Concurrency, Threads, and Events

Presented by Hakim Weatherspoon

On the Duality of Operating System Structures

Hugh C. Lauer and Roger M Needham

- Hugh C. Lauer
  - Another Xerox Park person
  - Founded a number of businesses:

    Real-Time Visualization unit of

    Mitsubishi Electric Research Labs (MERL)
- Roger M. Needham
  - Known for

    Kerberose, Needham-Schroeder security protocol, and key exchange systems

Message vs Procedure oriented system (i.e. Events vs Threads)

- Are they really the same thing?
- Lauer and Needham show
  - 1) two models are duals
    - Mapping exists from one model to other
  - 2) dual programs are logically identical
    - Textually similar
  - 3) dual programs have identical performance
    - Measured in exec time, compute overhead, and queue/ wait times

Message-oriented system

- Calls:
  - SendMessage, AwaitReply
  - SendReply
  - WaitForMessage
- Characteristics
  - Synchronization via message queues
  - No sharing of data structures/address space
  - Number of processes static

Message-oriented system

```
begin m: messageBody;
        i: messageId;
        p: portId;
        s: set of portId;
        . . . --local data and state information for this process
        initialize;
        do forever;
                [m, i, p] ← WaitForMessage[s];
                case p of
                        port1 =>. . . ; --algorithm for port1
                        port2 =>. . .
                                if resourceExhausted then
                                        s ← s - port2;
                                SendReply[i, reply];
                                . . . ; --algorithm for port2
                        . . .
                        portk =>. . .
                                s ← s + port2
                                . . . ; --algorithm for portk
                endcase;
        endloop;
end.
```

Process-oriented system

- Calls:
  - Fork, Join (process)
  - Wait, Signal (condition variables)
- Characteristics
  - Synchronization via locks/monitors
  - Share global address space/data structures
  - Process creation very dynamic and low-overhead

Process-oriented system

```
ResourceManager: MONITOR =
        c: CONDITION;
        resourceExhausted: BOOLEAN;
        . . . --global data and state information for t
        proc1: ENTRY PROCEDURE[ . . . ] =
                . . . ; --algorithm for proc1
        proc2: ENTRY PROCEDURE[ . . . ] RETURNS[ . . . ]
                BEGIN
                        IF resourceExhausted THEN WAI
                        . . .
                        RETURN[results];
                        . . . ;
                END; --algorithm for proc2
        . . .
        procL: ENTRY PROCEDURE[ . . . ] =
                BEGIN
                        . . . ;
                        resourceExhausted ← FALSE;
                        SIGNAL c;
                        . . . ;
                END; --algorithm for procL
        endloop;

        initialize;
END.
```

Duality

Message-oriented system	*Procedure-oriented system*
Processes, **CreateProcess**	Monitors, **NEW/START**
Message Channels	External Procedure identifiers
Message Ports	**ENTRY** procedure identifiers
SendMessage; AwaitReply (immediate)	simple procedure call
SendMessage; . . . AwaitReply (delayed)	**FORK; . . . JOIN**
SendReply	**RETURN** (from procedure)
main loop of standard resource manager, **WaitForMessage** statement, **case** statement	monitor lock, **ENTRY** attribute
arms of the **case** statement	**ENTRY** procedure declarations
selective waiting for messages	condition variables, **WAIT**, **SIGNAL**

- Can map one model to the other

Preservation of Performance

- Performance characteristics
  - Same execution time
  - Same computational overhead
  - Same queuing and waiting times
- Do you believe they are the same?
- What is the controversy?

SEDA: An Architecture for Well-Conditioned, Scalable Internet Services (Welsh, 2001)

- 20 to 30 years later, still controversy!

- Analyzes threads vs event-based systems, finds problems with both
- Suggests trade-off: stage-driven architecture
- Evaluated for two applications
  - Easy to program and performs well

SEDA: An Architecture for Well-Conditioned, Scalable Internet Services (Welsh, 2001)

- Matt Welsh
  - Cornell undergraduate Alum
    - Worked on U-Net
  - PhD from Berkeley
    - Worked on Ninja and other clustering systems
  - Currently works on sensor networks

What is a thread?

- A traditional “process” is an address space and a thread of control.
- Now add multiple thread of controls
  - Share address space
  - Individual program counters and stacks
- Same as multiple processes sharing an address space.

Thread Switching

- To switch from thread T1 to T2:
  - Thread T1 saves its registers (including pc) on its stack
  - Scheduler remembers T1's stack pointer
  - Scheduler restores T2' stack pointer
  - T2 restores its registers
  - T2 resumes

Thread Scheduler

- Maintains the stack pointer of each thread
- Decides what thread to run next
  - E.g., based on priority or resource usage
- Decides when to pre-empt a running thread
  - E.g., based on a timer
- Needs to deal with multiple cores
  - Didn't use to be the case
- "fork" creates a new thread

Synchronization Primitives

- Semaphores
  - P(S): block if semaphore is “taken”
  - V(S): release semaphore
- Monitors:
  - Only one thread active in a module at a time
  - Threads can block waiting for some condition using the WAIT primitive
  - Threads need to signal using NOTIFY or BROADCAST

Uses of threads

- To exploit CPU parallelism
  - Run two CPUs at once in the same program
- To exploit I/O parallelism
  - Run I/O while computing, or do multiple I/O
  - I/O may be “remote procedure call”
- For program structuring
  - E.g., timers

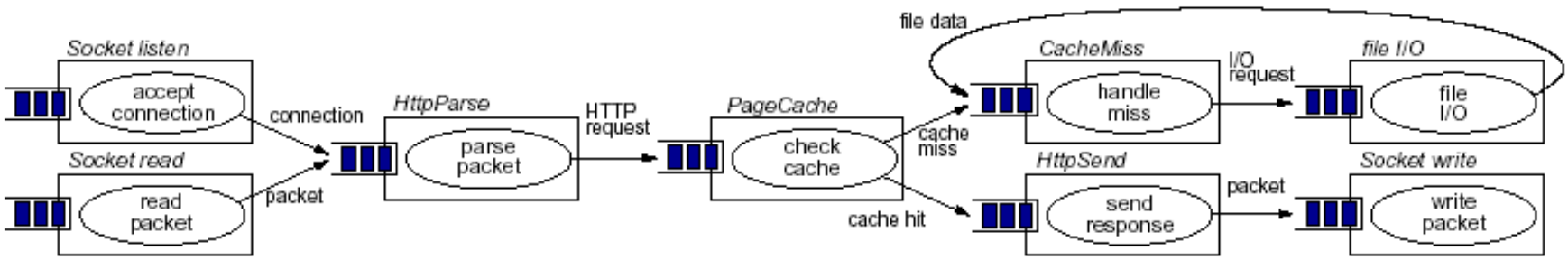
Common Problems

- Priority Inversion
  - High priority thread waits for low priority thread
  - Solution: temporarily push priority up (rejected??)
- Deadlock
  - X waits for Y, Y waits for X
- Incorrect Synchronization
  - Forgetting to release a lock
- Failed “fork”
- Tuning
  - E.g. timer values in different environment

What is an Event?

- An object queued for some module
- Operations:
  - create_event_queue(handler) → EQ
  - enqueue_event(EQ, event-object)
    - Invokes, eventually, handler(event-object)
- Handler is *not* allowed to block
  - Blocking could cause entire system to block
  - But page faults, garbage collection, …

Example Event System



(Also common in telecommunications industry, where it's called "workflow programming")

Event Scheduler

- Decides which event queue to handle next.
  - Based on priority, CPU usage, etc.
- Never pre-empts event handlers!
  - No need for stack / event handler
- May need to deal with multiple CPUs

Synchronization?

- Handlers cannot block → no synchronization
- Handlers should not share memory
  - At least not in parallel
- All communication through events

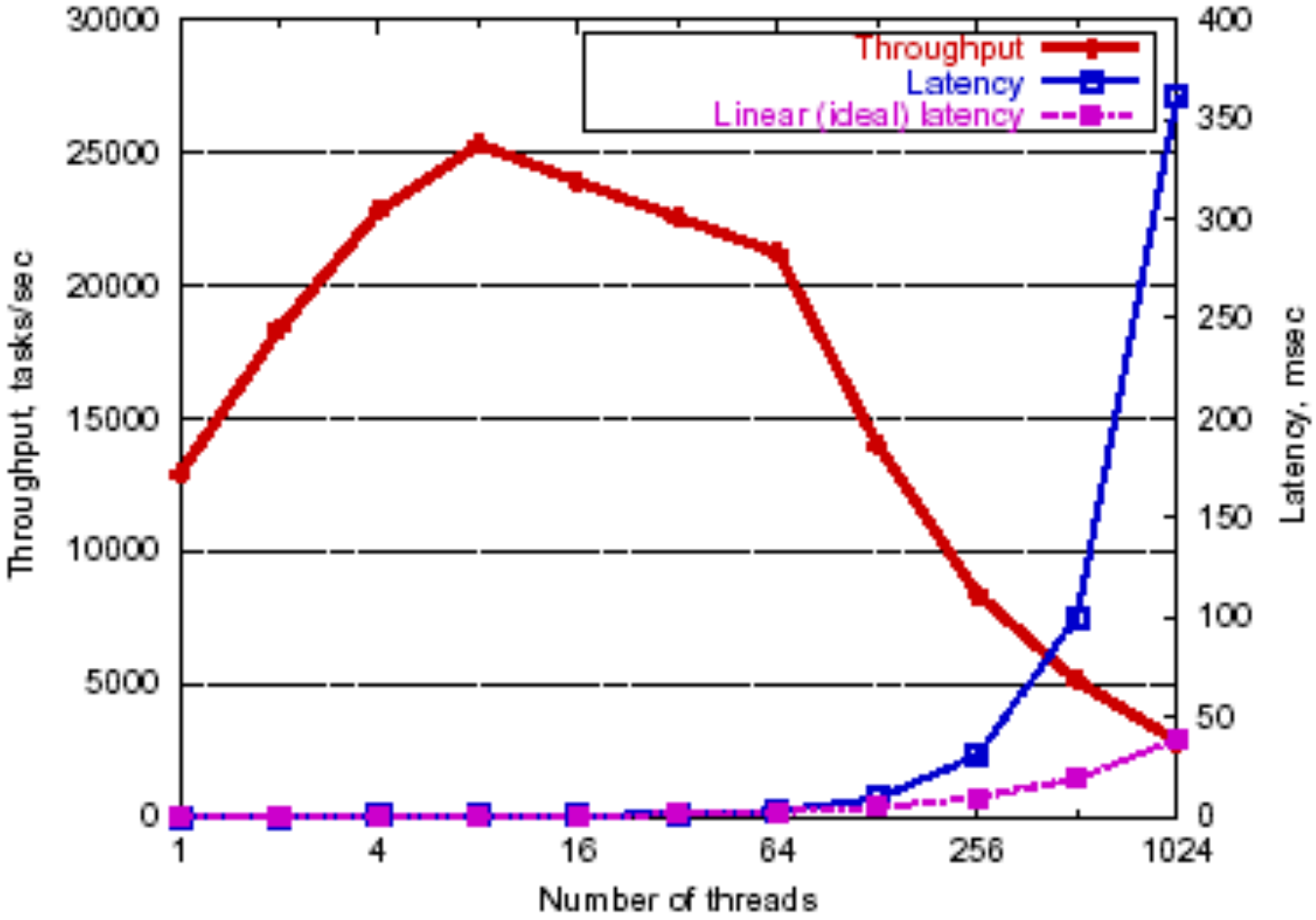
Uses of Events

- CPU parallelism
  - Different handlers on different CPUs
- I/O concurrency
  - Completion of I/O signaled by event
  - Other activities can happen in parallel
- Program structuring
  - Not so great…
  - But can use multiple programming languages!

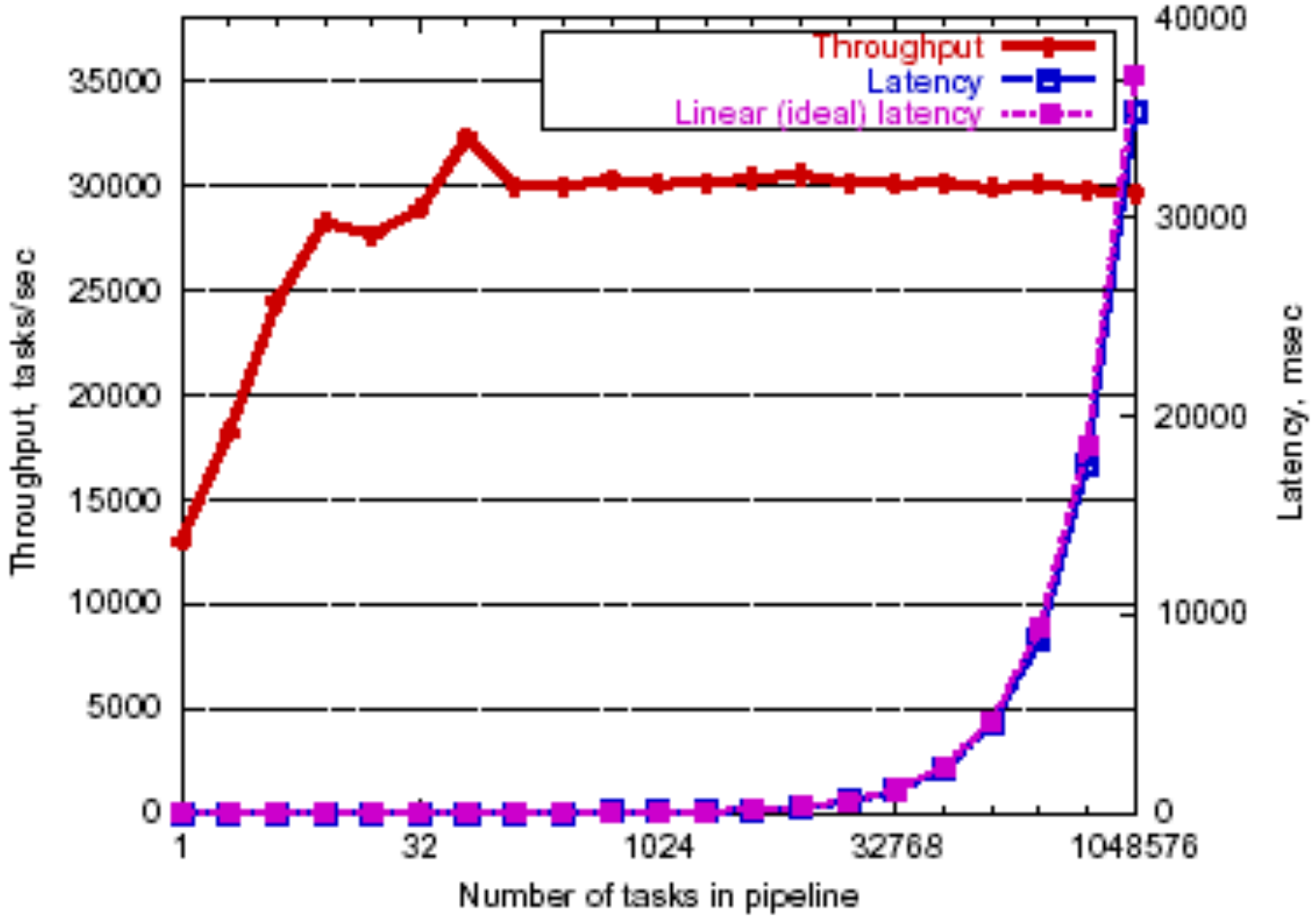
Common Problems

- Priority inversion, deadlock, etc. much the same with events
- Stack ripping

Threaded Server Throughput



Event-driven Server Throughput



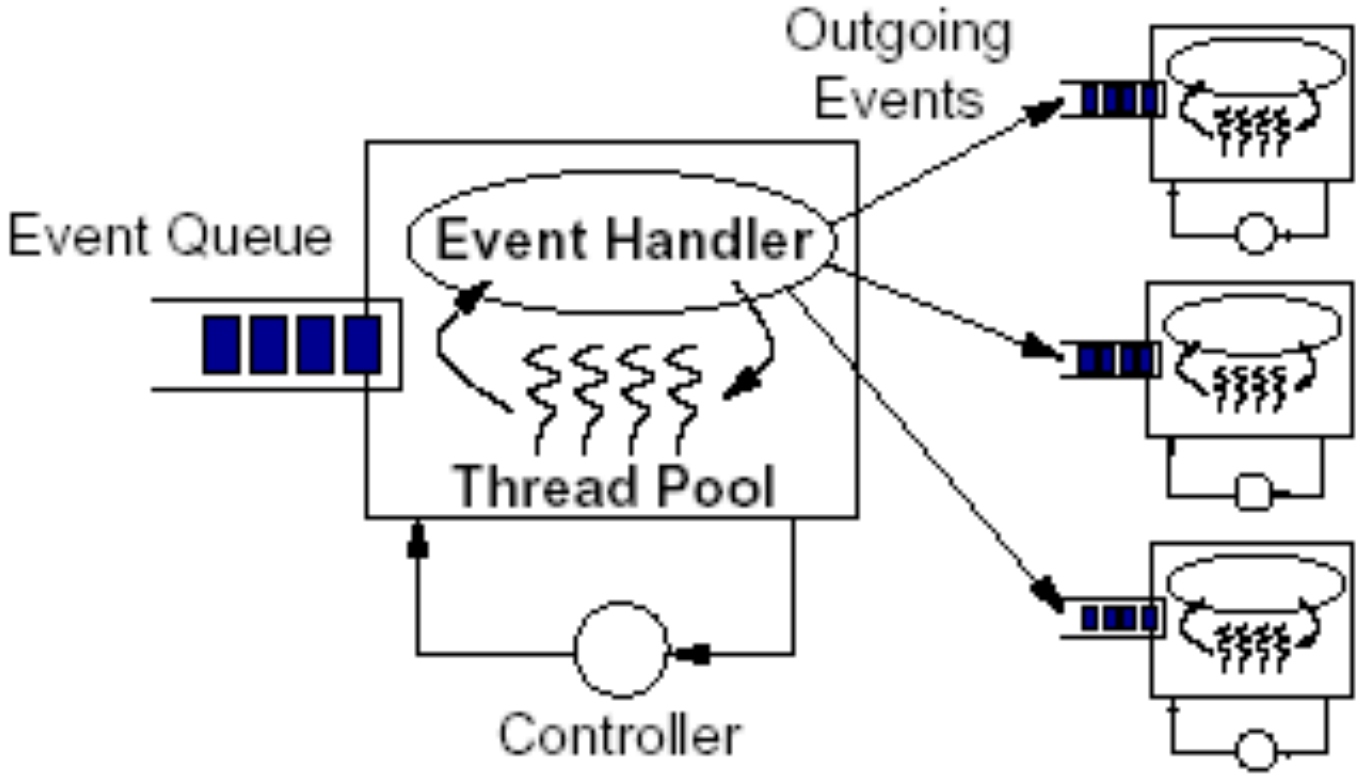
Threads vs. Events

- Events-based systems use fewer resources
  - Better performance (particularly scalability)
- Event-based systems harder to program
  - Have to avoid blocking at all cost
  - Block-structured programming doesn't work
  - How to do exception handling?
- In both cases, tuning is difficult

SEDA

- Mixture of models of threads and events
- Events, queues, and “pools of event handling threads”.
- Pools can be dynamically adjusted as need arises.

SEDA Stage


Outgoing
Events
Event Queue
Event Handler
Thread Pool
Controller

Best of both worlds

- Ease of programming of threads
  - Or even better
- Performance of events
  - Or even better

- Did we achieve Lauer and Needham's vision with SEDA?

Next Time

- Read and write review:

- Lab 0 – graded
- Lab 1 – due this Friday
  - Let us know how you are doing; if need help

- Project Proposal due in one and half weeks
  - Projects presentations tomorrow, Wed, 4pm, syslab
  - Also, talk to faculty and email and talk to me

Next Time

- Read and write review:
  - *A Fast File System for UNIX.* Marshall K. McKusick, William N. Joy, Samuel J. Leffler, Robert S. Fabry. ACM TOCS 2(3), Aug 1984, pages 181 -- 197.
  - *The Design and Implementation of a Log-Structured File System*, Mendel Rosenblum and Ousterhout. Proceedings of the thirteenth ACM symposium on Operating systems principles, October 1991, pages 1--15.

Ken Birman's research

- I work primarily on scalable, fault-tolerant computing for the cloud. Also interested in practical security technologies
- I'm a builder. Right now I'm building a system called Isis^2 (hear more at upcoming BB lunch)
  - Isis^2 embodies some deep principles: a rigorous model
  - Think of it as the implementation of a new theory of scalability and stability for cloud-scale data replication
- My current agenda: leverage advances in machine learning to overcome obstacles in scalability for reliable distributed systems

Three possible cs6410 topics: I

- Brewer sees a deep tradeoff between consistency in replicated data, availability and partition tolerance (CAP). Nancy Lynch formalized this and proved a theorem.
- But is CAP a valid barrier in real cloud systems?
  - The cloud community thinks so (but what do they know?)
  - Alternative hypothesis: CAP holds, but only in some peculiar conditions, and only if the system is limited to pt-to-pt (TCP) communication (reminiscent of FLP impossibility)
  - Topic: establish the bottom line truth
  - Challenge: *experimental* validation of findings would be obligatory
- Longer term: Leverage insights to offer a consistency "platform" to developers of cloud applications

Three possible cs6410 topics: II

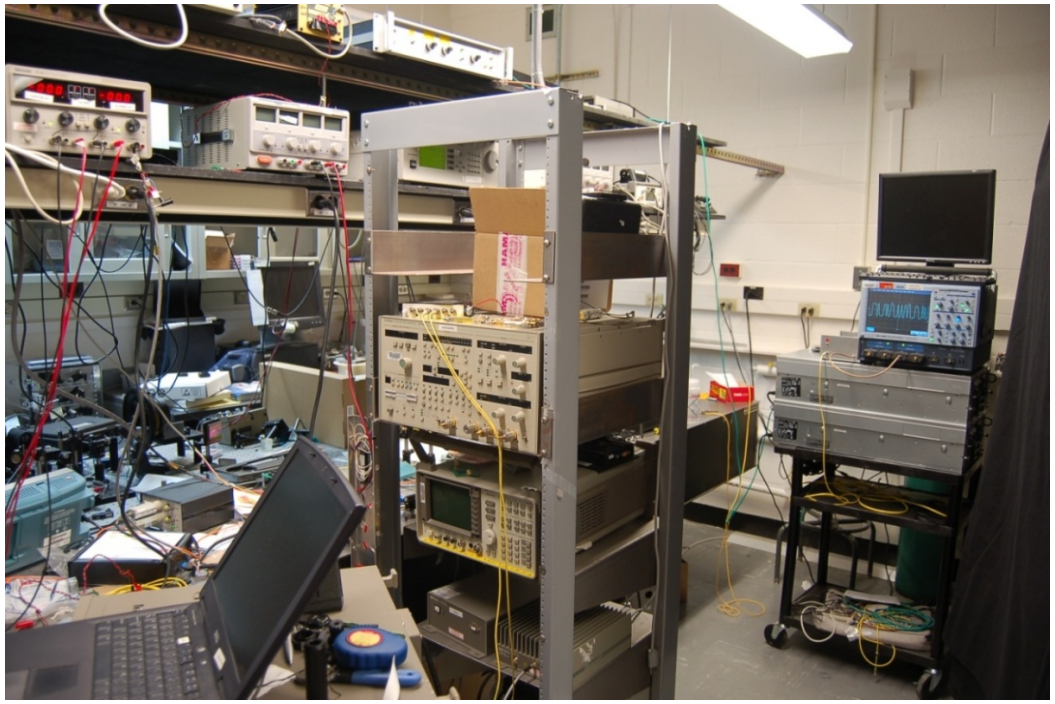
- Barebones routing
  - Suppose you have a physical router under control of your software, with ownership of its own optical fiber
    - Or a virtual one, running with a virtual "share" of the optical fibers in some Internet setting
  - Minimal operating system, other software
  - Could you implement a new routing infrastructure that composes, is secure, offers path redundancy (for mobile sources too, not just destinations), and scales up to handle billions of connections?
- Longer term: build it, deploy on NEBULA (joint project with Cisco researchers)

Three possible cs6410 topics: III

- What is the very best way to do flow control for multicast sessions?
  - We already have flow control for point-to-point; we call it TCP and it rules the world
  - IP multicast malfunctions by provoking loss if abused, yet we lack a good flow control policy for IPMC. But prior work in our group suggests that these issues can be overcome
  - Goal here would be to solve the problem but also create a *theory of stability* for scaled-up solution
- Long term: implement within Isis^2

Connection to machine learning

- Most of these are “old” topics, but in the old days we worked on small scenarios: 3 servers replicating data, for example
- Today, cloud computing systems are immense and scale can make these problem seem impossibly hard (in sense of complexity theory)
- But with machine learning can potentially
  - Discover structure, such as power-law correlations in behavior, preferential attachment
  - Exploit that structure to obtain provably stable and scalable solutions to problems that matter

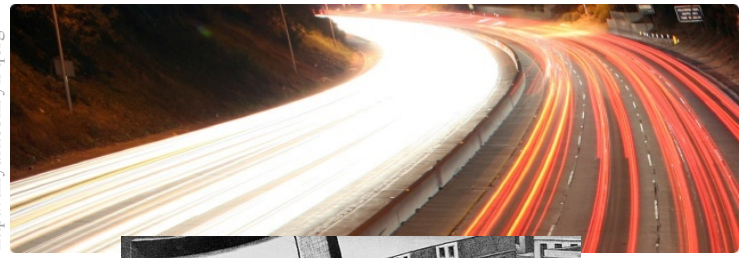


Exact temporal characterization of 10 Gbps optical wide-area network

Daniel A. Freedman, Tudor Marian, Jennifer H. Lee,
Ken Birman, Hakim Weatherspoon, Chris Xu

Research Agenda…

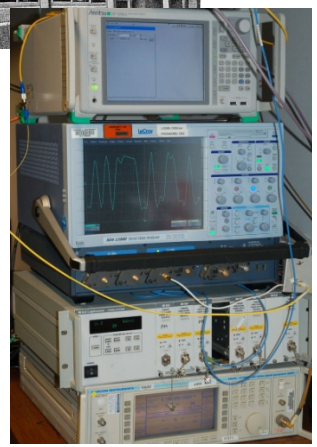
- Understand novel behavior of high-performance, lightly loaded WAN links



http://tinyurl.com/yl9q8rg

http://tinyurl.com/y8qoqwx

- Appreciate distortive impact of endpoint network adapters



- Design instrumentation (BiFocals) for precise network measurements

End-to-End Loss and the WAN

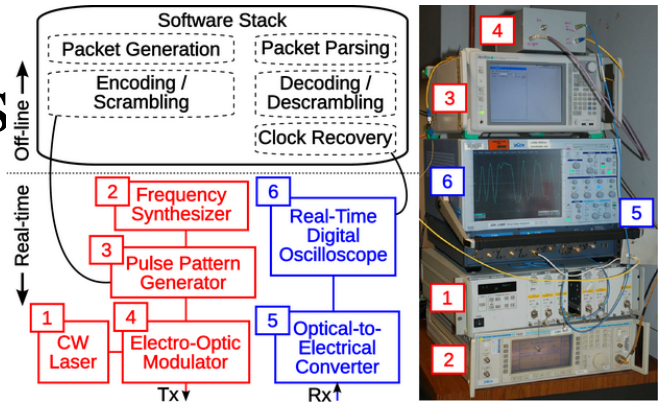
http://tinyurl.com/28jd73o

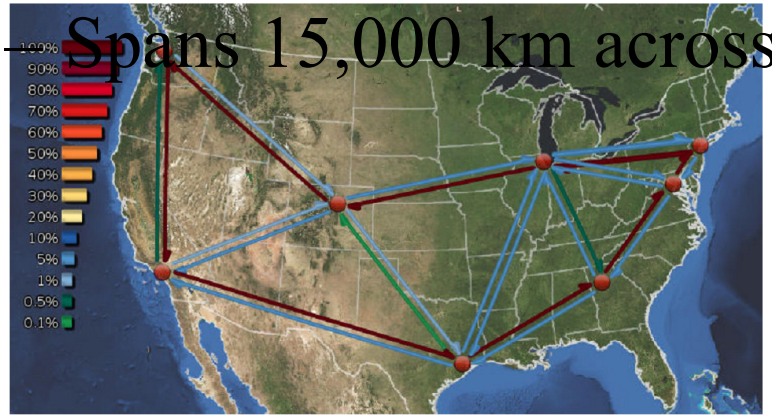
- Endpoints drop packets
  - Even at moderate data rates
  - Dropped **at** endpoint
  - **Not** an endpoint-only effect
- WAN converts input flow, with packets homogeneously distributed in time, into series of minimally-spaced chains of packets

Instrumentation and WAN Testbed

- Core architecture of BiFocals

  Exact timings at 10 Gbps!
- National LamdaRail (NLR)
  - Static routing
  - High-performance & semi-private
  - Spans 15,000 km across 11 routers



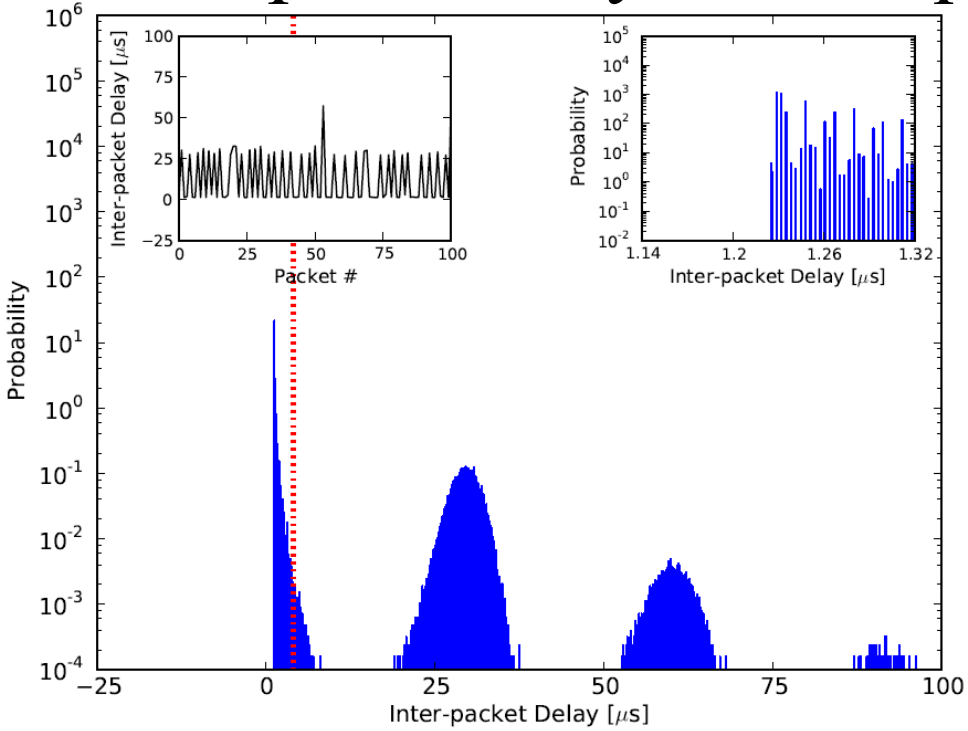


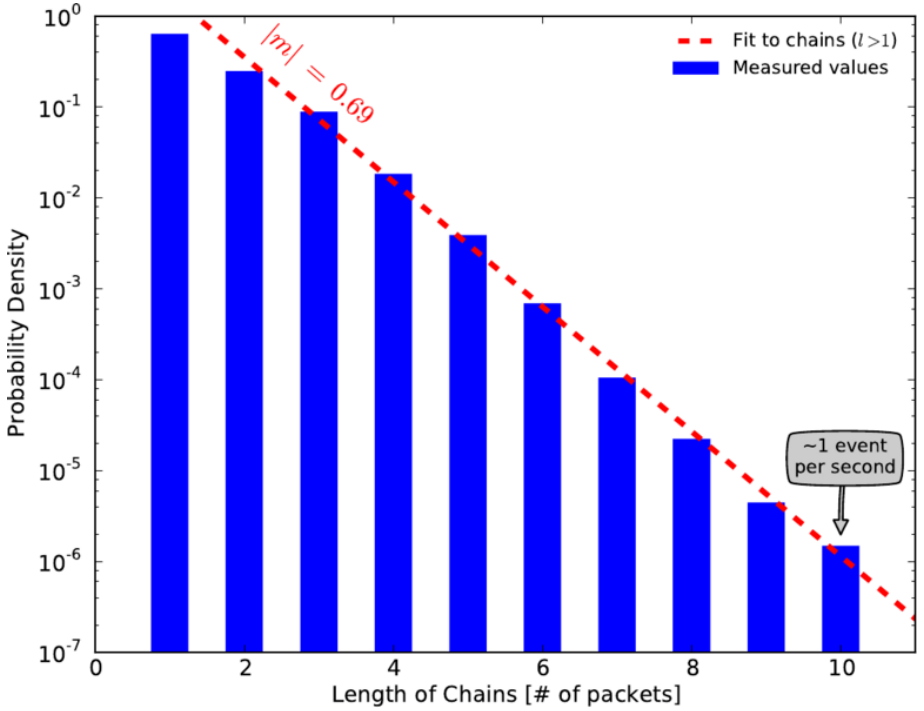
http://tinyurl.com/y8pjlt5

http://tinyurl.com/5blu3n

Exact Packet-Timing Measurements

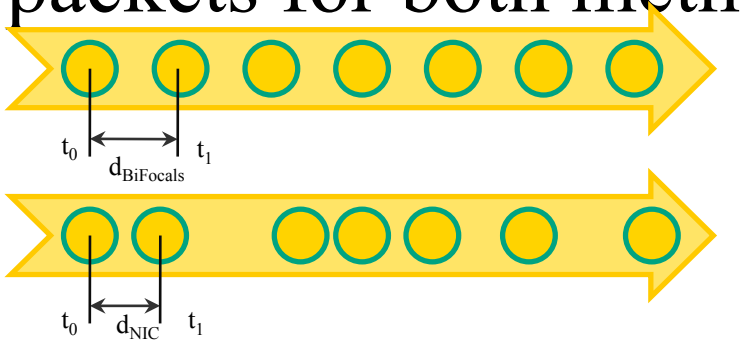
- Peak at minimum inter-packet gap
- Packet chains of increasing length are exponentially less frequent!

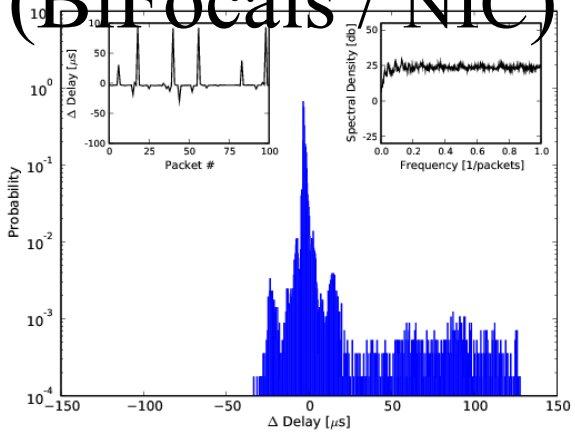




Future Work: Characterizing NICs

- Compute time delay between consecutive packets for both methods (BiFocals / NIC)





- Use to build empirical deconvolution function
  - Allows higher precision measurements with normal NICs by "backing out" distortive effects