Measuring Performance in Real-Time Linux

Frederick M. Proctor

Group Leader, Control Systems Group

National Institute of Standards and Technology

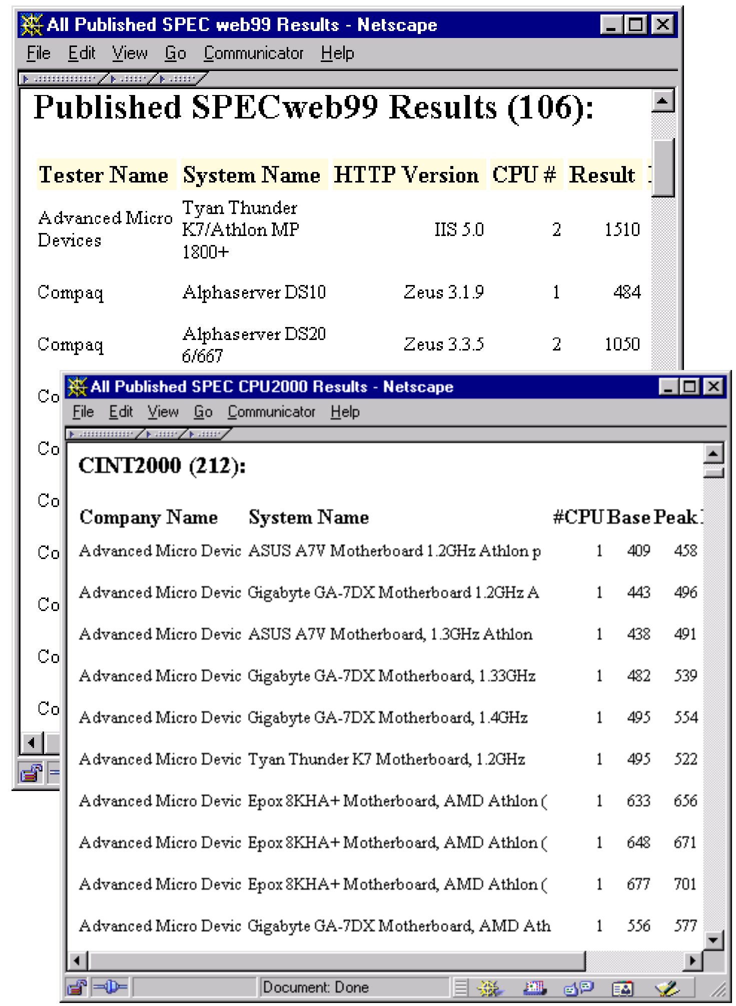
NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

1901-2001
NIST CENTENNIAL

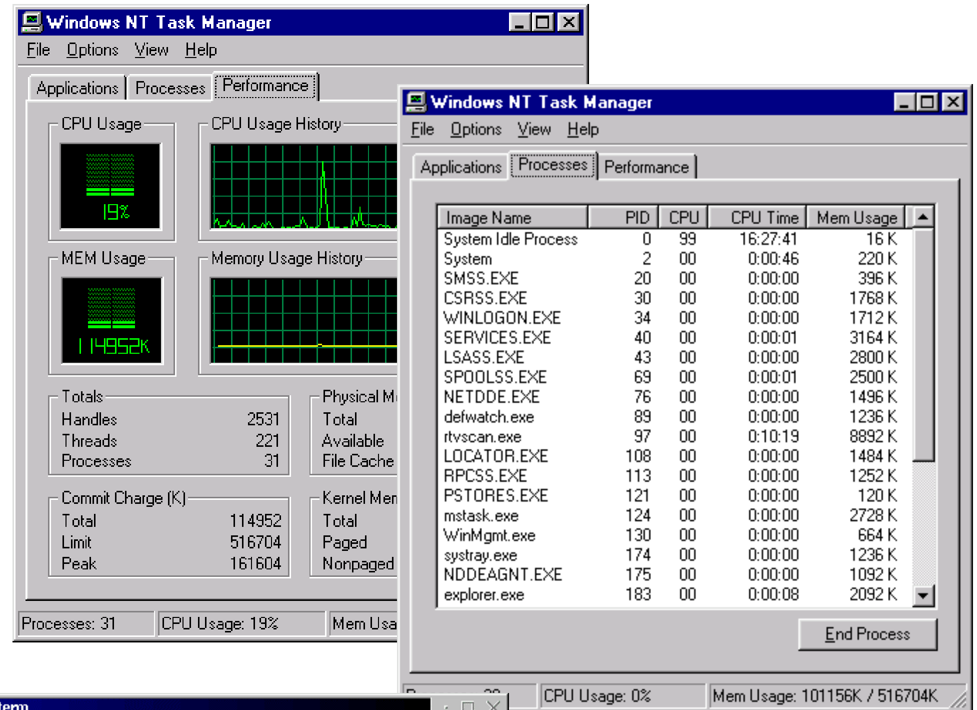
ISD
Intelligent Systems Division
Manufacturing Engineering Laboratory

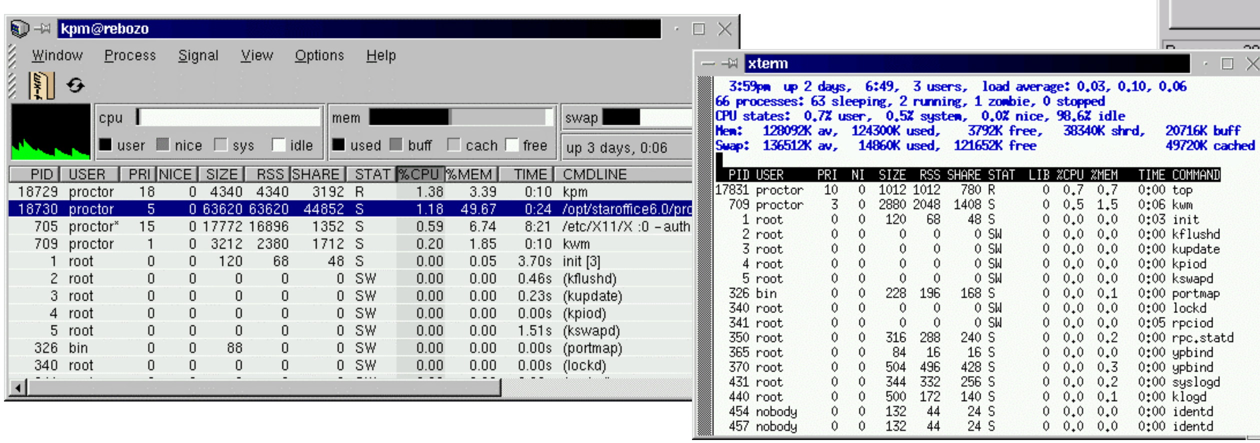
Performance Measures

- *Performance measures* are figures of merit that indicate how well a system behaves
- *Benchmarks* can provide performance measures for specific areas of interest, e.g.,
  - SPEC CPU2000 measures performance of processor, memory, compiler
  - SPEC WEB99 measures performance of web servers
  - x11perf measures performance of X servers

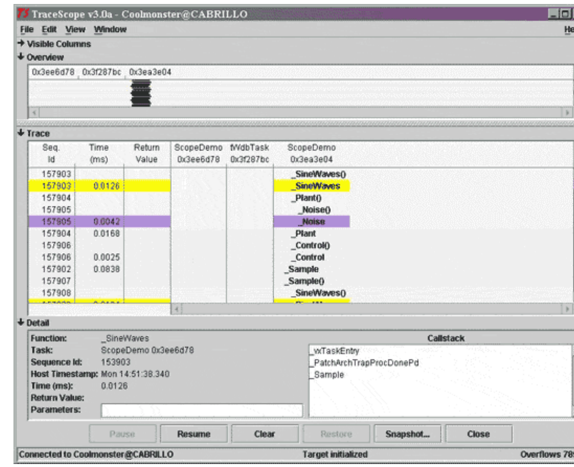


- *Monitors* show general resource use of programs in a system, e.g.,
  - ps, top and its graphical front ends
  - Windows Task Manager

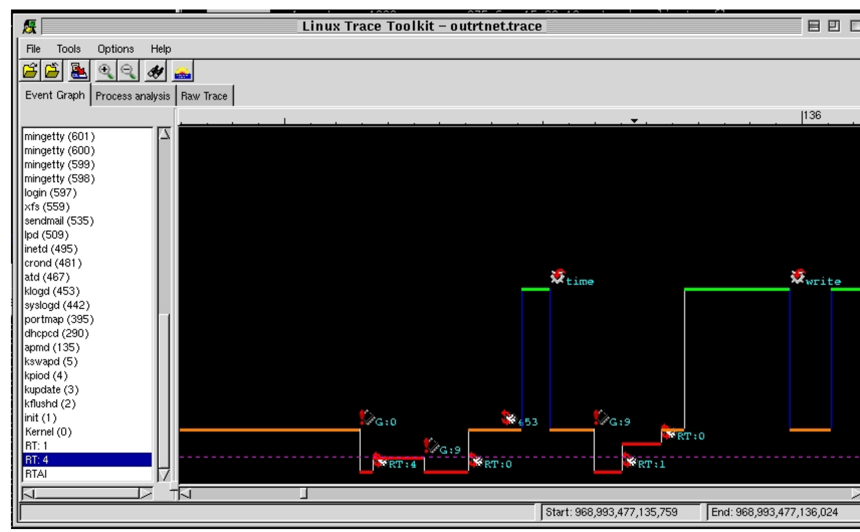




- *Profilers* show details of program execution, e.g.,
  - the profil() function, gprof, strace
  - ParaSoft insure++, inuse
  - Rational Quantify, WindRiver WindView, RTI ScopeTools
  - the Linux Trace Toolkit



- None of these specifically address performance measures for real-time systems



For us, performance measures answer the question:

How can I tell that a real-time operating system is able to satisfy my application's timing requirements?

RT Performance Measures

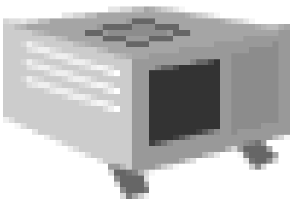
- Real-time software must execute *on time* to be correct
- *On time* can mean:
  - any time between now and a deadline
  - within some interval around a target time
- For RT operating systems, performance measures should indicate how well the RTOS satisfies on-time demands
  - what is the shortest deadline by which the RTOS can guarantee a task's execution?
  - what is the smallest interval around a target time within which the RTOS can guarantee a task's execution?
  - how do these scale with task loading?

Classic RTOS Performance Measures

- The shortest deadline measure applies to instances where an event initiates code that must run before a deadline
  - Typically the event is an interrupt, and the code is the interrupt service routine (ISR)
  - *Worst-case ISR latency* is the classic performance measure
- The smallest interval measure applies to instances where code must execute as close as possible to a target time
  - Typically the target time is one of a series of periodic timer expirations
  - *Scheduling jitter* is the classic performance measure

Types of Testing

- *External testing* uses instrumentation not normally part of the RT system to stimulate and measure RT response
  - e.g., digital storage scopes, data acquisition systems
  - advantages: equipment is part of experiment's control; entire RT system is tested; can include arbitrary features, storage capacity, timing precision
  - disadvantages: additional cost



- *Internal testing* uses native resources of the RT system
  - e.g., processor time stamp counters
  - advantages: no additional cost; tests can be incorporated into RT application for continuous monitoring or performance improvement
  - disadvantages: as with students grading their tests, "cheating" is possible; some effects will be invisible (e.g., clock chip jitter)

Testing Environment

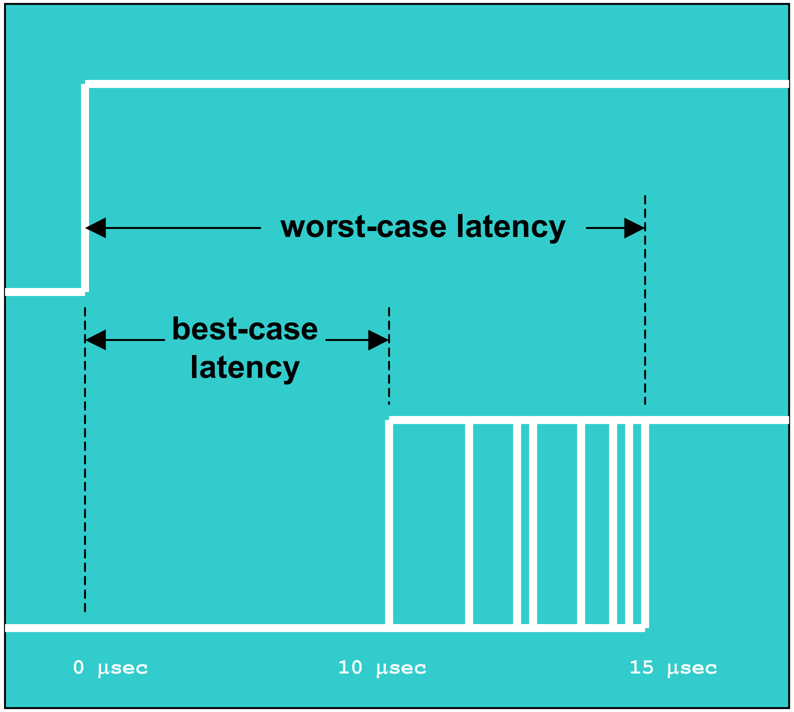
- If test results from different systems are to be compared, the testing environment must be adequately specified
  - what components must be present, e.g., network and video cards
  - what processes must be running; single v. multiuser mode
  - what optimizations are allowed or disallowed, e.g., disabling floating point support
- Hardware effects can be substantial, especially for general-purpose processors
  - optimizations like the cache introducing timing uncertainty
  - hardware reference platforms are one answer to this problem, e.g., WinCE HARP

ISR Latency

- *ISR latency* is the time between the occurrence of an interrupt and the execution of its service routine
  - "execution" is vague: time the ISR begins? completes?
  - maximum ISR latency is a system performance measure
- Latency contributors include:
  - hardware effects: processor must finish current instruction, and instruction lengths vary
  - software effects: interrupt masking and priority

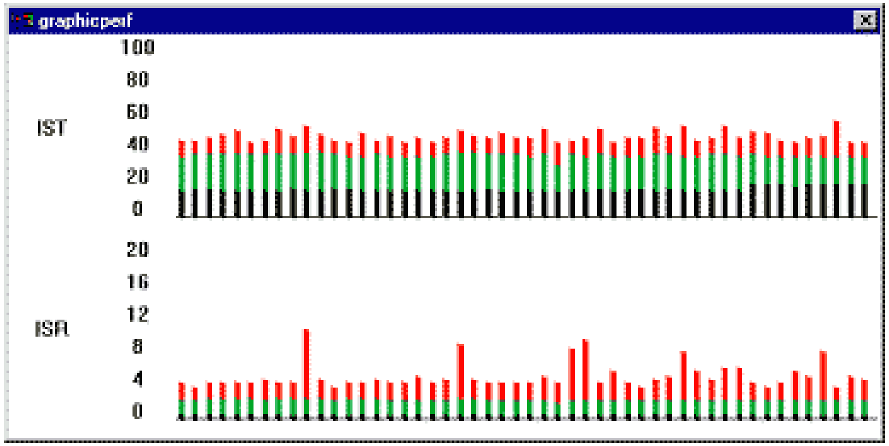
External Latency Measurement

- An ISR is written that generates a measurable output, e.g., setting a parallel port bit high
- The interrupt is triggered repeatedly and the output is recorded on a digital storage oscilloscope in persistent display mode
- Pick latency off the display



Internal Latency Measurement

- Use the programmable timer to down-count to zero from a start count and generate an interrupt
- The timer automatically reloads the start count and continues the down-counting
- The ISR is invoked and reads the timer
- The latency is the start count minus the reading
- WinCE "iltiming" tool does this

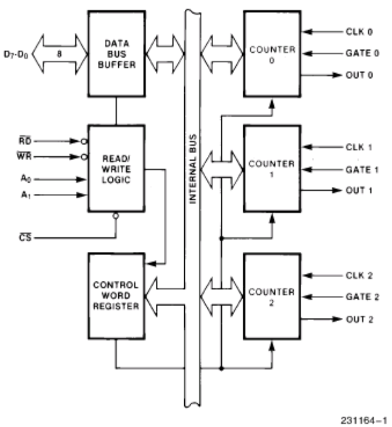


Scheduling Jitter

- *Scheduling jitter* is the variation in actual timing for a periodic task
- Jitter contributors include:
  - hardware effects: the cache
  - software effects: variation in branch instruction lengths in the scheduler
- External measurement technique:
  - a periodic task is written that generates a measurable output
  - the output timing can be analyzed with a hardware timing analyzer, e.g., LeCroy

Internal Jitter Measurement

- See Phil Wilshire's 2nd RTLW paper, "Real-Time Linux: Testing and Evaluation"
- A single RT task is scheduled, which reads the Pentium Time Stamp Counter (TSC) and logs readings into RAM
  - the TSC is a 64-bit integer, incrementing once per clock cycle (2.5 nanosec resolution for a 400 megahertz clock)
- Pure periodic scheduling:
  8254 Programmable Interval Timer (PIT) chip generates an interrupt, the RT scheduler is the interrupt service routine



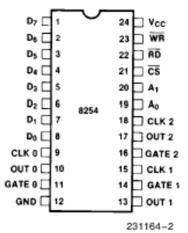


Figure 2. Pin Configuration

- The TSC log is later analyzed for jitter
  - logged values should be exactly one interrupt time apart
  - variations in combined execution time of scheduler and task code will show up as deviations from the nominal

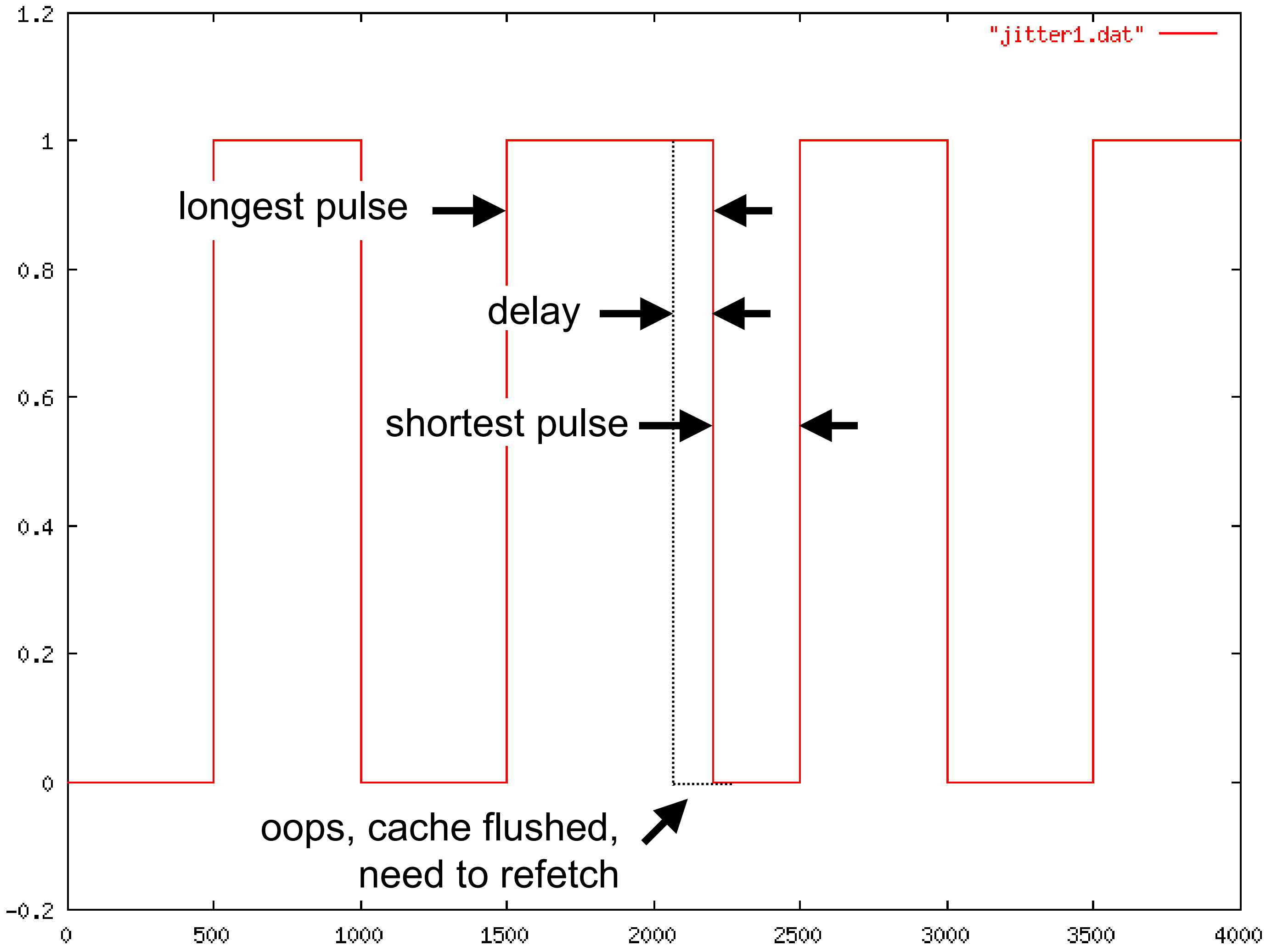
Interpreting Jitter

- If the TSC logging task were a square-wave pulse generator, then jitter would appear as variations in the pulse widths
- Two estimates of maximum jitter can be made
  - ***cycle-to-cycle jitter***: difference between longest and shortest pulse
  - ***period jitter***: largest difference between actual start/end of pulse and nominal expected
  - for the same TSC log, cycle-to-cycle jitter will be about twice the period jitter

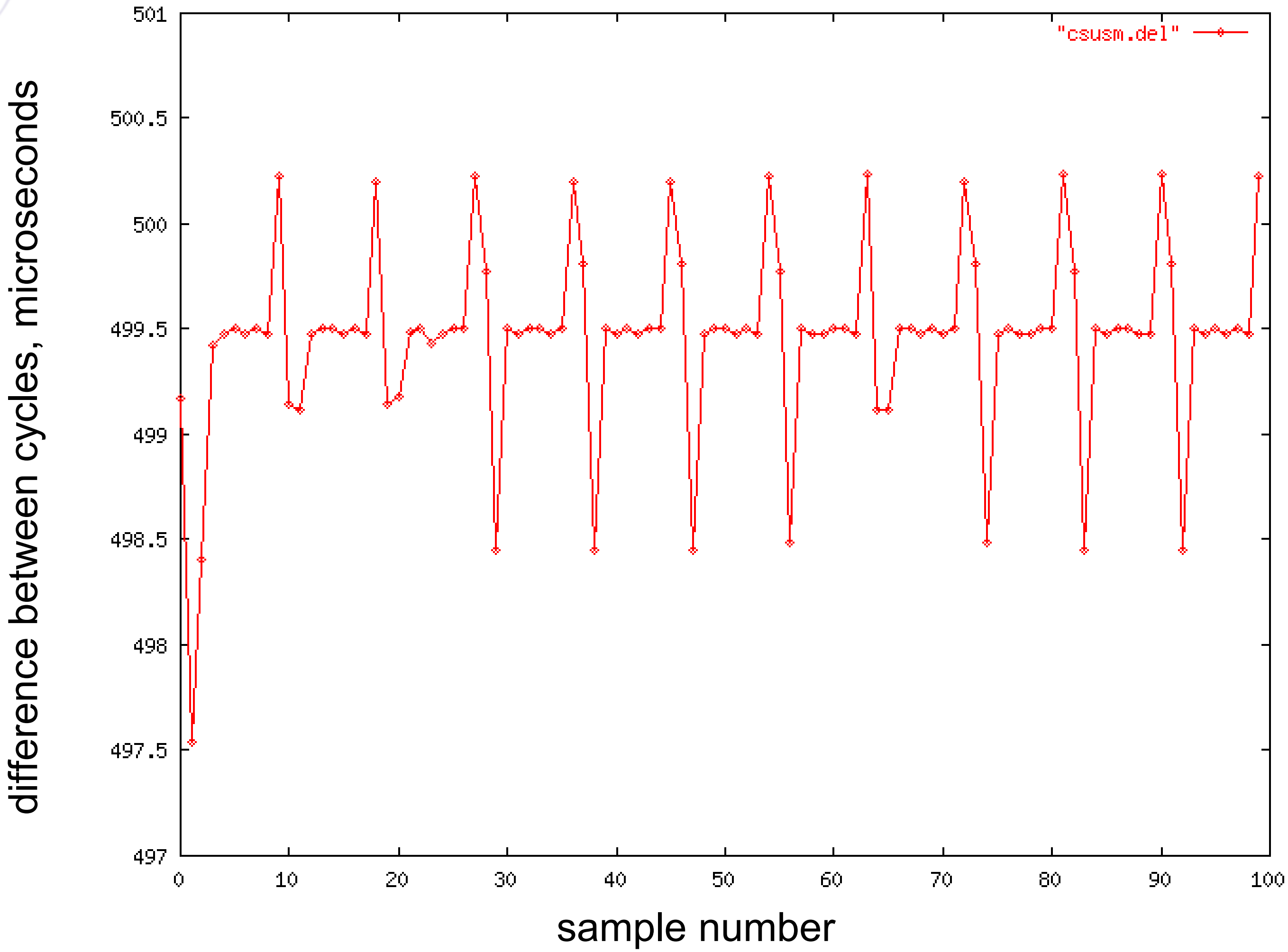
Cycle-to-Cycle Jitter

- Cycle-to-cycle jitter is calculated by differencing adjacent points in the TSC log to get the intervals, then taking the difference between the largest and smallest intervals
- With cycle-to-cycle jitter, a single late task invocation will lengthen one pulse, and shorten the following pulse
- This jitter value is effectively double the scheduling delay
- If relative task timing is important, as for a square wave pulse generator, the cycle-to-cycle jitter value is the most meaningful

Cycle-to-Cycle Jitter



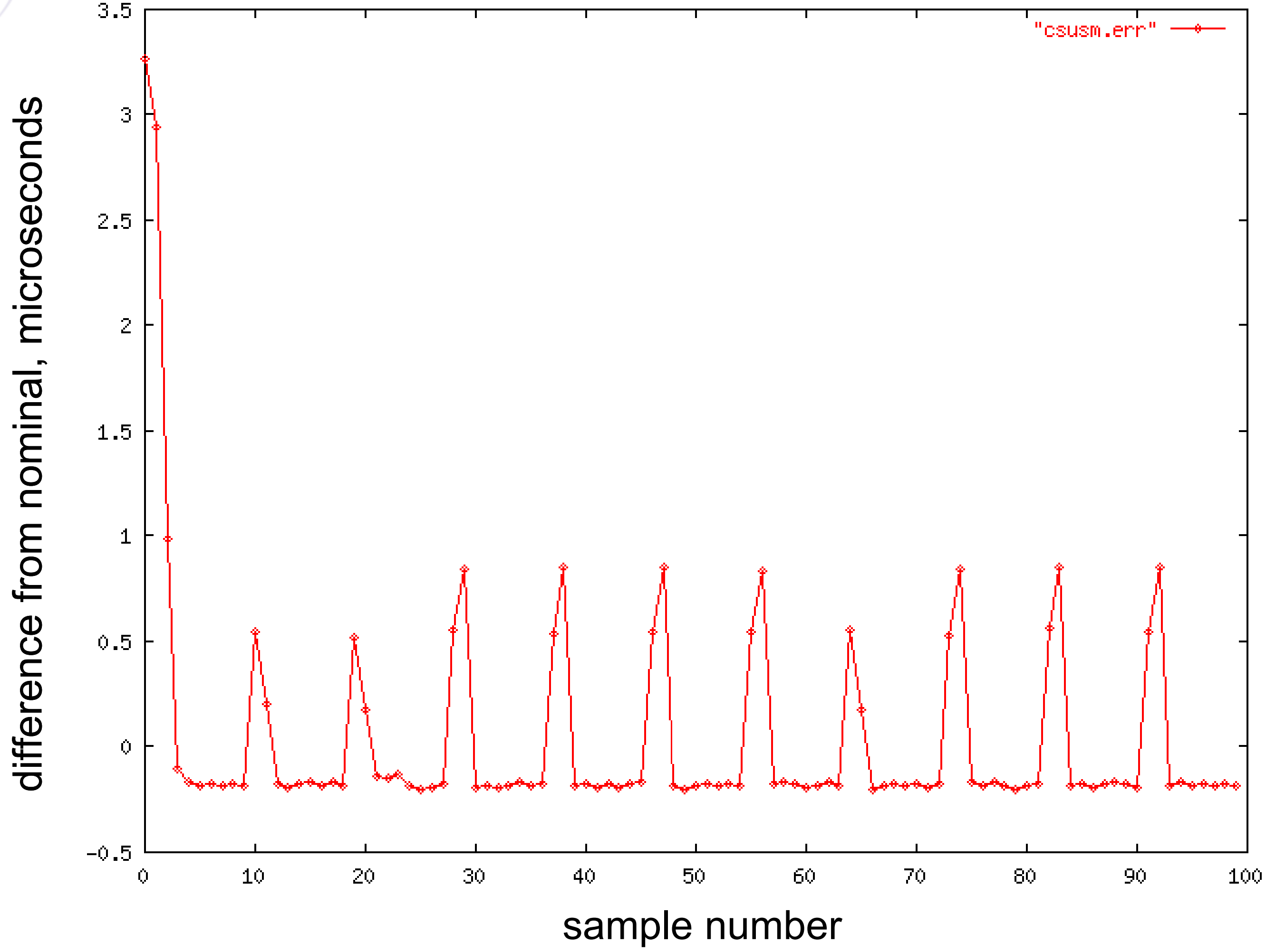
Cycle-to-Cycle Jitter

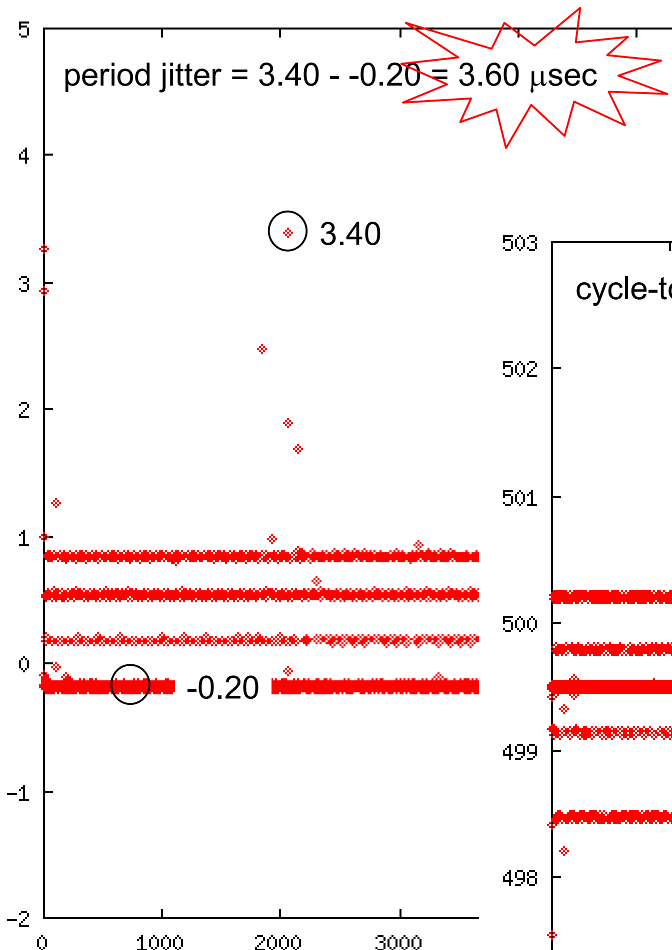


Period Jitter

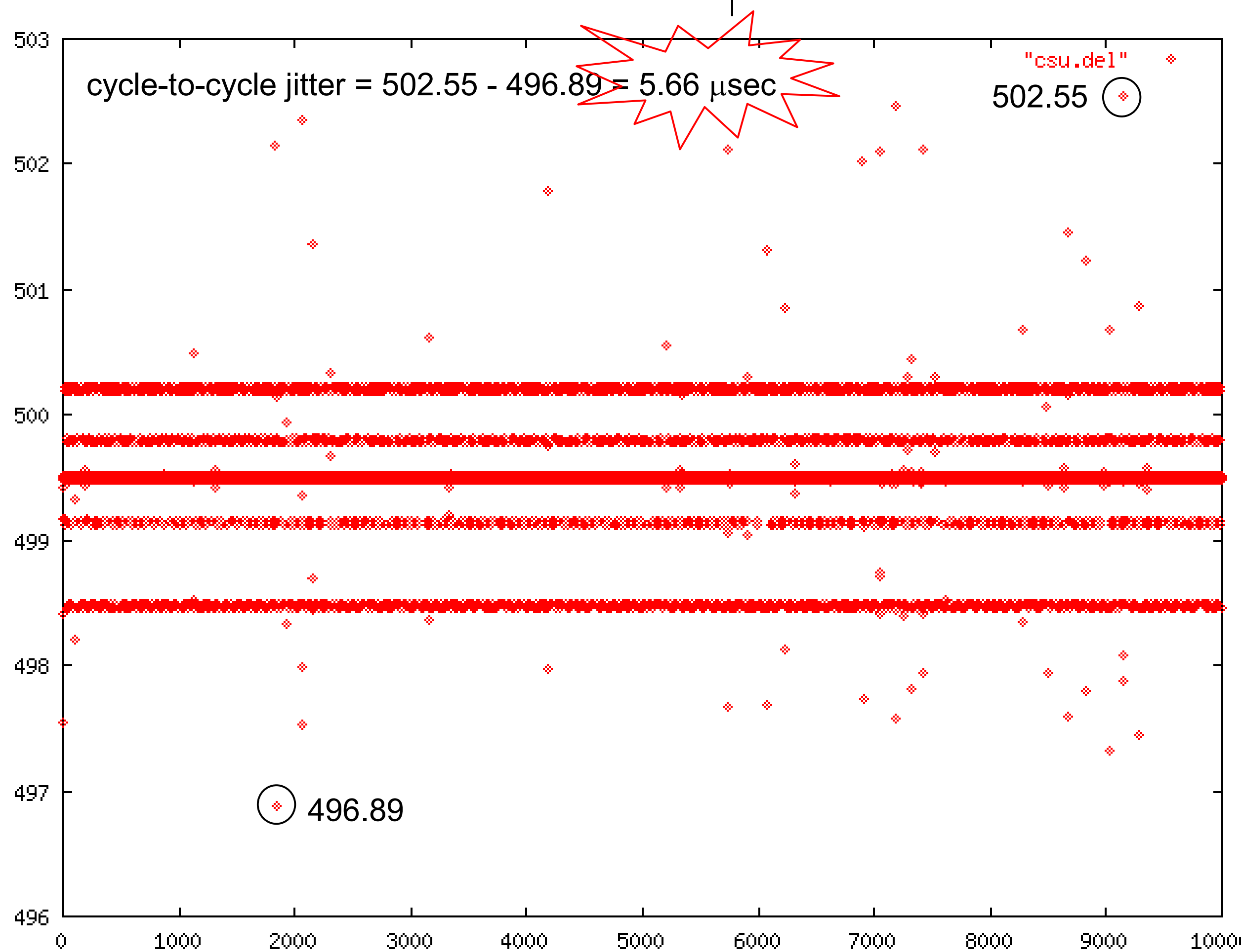
- Period jitter is calculated by computing best-fit line to TSC log values, then taking the difference between the maximum and minimum deviations from this line
- With period jitter, a single late task invocation will penalize only a single pulse; the following pulse will occur on schedule
- This jitter value is effectively equal to the scheduling delay, and is about half the cycle-to-cycle value
- If synchronization with external triggers is important, the period jitter value is the most meaningful

Period Jitter





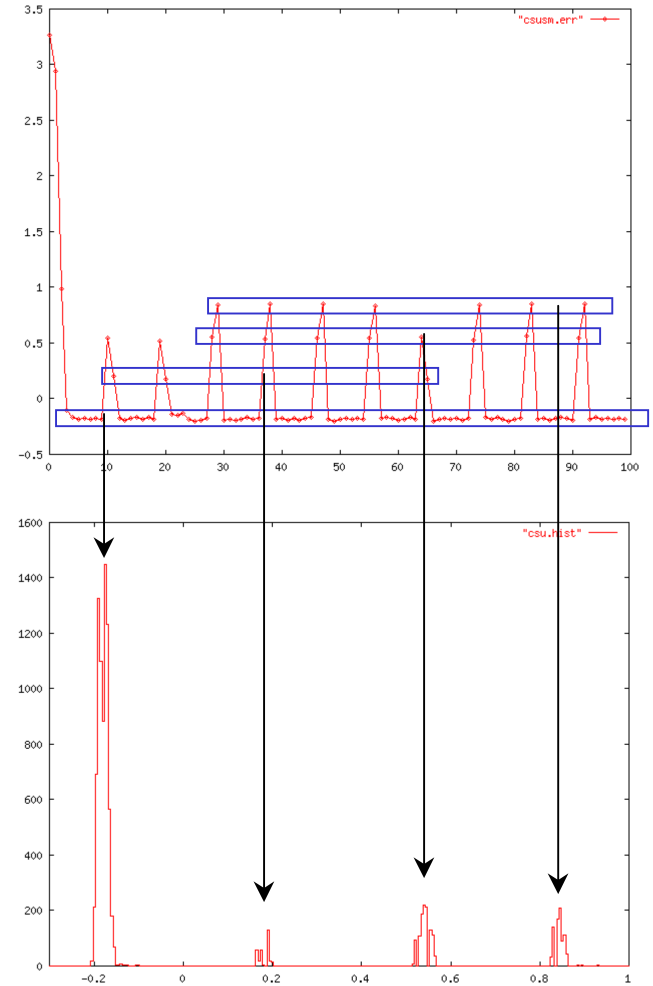
10,000 points logged from 500 µsec task



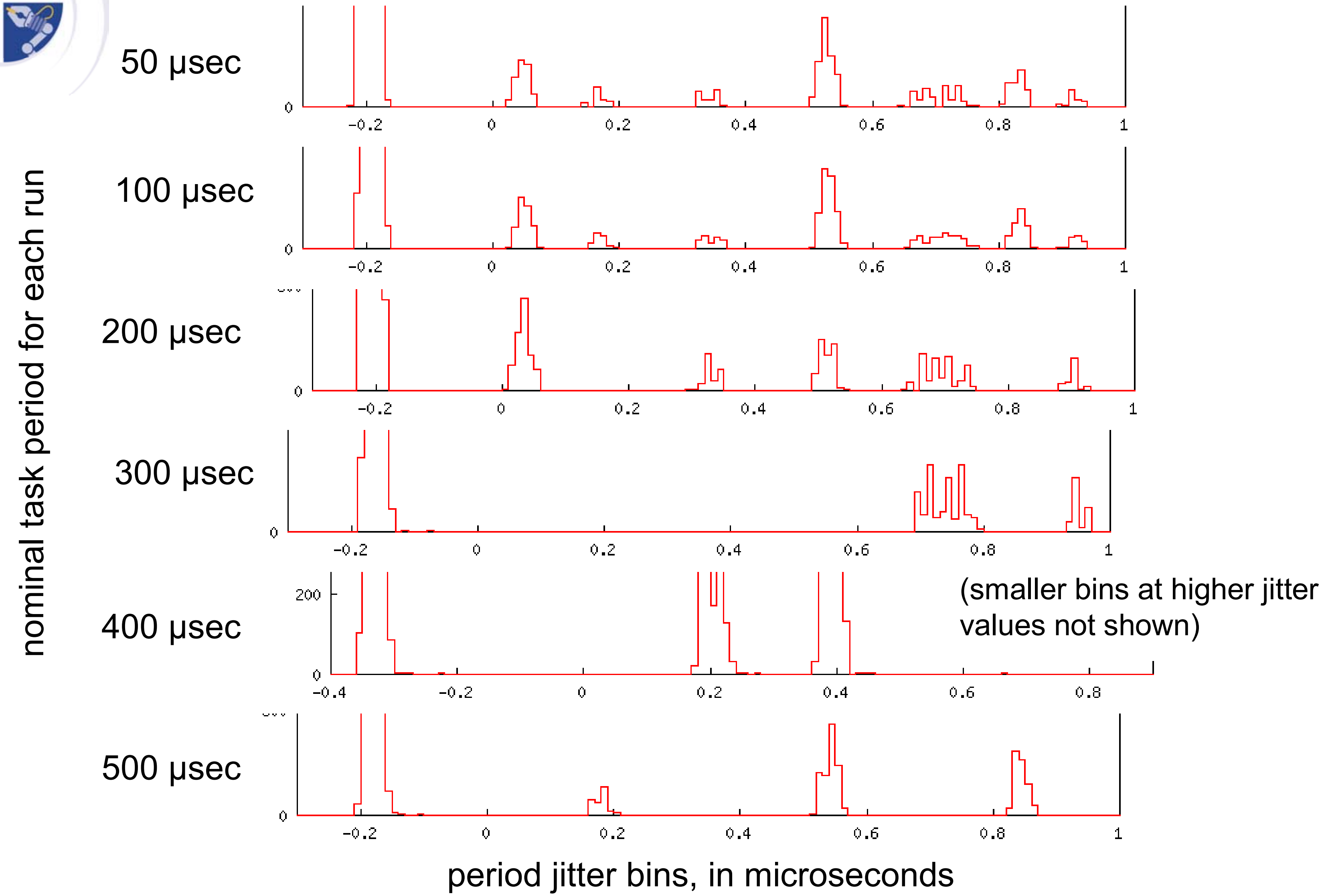
analysis for both plots done from same TSC log data

Jitter Bands

- Bands in the jitter plots indicate a clustering of time stamp deviations
- Histograms of the period jitter values show this clustering more clearly
- Clusters are consistent across different tests, suggesting common origins



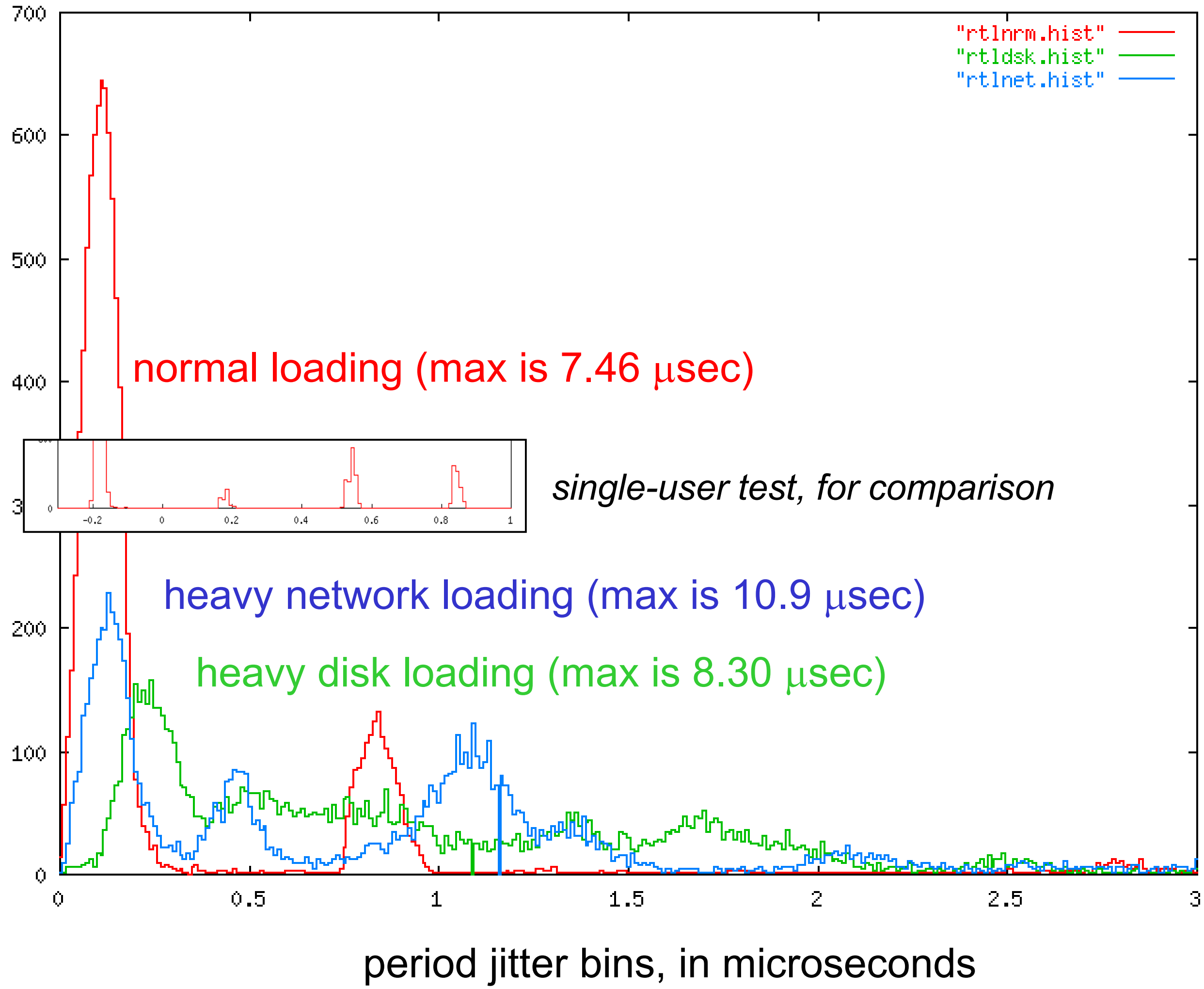
Period Jitter Histograms



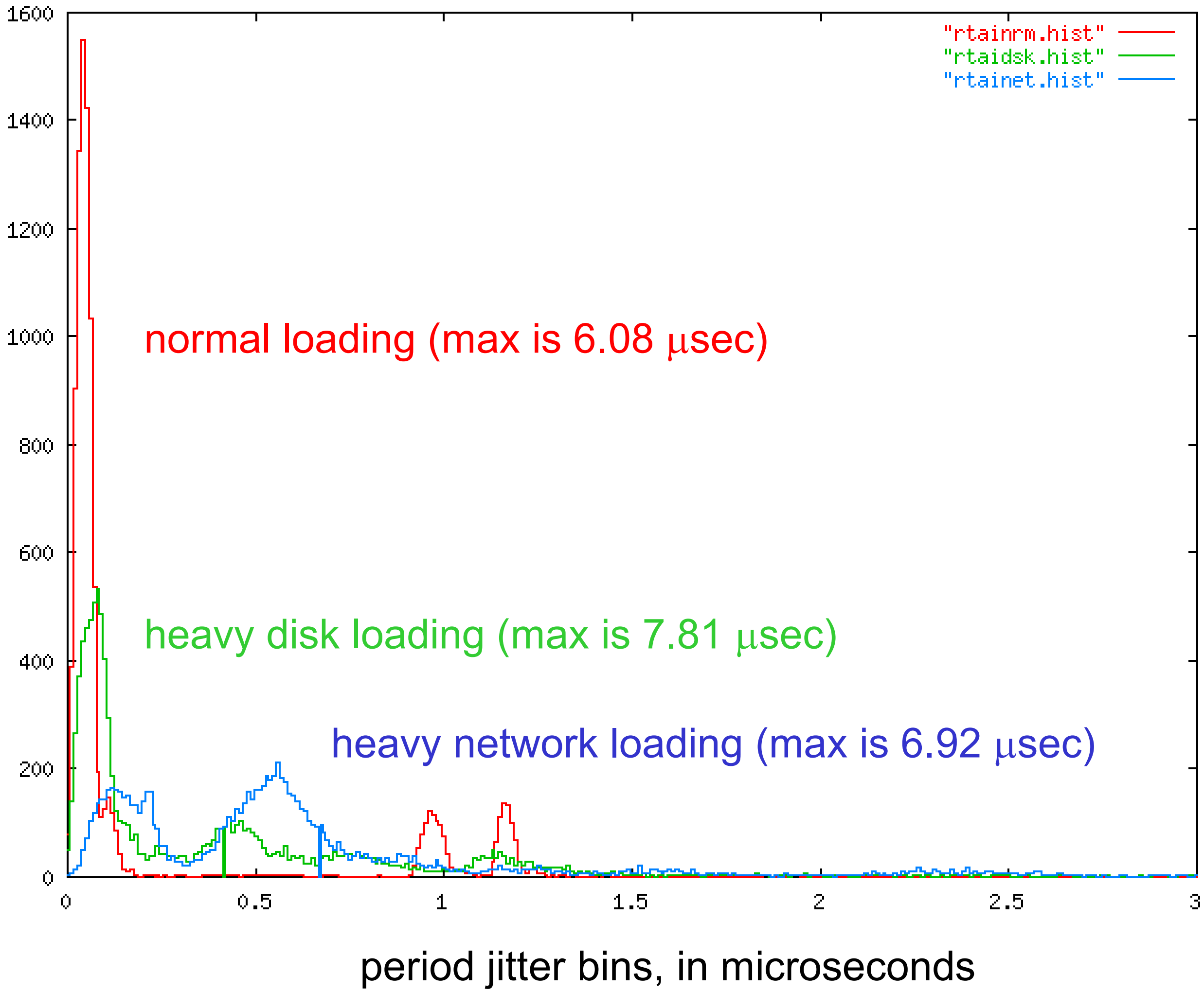
Effects of Processor Load

- As the processor is more heavily loaded, real-time performance will suffer, if only due to cache displacement of RT code
  - the previous jitter measurements were done in single-user mode, with minimal processor loading
  - subsequent measurements of period jitter in loaded conditions shows increased variation
- Surprisingly, for a given task period, faster processors will show slower RT task times
  - more non-RT code runs between RT tasks and dirties up the cache
  - multiprocessor partitioning of RT, non-RT code helps

RTL Loading


"rtlnrm.hist"
"rtldsk.hist"
"rtlnet.hist"
normal loading (max is 7.46 µsec)
single-user test, for comparison
heavy network loading (max is 10.9 µsec)
heavy disk loading (max is 8.30 µsec)
0
100
200
400
500
600
700
0
0.5
1
1.5
2
2.5
3
-0.2
0
0.2
0.4
0.6
0.8
1
period jitter bins, in microseconds

RTAI Loading



A Method to Reduce Jitter

- TSC can be used to reduce jitter, as proposed by Tomasz Motylewski of the University of Basel
- A series of subtasks polls the TSC for the precise instant that the time-critical code should execute
  - most subtasks return immediately, since target TSC is farther in future than the subtask period
  - the final subtask cycle polls the TSC until the target is reached
- CPU load depends on time to service subtasks, and time spent polling
  - more frequent subtasks incur too much overhead from null cycles
  - less frequent subtasks incur too much polling during final cycle

Optimal Subtask Scheduling

Load analysis:

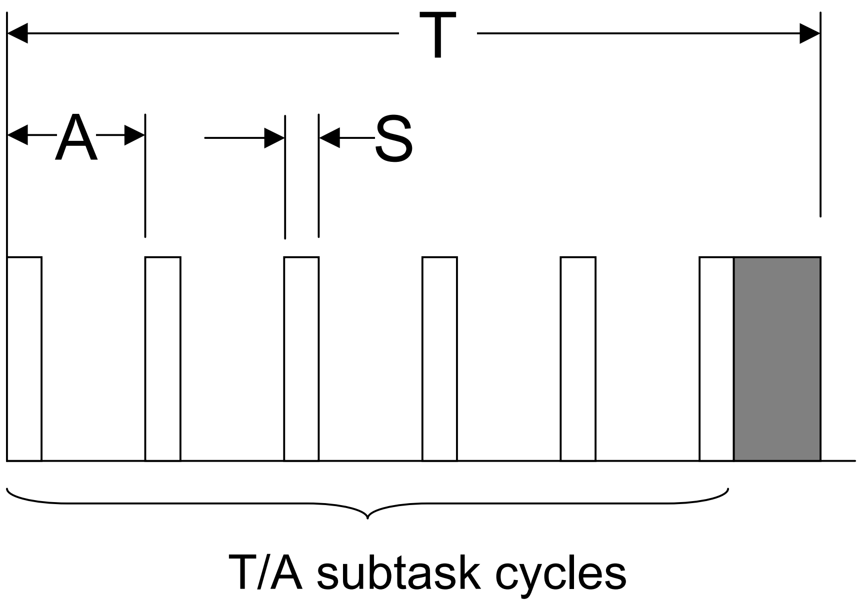
1. T/A subtask cycles
2. T/A-1 null cycles, 1 polling cycle
3. Time to service null cycles is (T/A-1) * S
4. Worst case poll time is A
5. Load is

$$load = \frac{(T/A-1)S + A}{T}$$

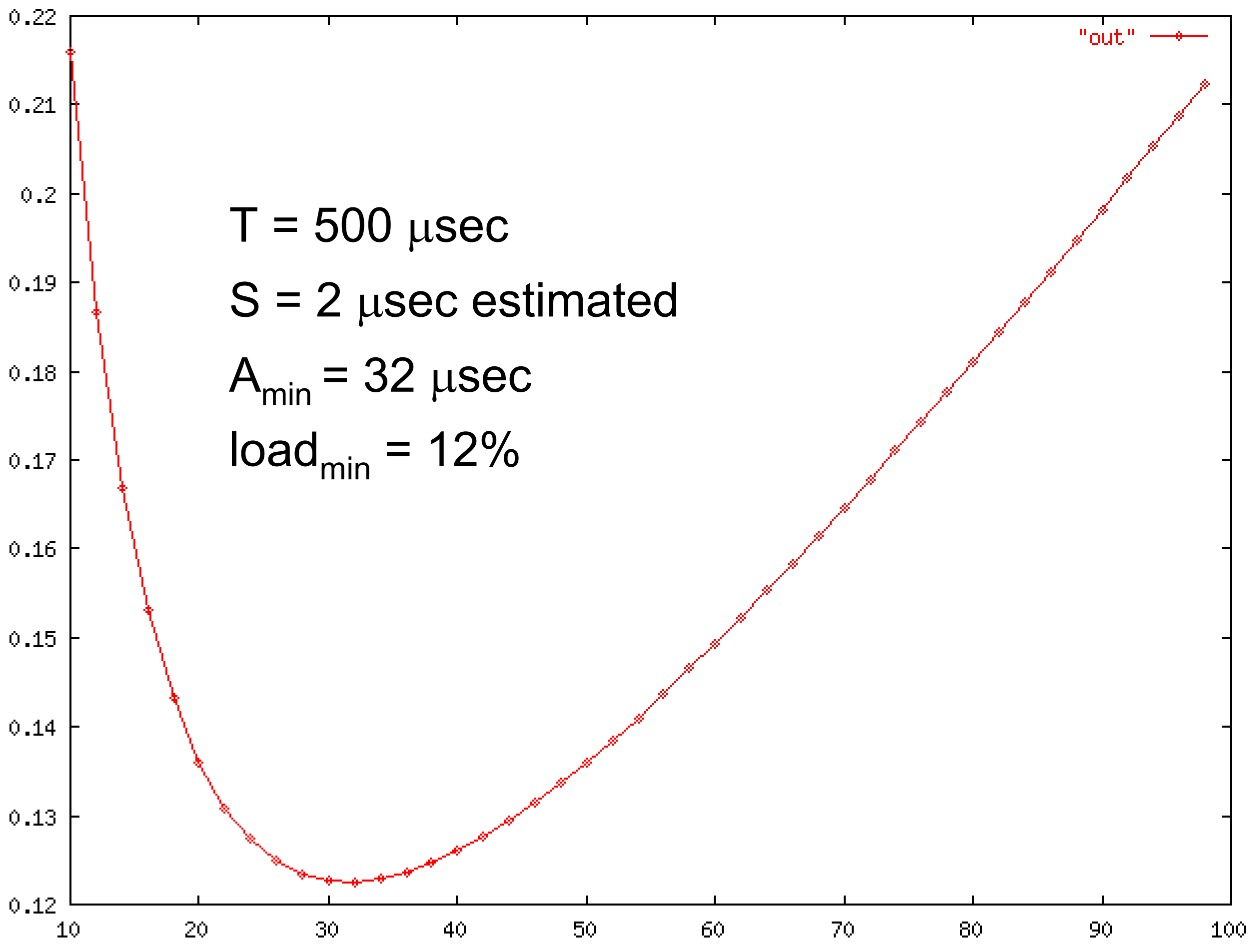
6. Minimizing with respect to A:

$$A_{\min} = \sqrt{ST}$$

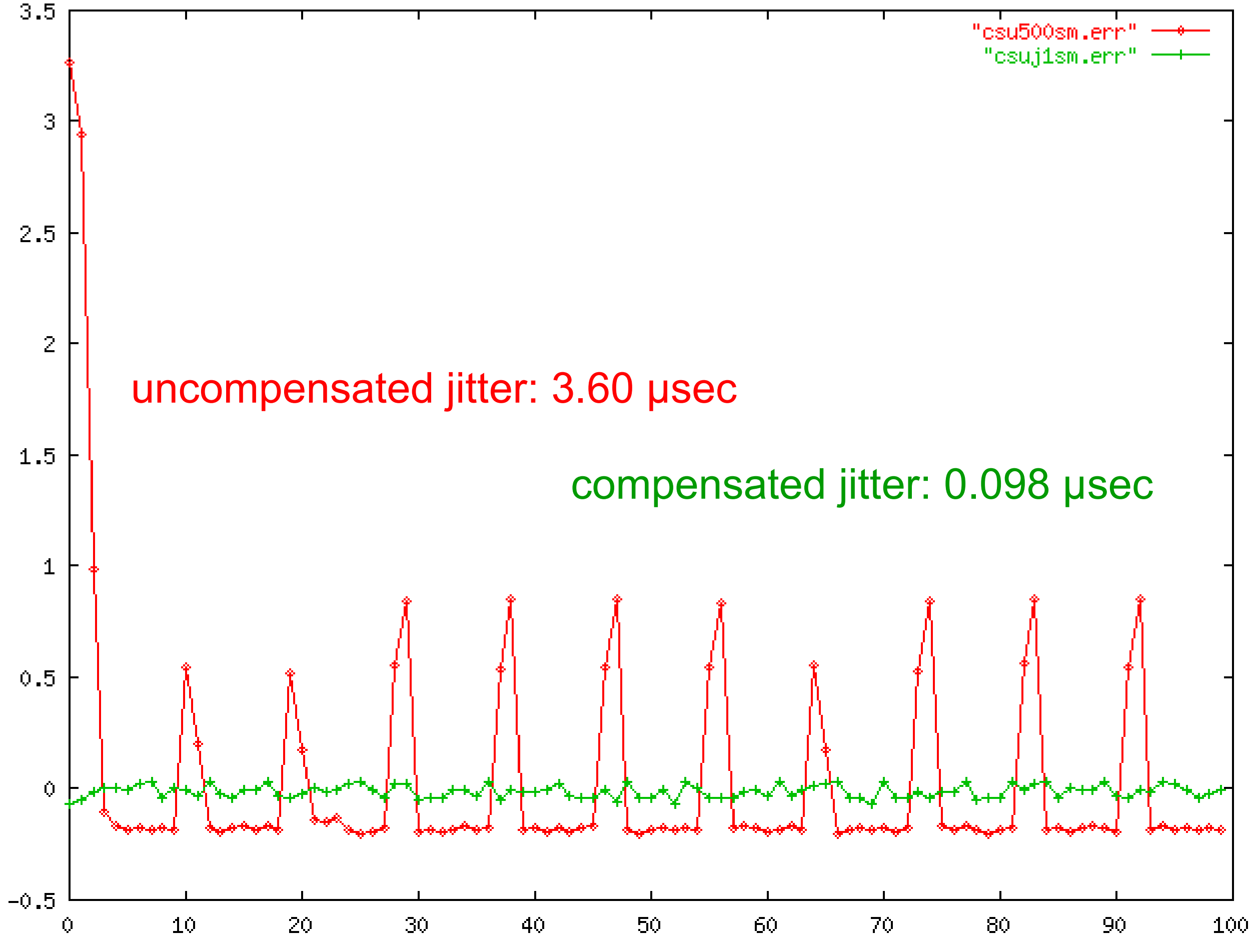
$$load_{\min} = \frac{2\sqrt{ST} - S}{T}$$



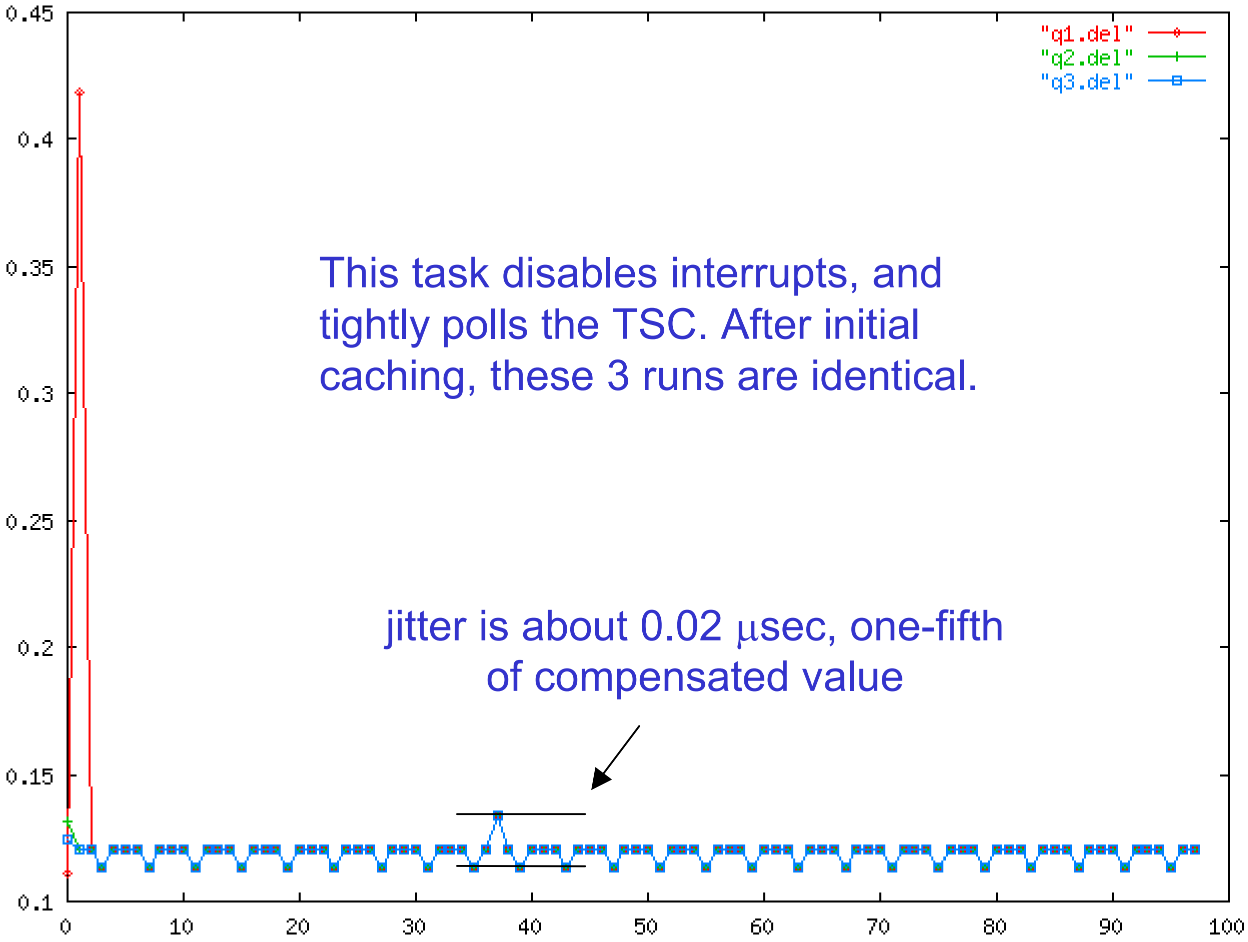
Optimal Example



Compensated Jitter



Best Case



0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0
10
20
30
40
50
60
70
80
90
100
"q1.del"
"q2.del"
"q3.del"
This task disables interrupts, and tightly polls the TSC. After initial caching, these 3 runs are identical.
jitter is about 0.02 μsec, one-fifth of compensated value

Summary

- Performance measures answer the question, "How can I tell that a real-time operating system is able to satisfy my application's timing requirements?"
- Classic measures include interrupt service routine latency and scheduling jitter
- Both external and internal techniques can be used to measure these
- The testing environment is important if results are to be compared
- Internal techniques can be adapted to reduce scheduling jitter at the expense of processor time