packets

Packet Type		IXP2400 Microengine	IXP2400 Latency (us)	Linux Latency (us)	Latency reduction
Control Packet	SYN	clientME	7.2	48	85%
	ACK/Request	clientME	8.8	52	83%
	SYN/ACK	serverME	8.5	42	80%
Data Packet	Data	serverME	6.5	13.6	52%
	ACK	clientME	6.5	13.6	52%

Analysis of Latency Reduction

Linux-based	NP-based
Interrupt: NIC raises an interrupt once a packet comes	polling
NIC-to-mem copy Xeon 3.0Ghz Dual processor w/ 1Gbps Intel Pro 1000 (88544GC) NIC, 3 us to copy a 64-byte packet by DMA	No copy: Packets are processed inside without two copies
Linux processing: OS overheads Processing a data packet in splicing state: 13.6 us	IXP processing: Optimized ISA 6.5 us

Throughput vs Request File Size



- Throughput is increased significantly
  - 5.7x for small file size @ 1KB, 2.2x for large file @ 1MB
- Higher improvement for small files
  - Latency reduction for control packets > data packets
  - Control packets take a larger portion for small files

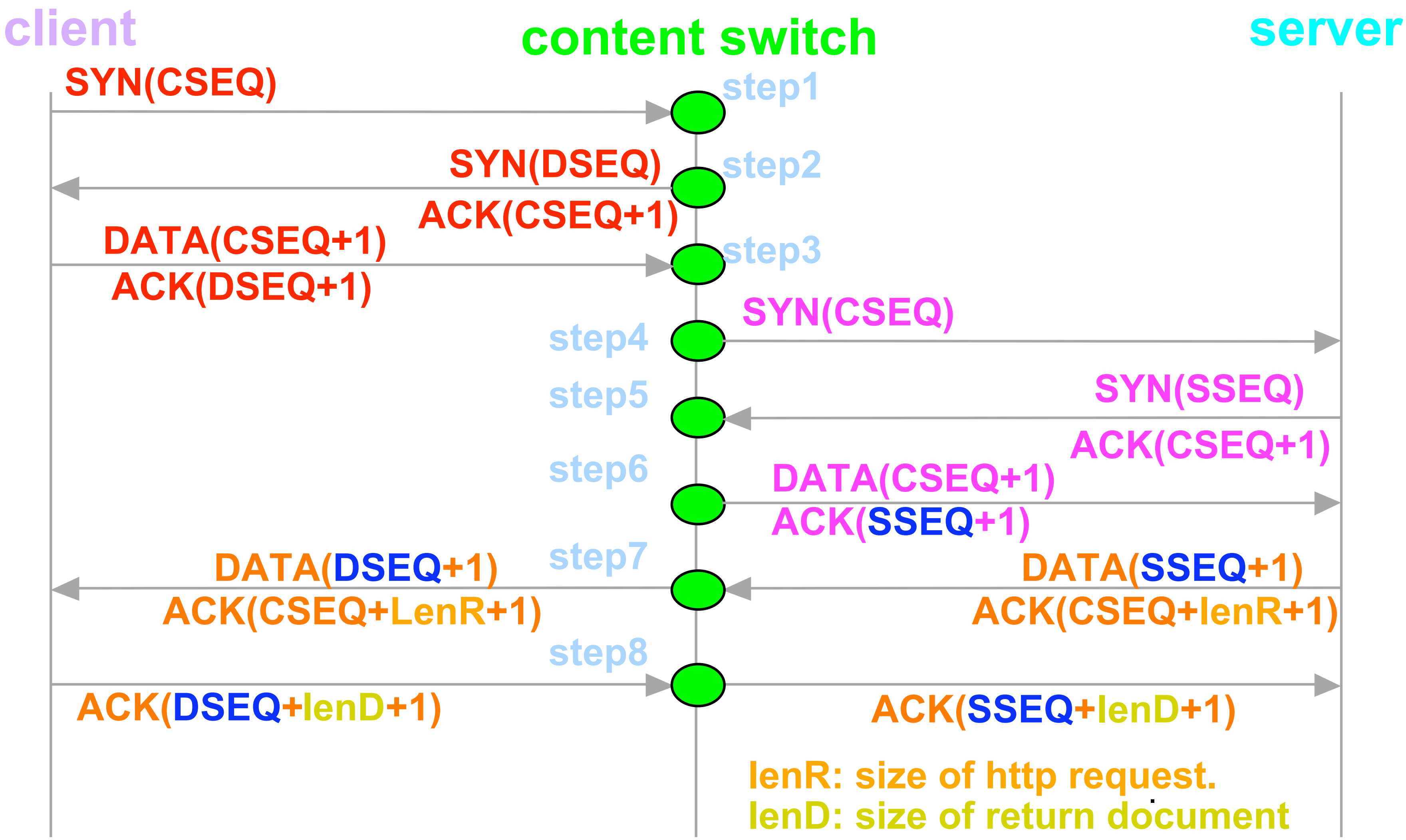
Conclusions

- Implemented TCP splicing on a network processor
- Analyzed various tradeoffs in implementation and compared its performance with a Linux-based TCP splicer
- Measurement results show that NP-based switch can improve the performance significantly
  - Process latency reduced by up to 83%
  - Throughput improved by up to 5.7x

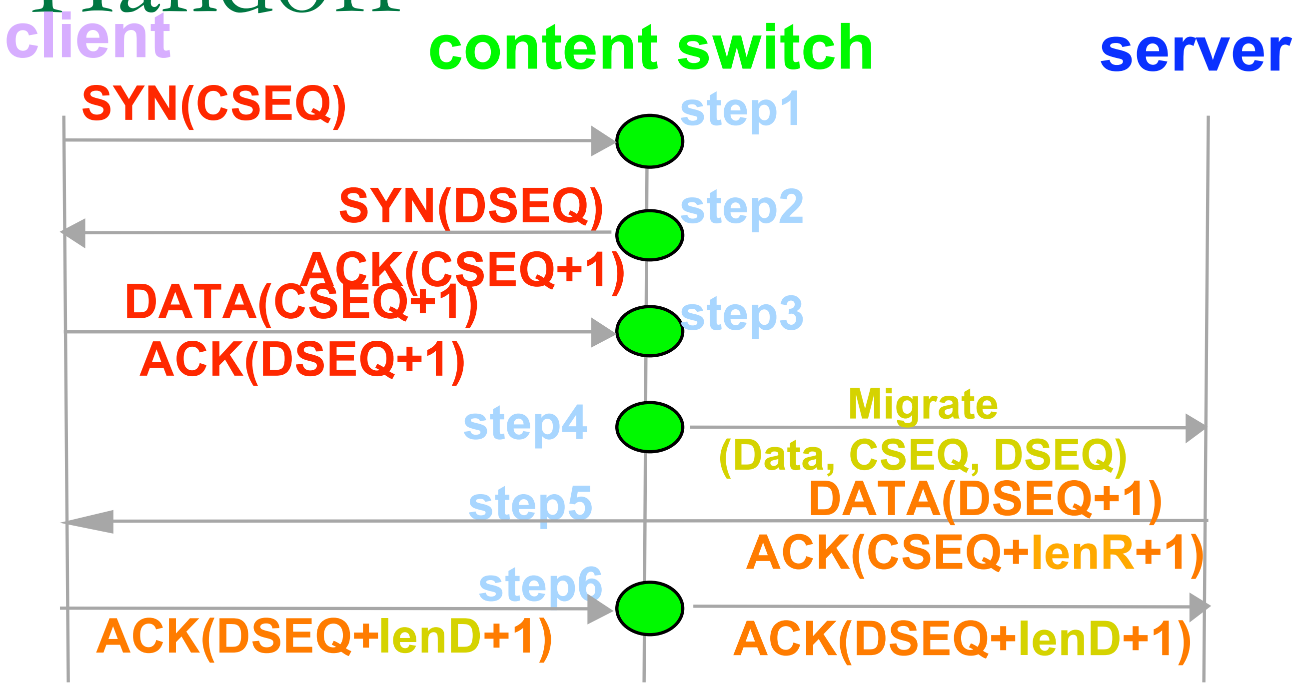
Thank you !

Questions ?

TCP Splicing



TCP Handoff



- Migrate the created TCP connection from the switch to the back-end sever
  - Create a TCP connection at the back-end without going through the TCP three-way handshake
  - Retrieve the state of an established connection and destroy the connection without going through the normal message handshake required to close a TCP connection
- Once the connection is handed off to the back-end server, the switch must forward packets from the client to the appropriate back-end server