Soft timers: efficient microsecond software timer support for network processing

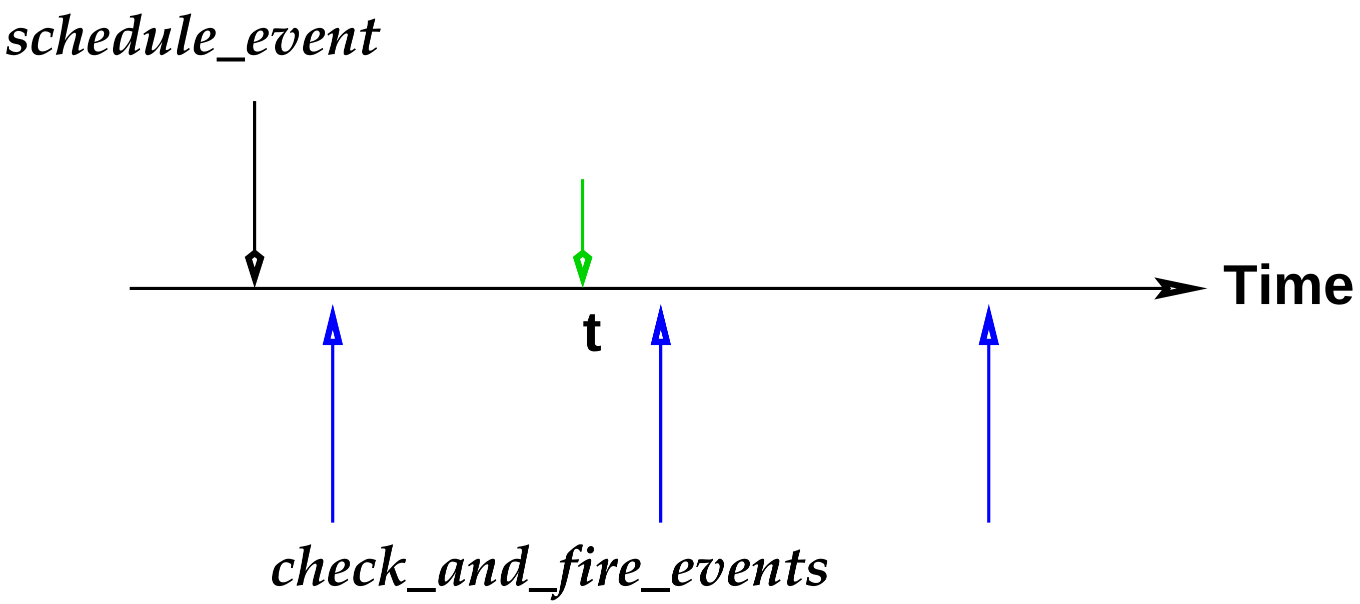
by
Mohit Aron
Peter Druschel

Department of Computer Science
Rice University

Introduction

- Soft timers: low-overhead software technique for scheduling fine-grained OS events
- Efficient high-speed networking requires fine-grained events
- Hardware timers have high overhead

Timers



- Timer module interface in OS
  - *schedule_event(t, handler)*
  - *check_and_fire_events()*
- **Note:** rate of invoking *check_and_fire_events()* determines event granularity and overhead

Outline

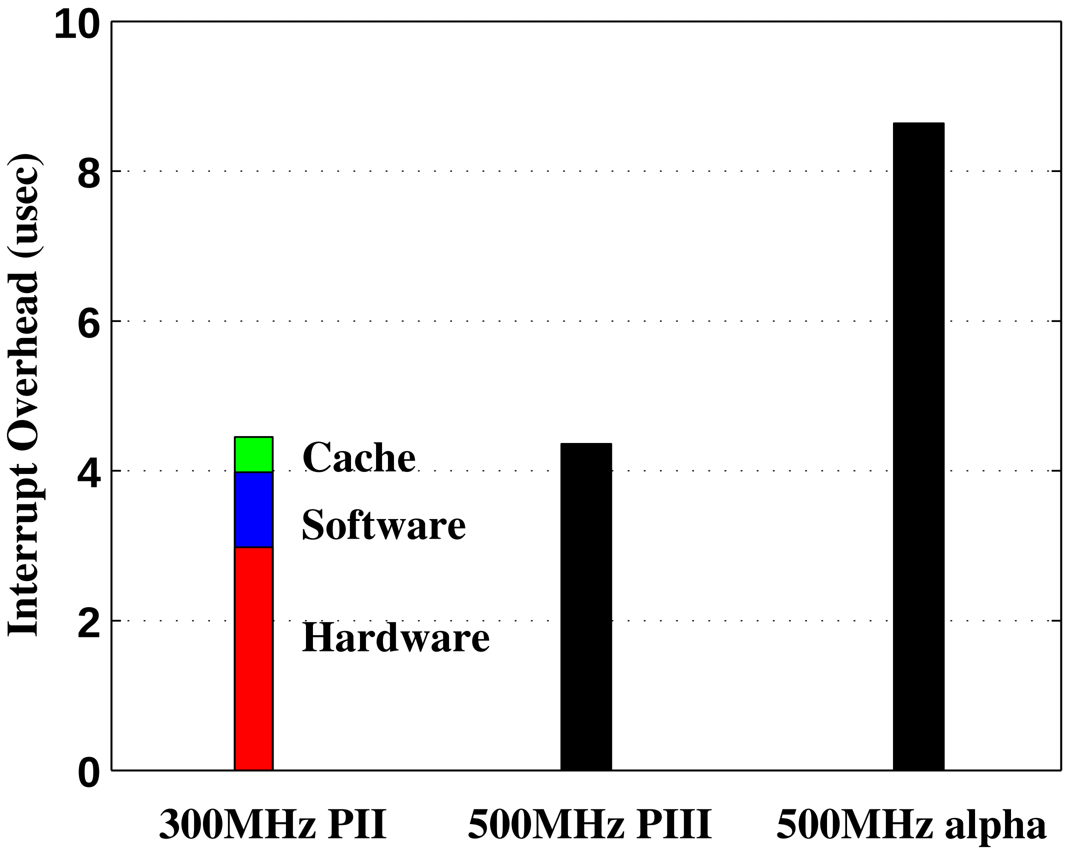
- Hardware Timers
- Soft Timers
- Rate-based pacing
- Network polling
- Conclusions

Hardware Timers

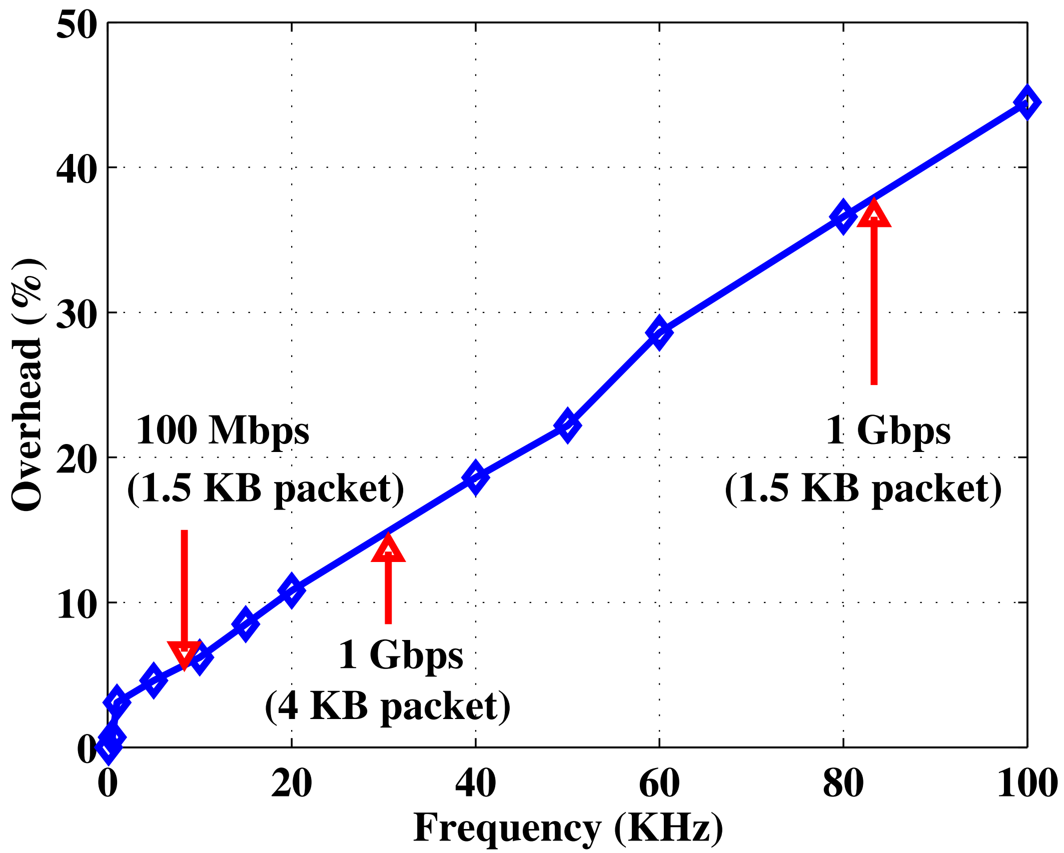
- **Programmable timer chip periodically interrupts the CPU**
  - **every 1 or 10ms in modern operating systems**
  - **invoke *check_and_fire_events***
- **Disadvantage: overhead associated with interrupts**
  - **base overhead: interrupt cost + state saving cost**
  - **cache pollution caused by event handler**

High interrupt frequency $\Longrightarrow$ high overhead

Overhead of hardware timers



Overhead of hardware timers



- Saturated Apache webserver application

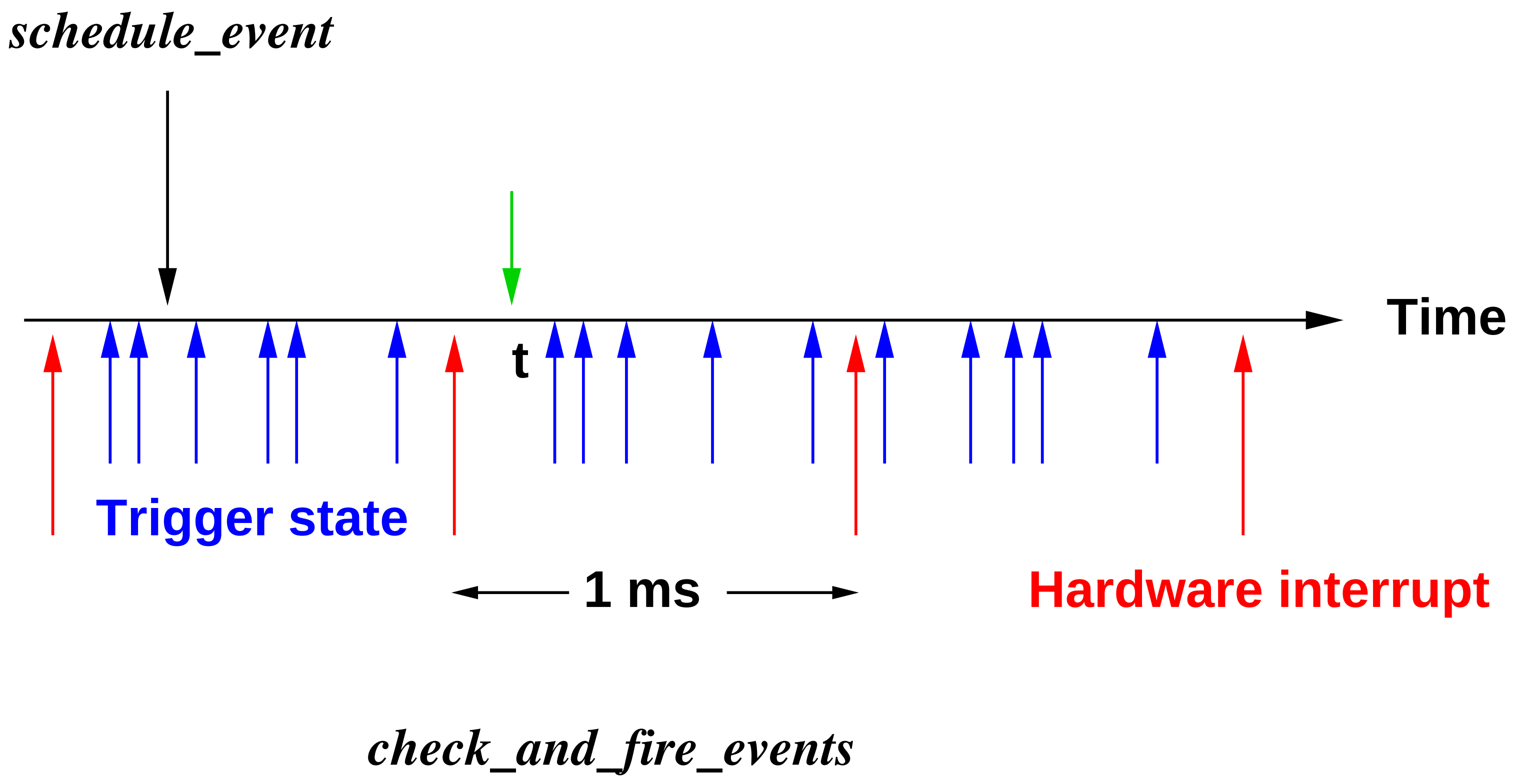
Outline

- Hardware Timers
- Soft Timers
- Rate-based pacing
- Network polling
- Conclusions

Soft Timers

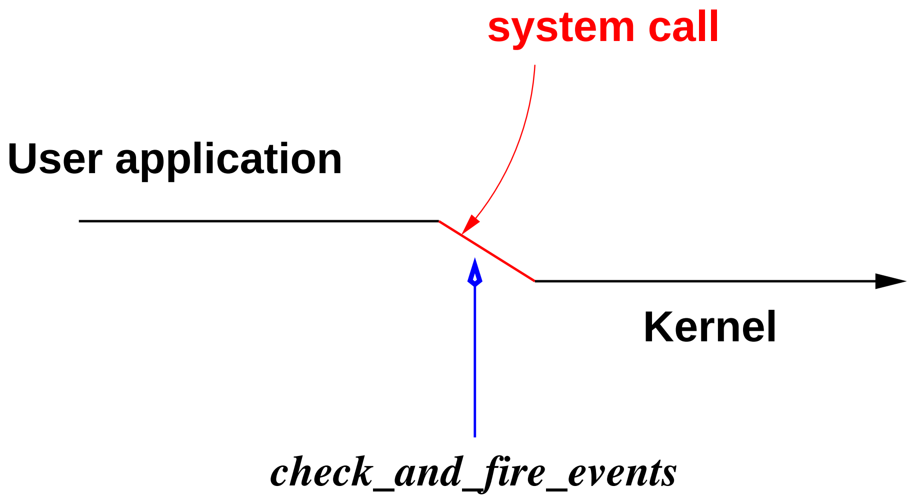
- Frequent states in system execution
  - system calls
  - exceptions (page faults, arithmetic exceptions)
  - hardware interrupts (disk interrupt, packet reception)
  - idle loop
- Key idea: invoke *check_and_fire_events* from such trigger states

Soft Timers



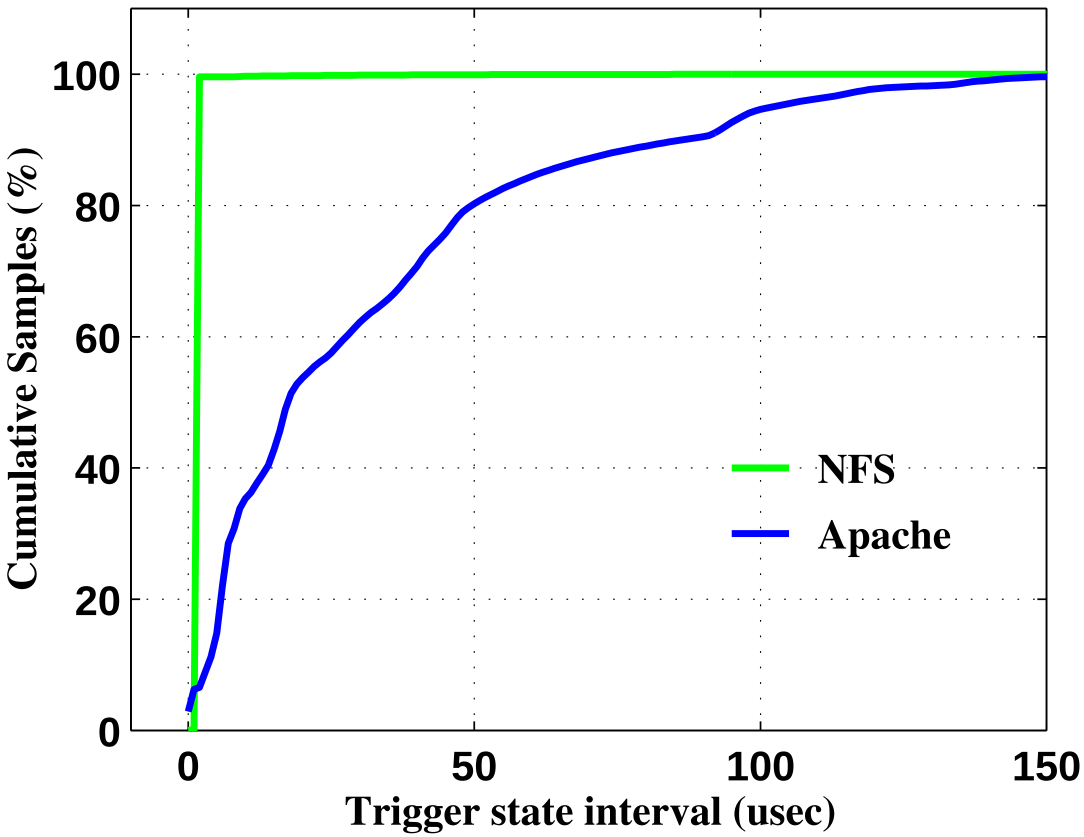
- Fine granularity (low expected delay)
- But: delay is probabilistic
- Worst-case delay bounded by periodic hardware timer

Soft Timers



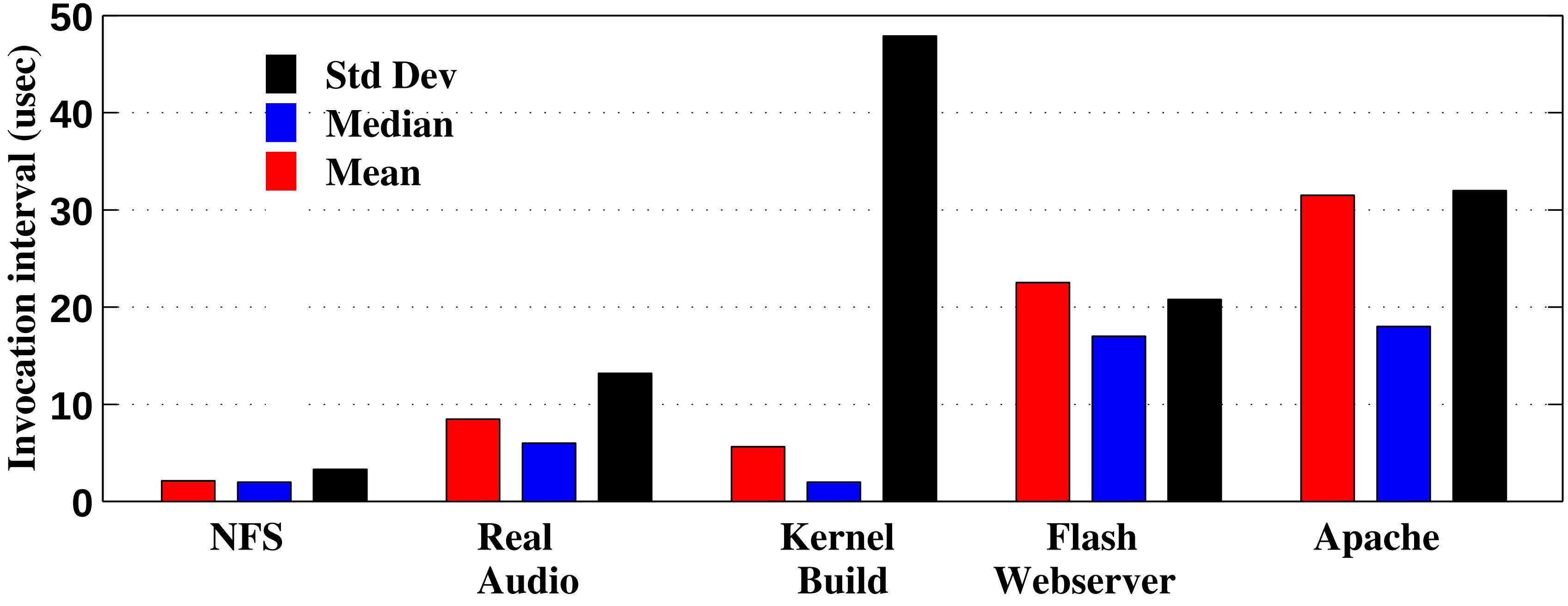
- Advantages
  - low overhead (procedure call instead of interrupt)
  - cache locality is less affected
  - fine-grained – trigger states are reached frequently

Distribution of Invocation Intervals

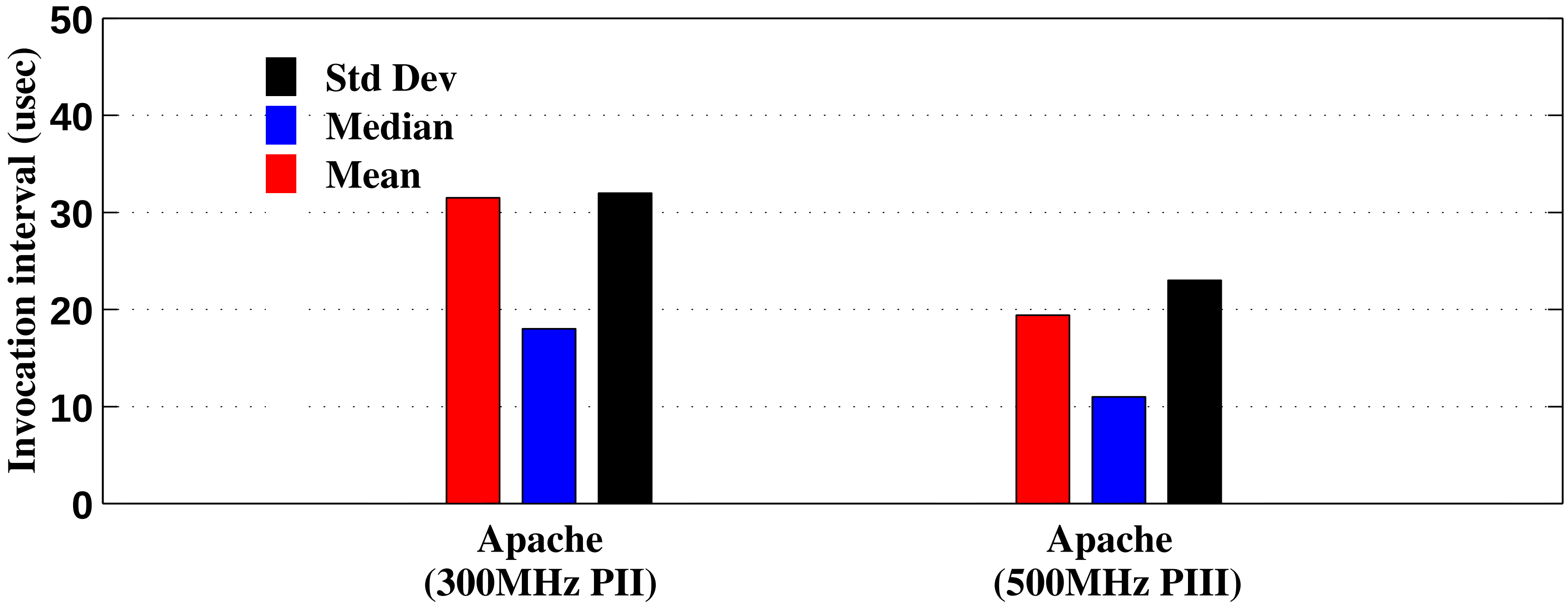


- Apache: Mean 31 μsec, Median 18 μsec
- NFS: Mean 2 μsec, Median 2 μsec

Trigger state invocation intervals



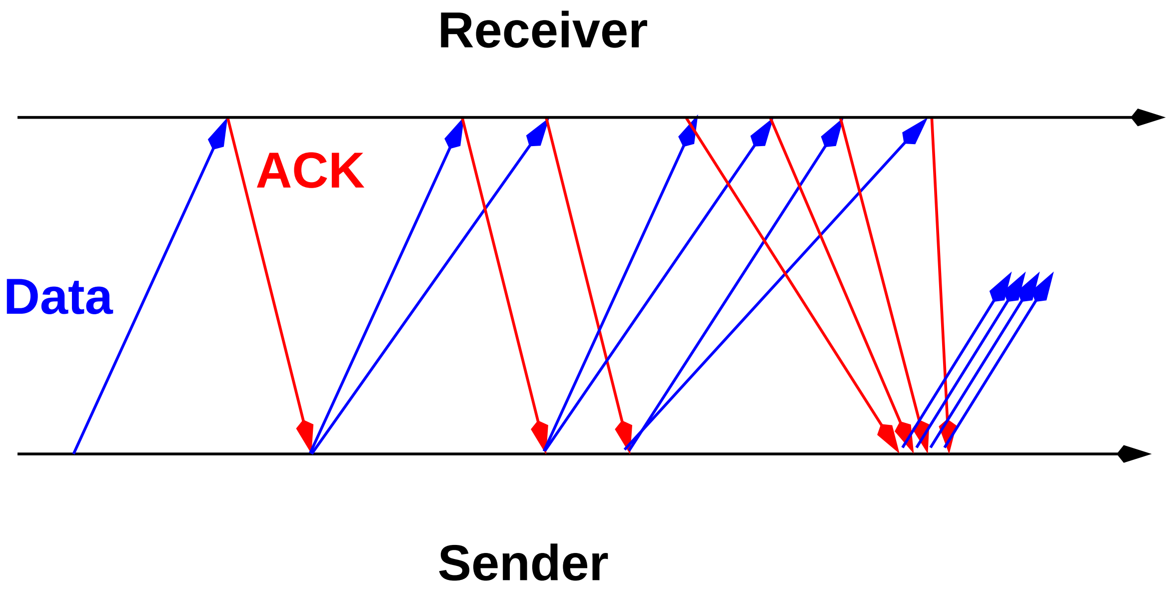
Effect of Faster CPU speed



Outline

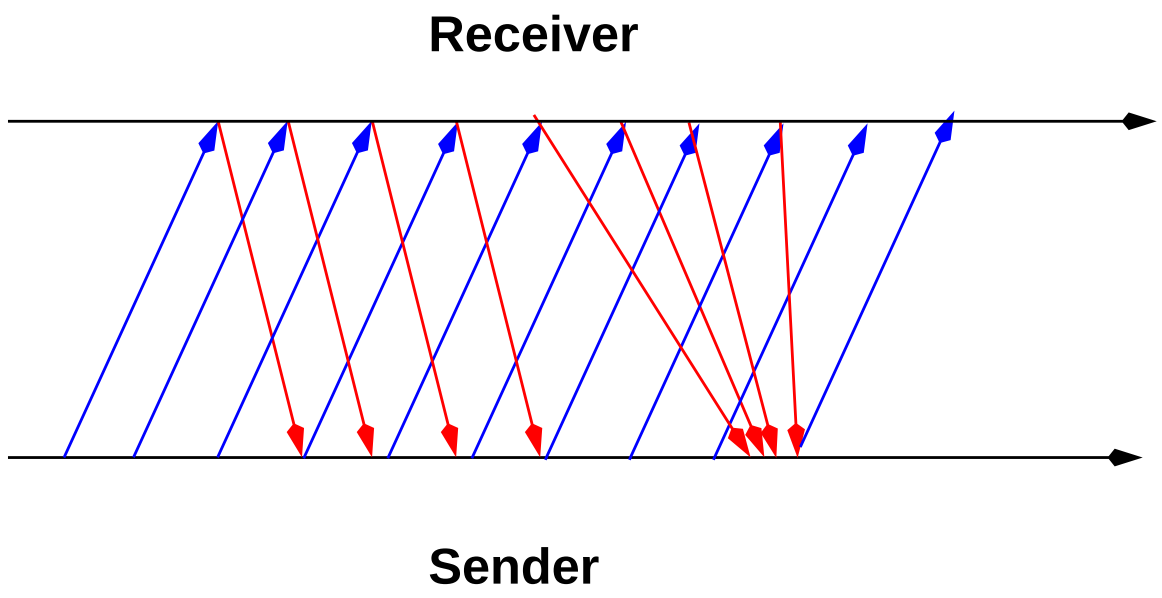
- Hardware Timers
- Soft Timers
- Rate-based pacing
- Network polling
- Conclusions

TCP self-clocking



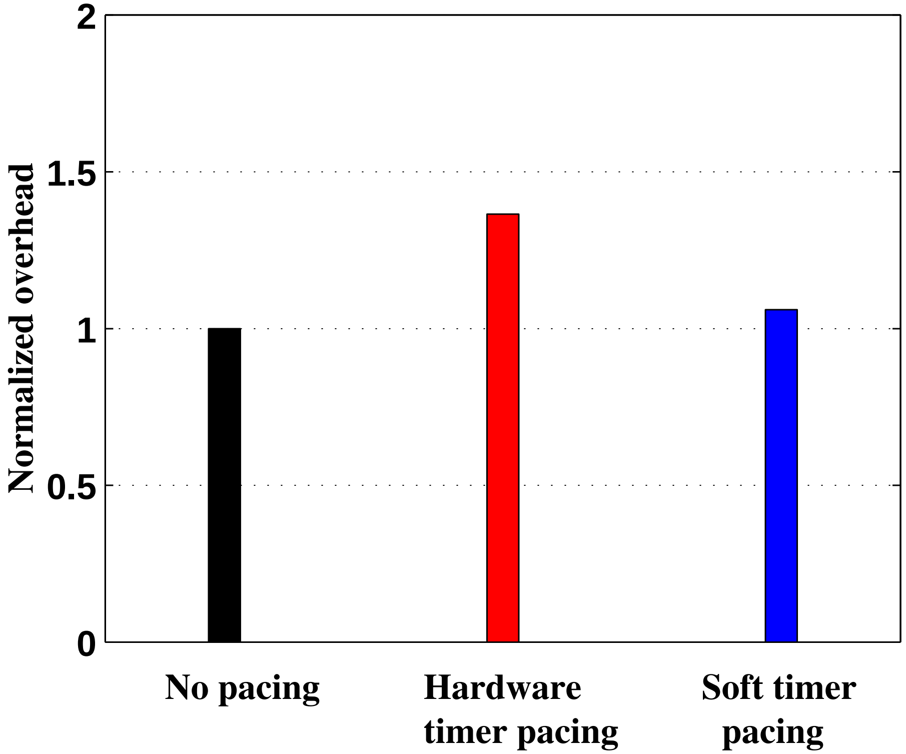
- TCP is self-clocking – packet transmissions are paced by ACKs
  - slow startup – low network utilization
  - bursty transmissions due to ACK compression or big ACKs

Rate-based pacing



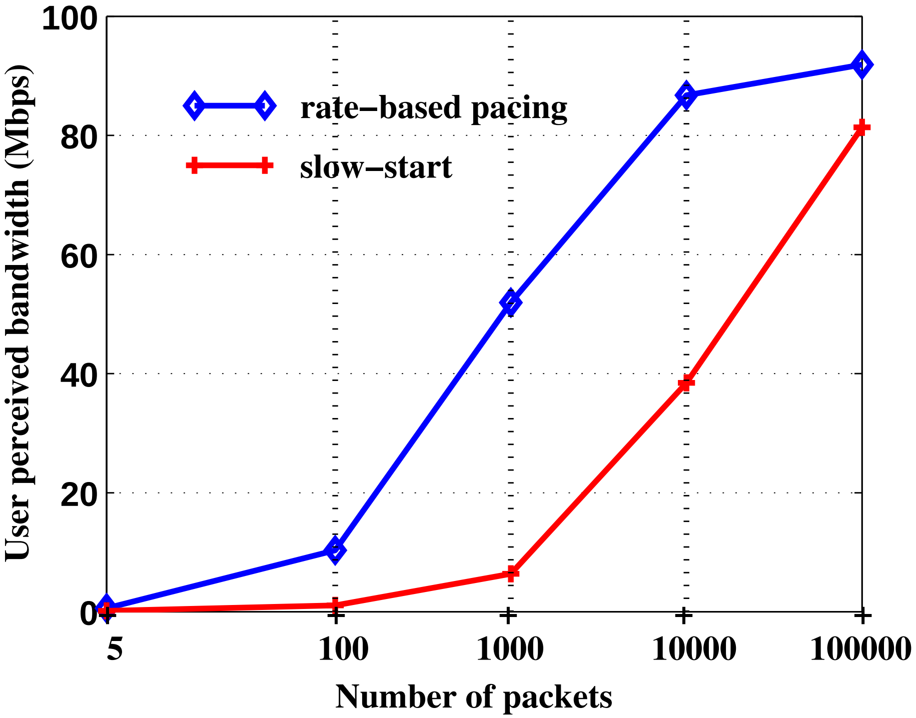
- Rate-based pacing – transmit packets independent of ACKs
  - high utilization if bandwidth is known
  - not affected by ACK spacing
  - allows QoS traffic shaping

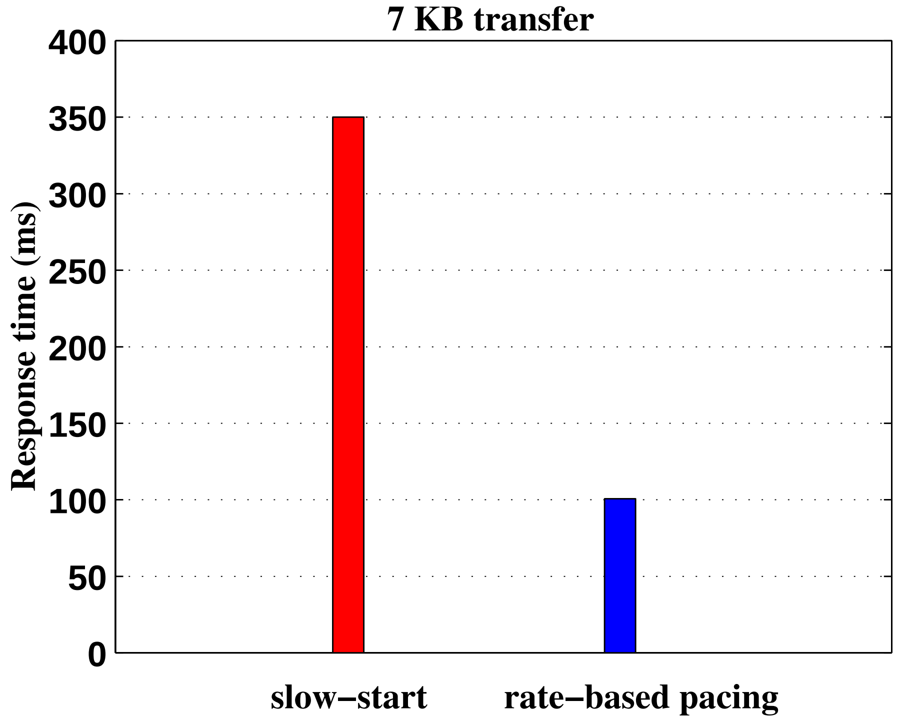
Timer overhead with rate-based pacing



- Webserver experiment designed to isolate pacing overhead
- 36% overhead with hardware timers
- 6% overhead with soft timers

Rate-based pacing vs TCP slow-start





- transfer over 100Mbps WAN with 100 ms round-trip latency
- up to a factor of 10 higher bandwidth with rate-based pacing
- 71% better response time for 7 KB transfer (typical for HTTP)

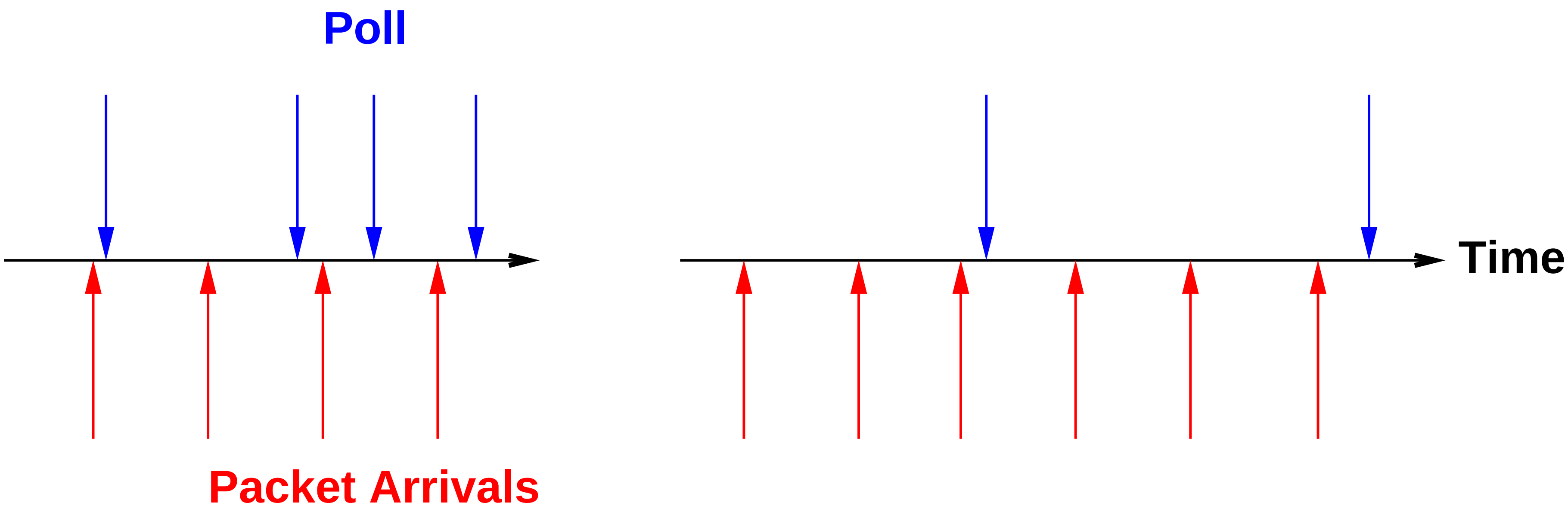
Outline

- **Hardware Timers**
- **Soft Timers**
- **Rate-based pacing**
- **Network polling**
- **Conclusions**

Network polling

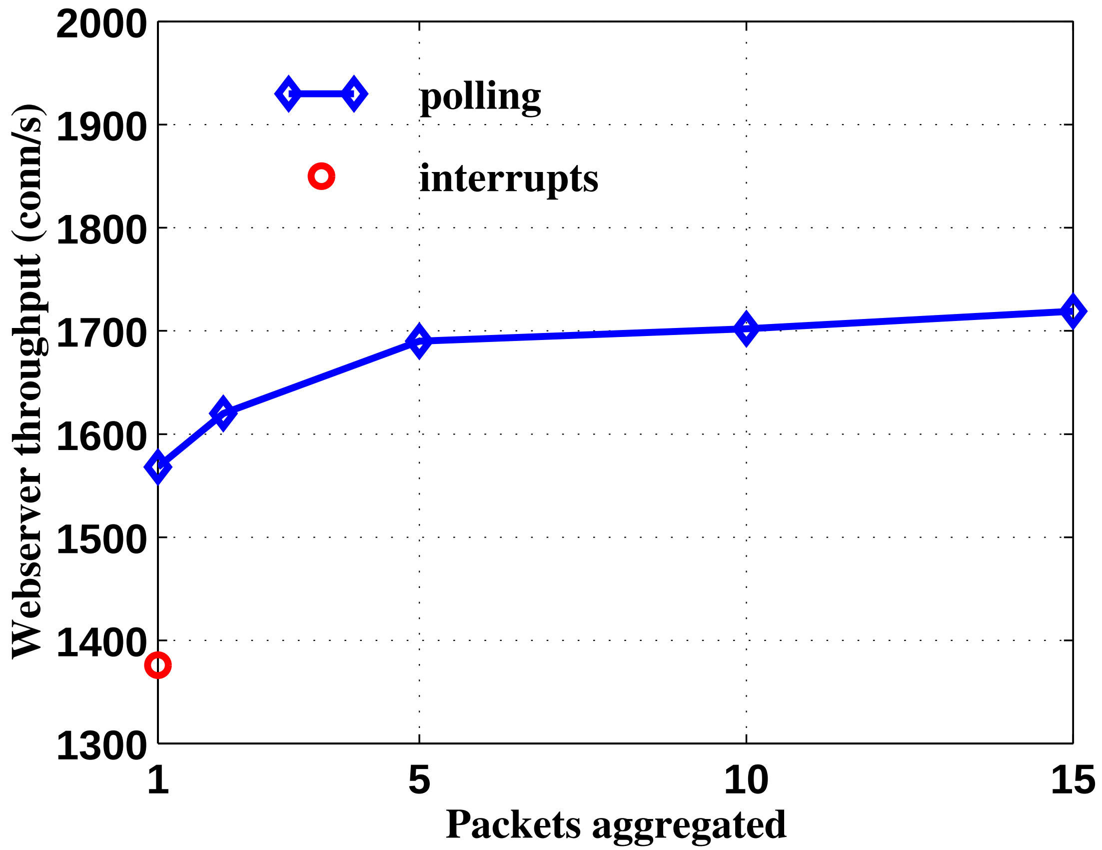
- Higher bandwidths $\longrightarrow$ higher rate of packet reception
- Interrupt upon each packet reception
  - overhead
  - loss in cache locality
- Disable interrupts and poll for network packets

Network polling with soft timers



- Dynamically schedule poll events at average packet arrival rate
- Can also aggregate processing of multiple packets
  - schedule polls to receive multiple packets
  - caching benefits

Performance with soft timer based polling



Conclusions

- **Hardware timers have large overhead**
- **Soft timers: low overhead software-based timer facility**
  - **probabilistic but bounded delays**
  - **fine-grained (tens of μsec) under realistic workloads**
  - **scale with CPU speed**
- **Used for rate-based pacing and network polling**
  - **up to factor of 10 higher bandwidth with rate-based pacing**
  - **up to 25% improvement in performance with network polling**