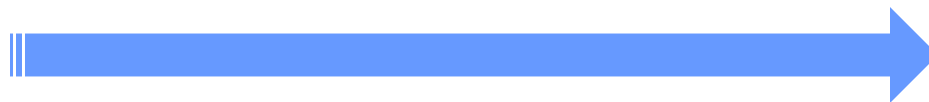
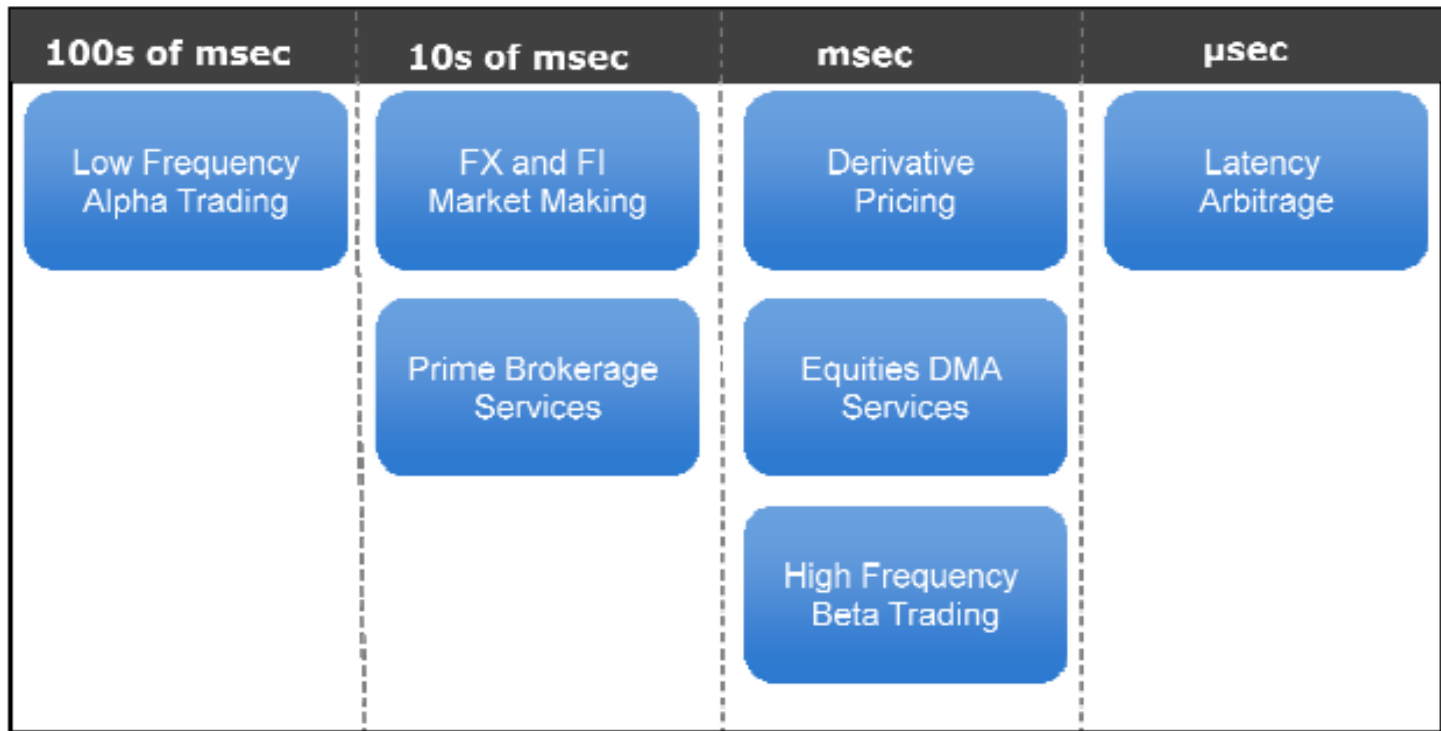


LOW LATENCY FRAMEWORK

- It may be necessary to lower latency just to retain a Market making position
- It may be desirable to improve latency to improve ones ranking or to enter a new market
- It may be desirable to improve latency to stop getting picked off by competitors
- Traditionally limited to Equities but now becoming critical for FX and other OTC's
- The value of lower latency is intrinsically understood but extremely difficult to quantify
- Lower latency systems cost more to build and deploy
 - Challenge is to find the right balance between investment in lowering latency and obtaining a return on this investment
 - Some of this cost can be shared by adopting standardized low latency solutions



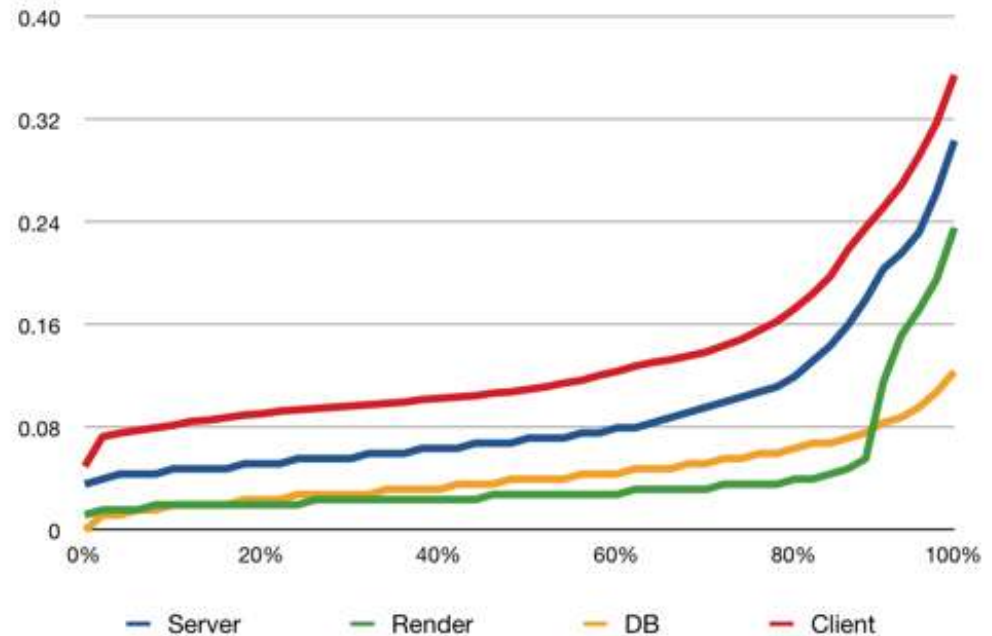
Source: Donovan Ransome, Citihub, 2009

- Unconstrained demand
 - Worsened as network pipes have got fatter
- All markets are growing both in volume and # of trades
 - High frequency and Algo trading now accounts for around 80% of total trading¹ and growing
 - Average Equity trade value has decreased as bigger trades are automatically segmented to reduce market impact
 - # of cancellations increased as Traders/Algo's seek to discover the book
 - Increasing sophistication of algorithms makes it more difficult to predict behaviour
- Everybody bursts at the same time
 - Produces biggest trading opportunities – need to stay in the market
 - Greatest risk of loss if fail to keep up – may get picked off
- Developer productivity improvements (e.g. Java and .NET) has made the control and visibility of the underlying infrastructure more difficult
- Distributed computer architectures have more complex performance and reliability characteristics
- Need to increase throughput and lower latency simultaneously
- Can no longer rely on Moore's law to save us – only delivering more cores at the same speed, not getting any faster

Note 1: High frequency trading firms account for 73% of all US equity trading volume, [AdvancedTrading.com](#), 2009.
FX has been reported as having about 25% of orders automatically traded in 2006

Burst capacity and computer meltdown

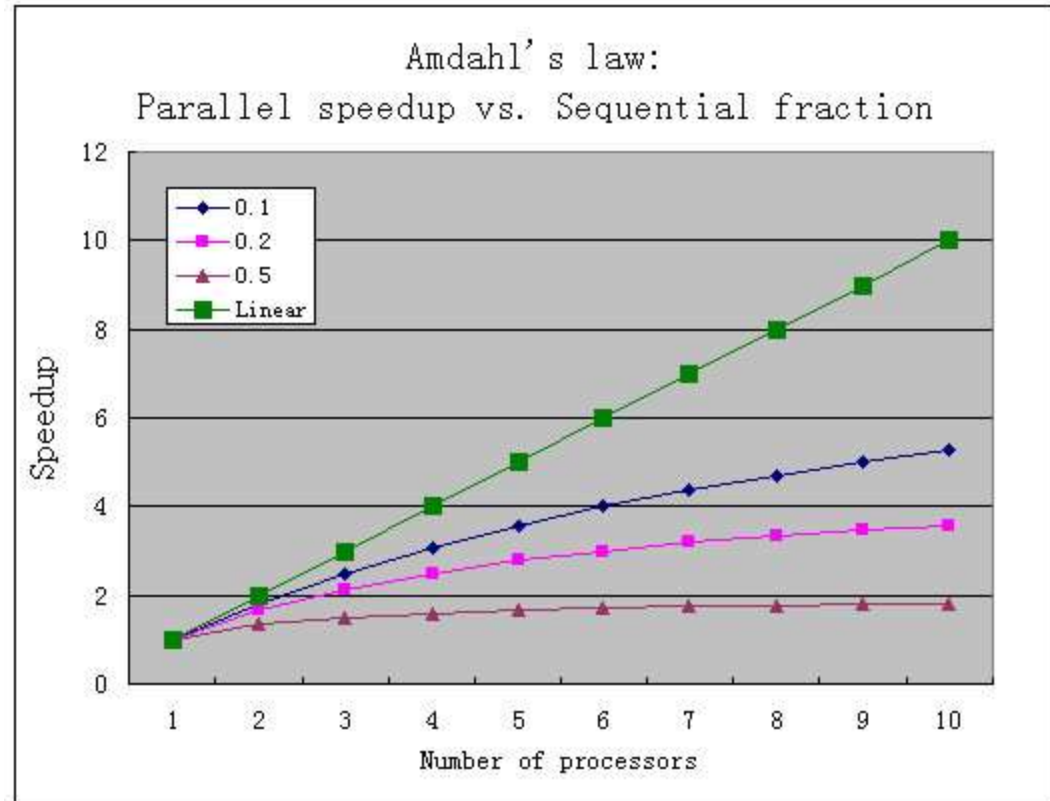
- We need massive headroom free to allow for bursts
 - typical guide is 10:1
- But computers degrade in a non linear fashion
- Can be triggered by a shortage of any one of it's resources:
 - CPU cycles
 - Memory
 - I/O channel capacity
 - Disk transfers
 - Network transfers
- Shortage of one can trigger another e.g. shortage of memory causing CPU and I/O thrashing
 - Distributed deployments add an order of magnitude more complexity



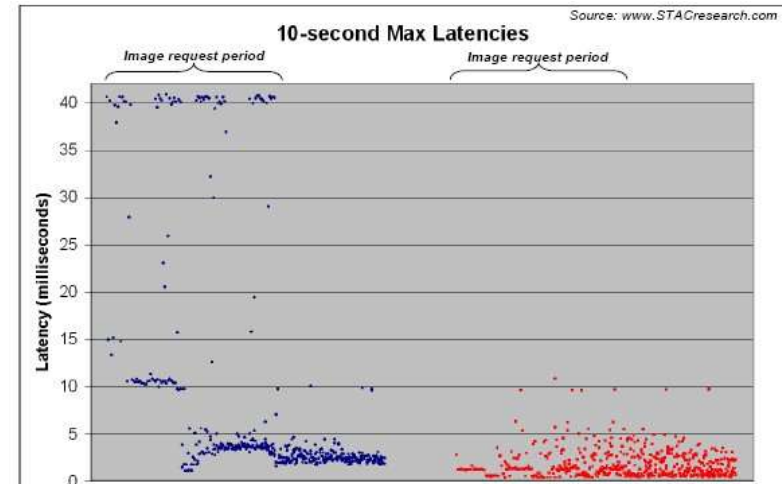
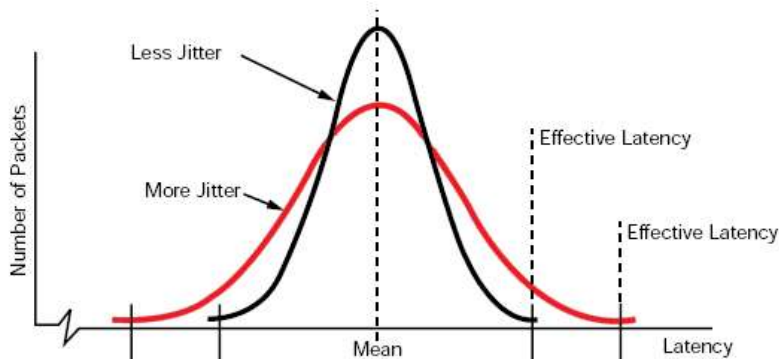
Typical system response time against throughput

Suffering consequences of Amdahl's Law

- New CPU's are just giving us more cores at a similar speed
 - Stopped getting faster at 60nm geometries
- Dependant on multi-threading capability of the application to utilize additional cores
- Sequential code streams increasingly dominate latency
- Resulting in additional cores providing diminishing benefit
- Sacrificing cores for speed may improve latency in some circumstances e.g. Intel TurboBoost but power saving measures introduce unpredictability



- Whilst there is a lot of attention to latency, the consequences of jitter can be more damaging e.g.
 - A single lost message can wipe out all profits from hours of successful trading
- Need to reduce both latency and jitter
- Requires latency distribution analysis and goals around median not minimum latencies attained



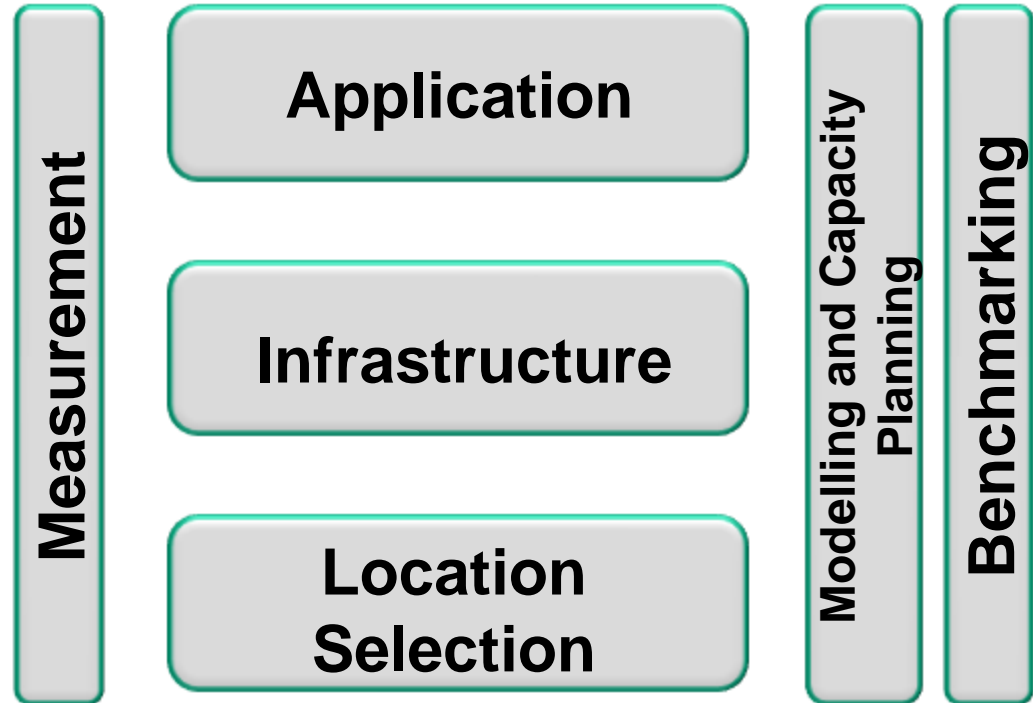
Reducing latency shifts the mean to the left

Reducing jitter tightens the distribution and reduces the range of possible values

- Design it in – requires proven components and methodologies
- Testing to identify the insertion latency of each component in existing systems together with end-to-end latency
- Baseline measurement so that improvements can be tracked and new functionality does not cause surprise differences in latency
- Model infrastructure improvements before committing to big spending
- Capture and share best practice development methods
- Identify best of breed components and tune for low latency
- Pick the best location and deployment strategy

Requires a structured approach and framework

- Recognizes latency contribution from all layers
- Multiple disciplines and SME's
- Develops and shares expertise and tooling around low latency measurement, modelling and benchmarking
- Supports low latency design



Component breakdown of latency

- Latency can be broken down into the following components
 - $L = P + N + S + I + AP$
- **P** = Propagation time - sending the bits along the wire, speed of light constrained
- **N** = Network packet processing – routing, switching and protection
- **S** = Serialization time - pulling the bits off the wire
- **I** = Interrupt handling time – receiving the packet on a server
- **AP** = Application Processing time

- In most situations it involves a data message exchange between a minimum of 2 systems i.e.
 - $L = AP^1 + I^1 + S^1 + N^1 + P + N^2 + S^2 + I^2 + AP^2$
- This can be expanded to add more hops as required

- We can break this down into three groups for optimization:
 - Distance: **P**
 - Infrastructure: **I, N, S**
 - Application: **AP**

Typical round trip latency
cost of routing over a
long line

e.g. for 1000bytes at 1Gbps
(including access network)

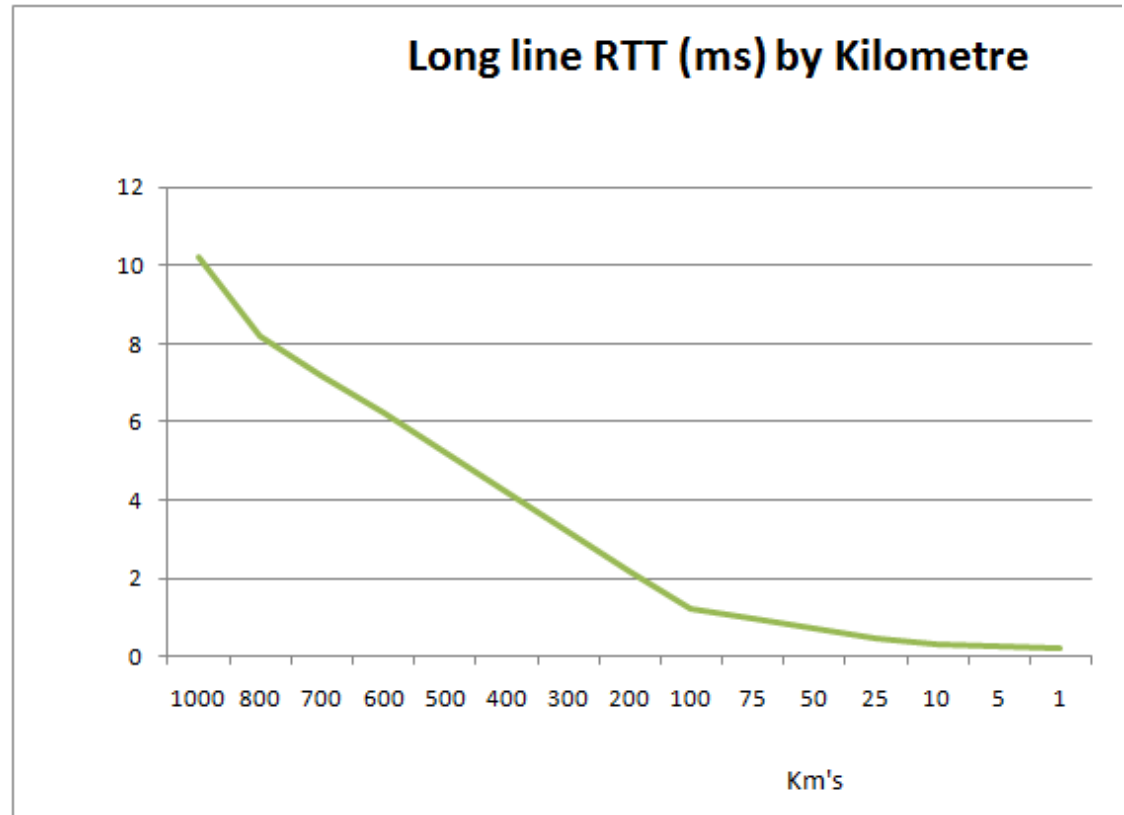
1000km = 10.1mS

100km = 1.2mS

10km = 303µS

1km = 213µS

0km = 203µS



Modelled using distance, packet size, bandwidth, serialization, router delay, server delay
Router and serialization latency dominate as distance decreases
Stretched VLAN can be used to remove router insertion latency cost

Low Latency is relative

Goal is for Infrastructure to account for 10-20% of Application Latency

	Latency tolerant	Low Latency	Ultra Low latency
Application (end-to-end including middleware and run Time environments)	> 10 mS	< 10 mS	< 500 μ S
Infrastructure (LAN, Server, storage, OS)	> 1 mS	< 1 mS	< 50 μ S
Propagation Location (CoLo), Carrier and path selection			

The latency cost of sending a packet

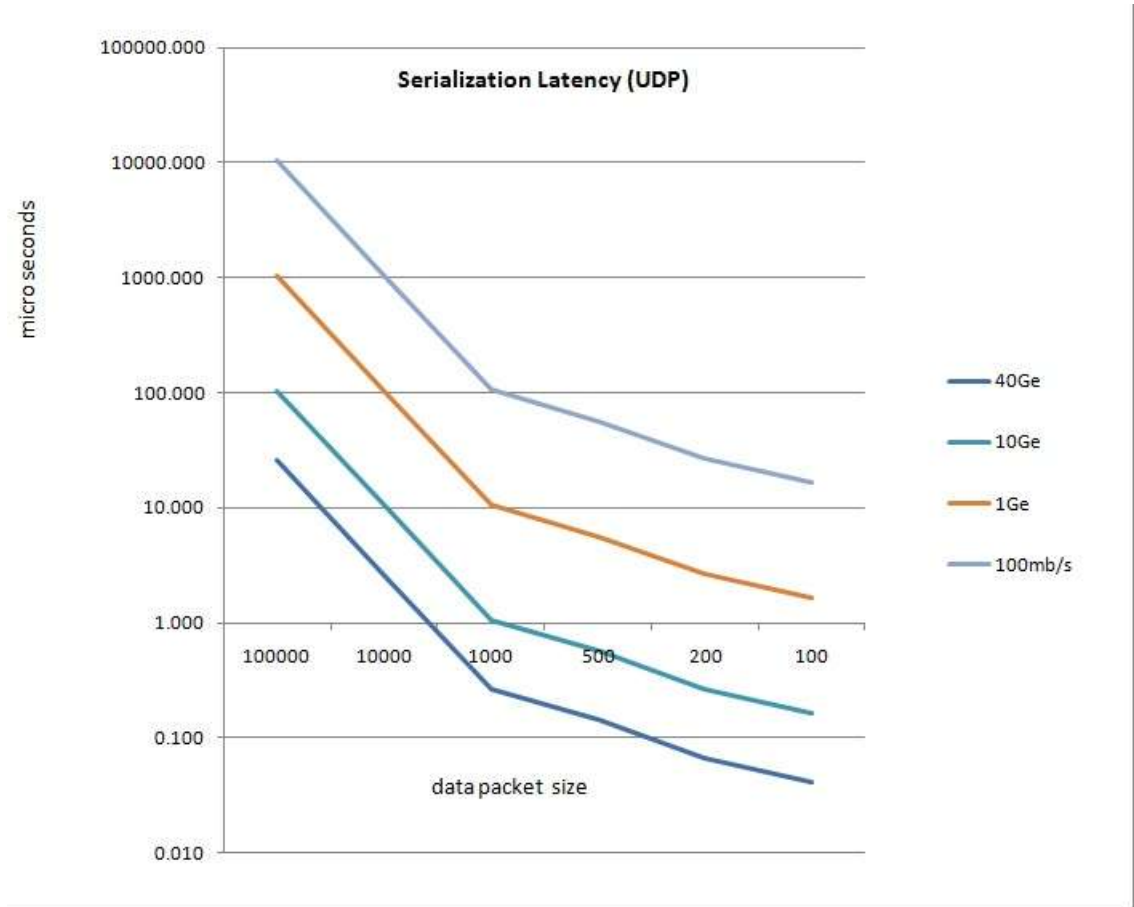
e.g for 500byte packet

10Mb/s = 566μS

100Mb/s = 56μS

1Gb/s = 5.7μS

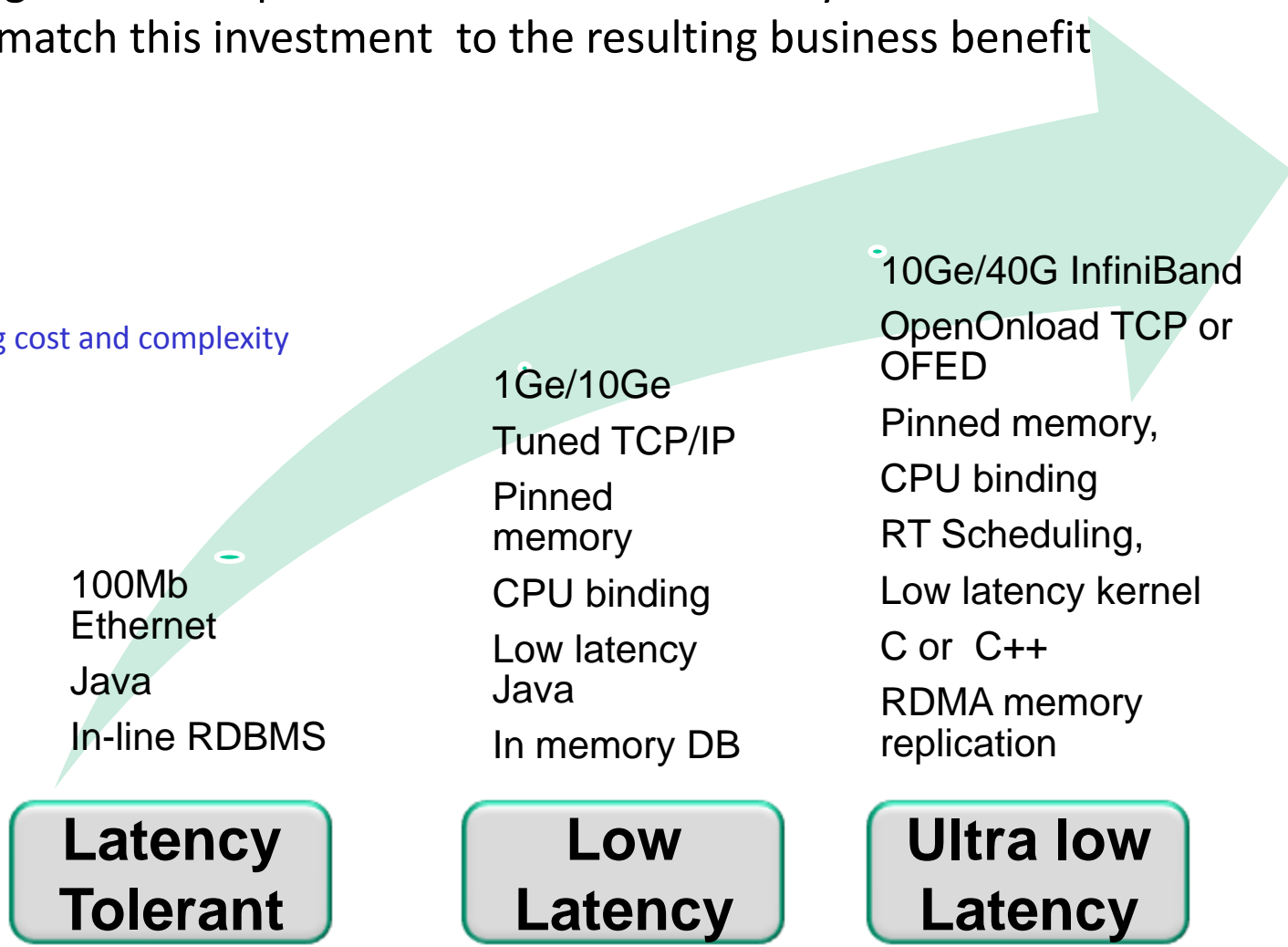
10Gb/s = 0.57μS



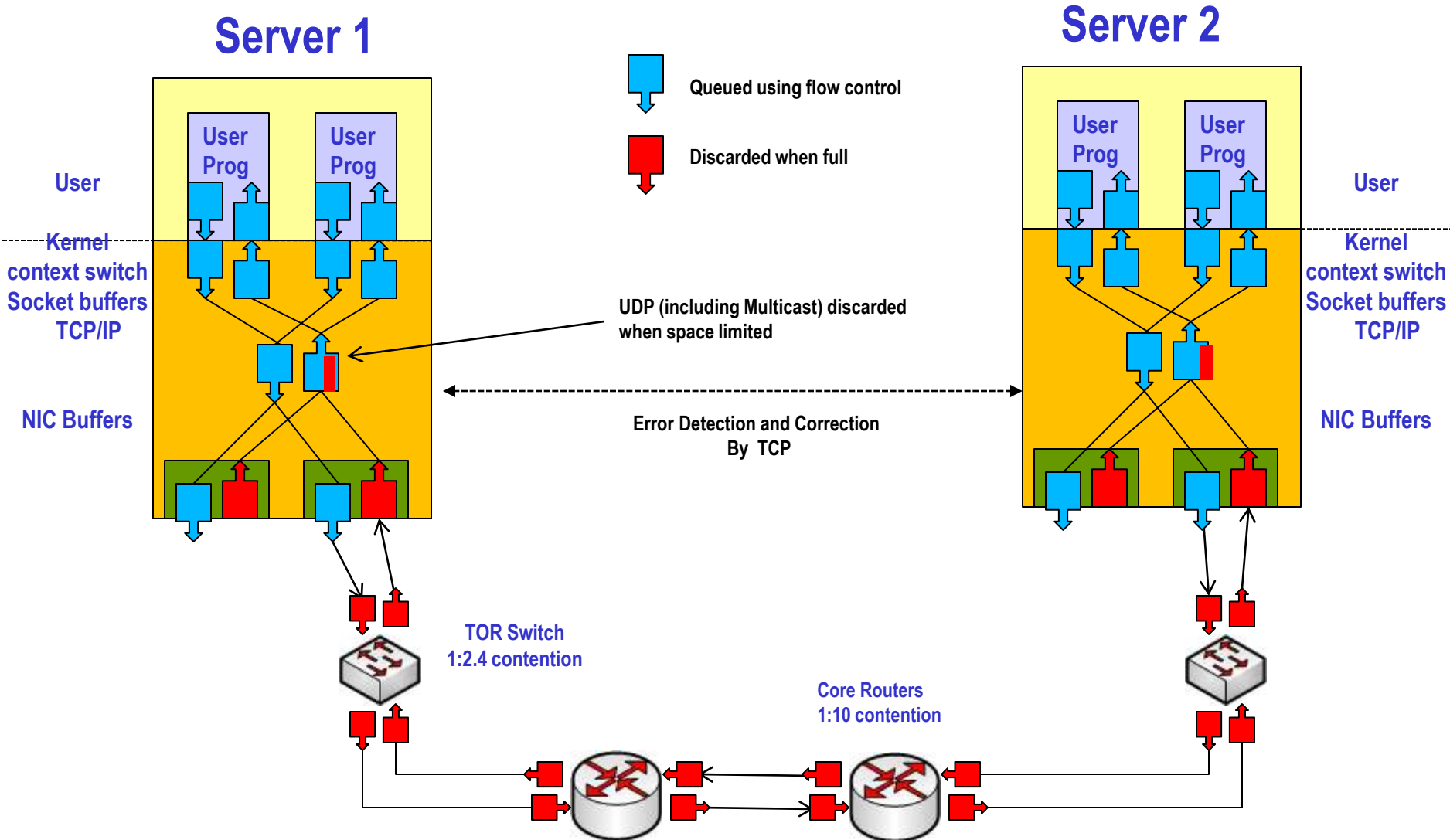
Serialization latency is added at every server and most network ports (exception is cut-through)

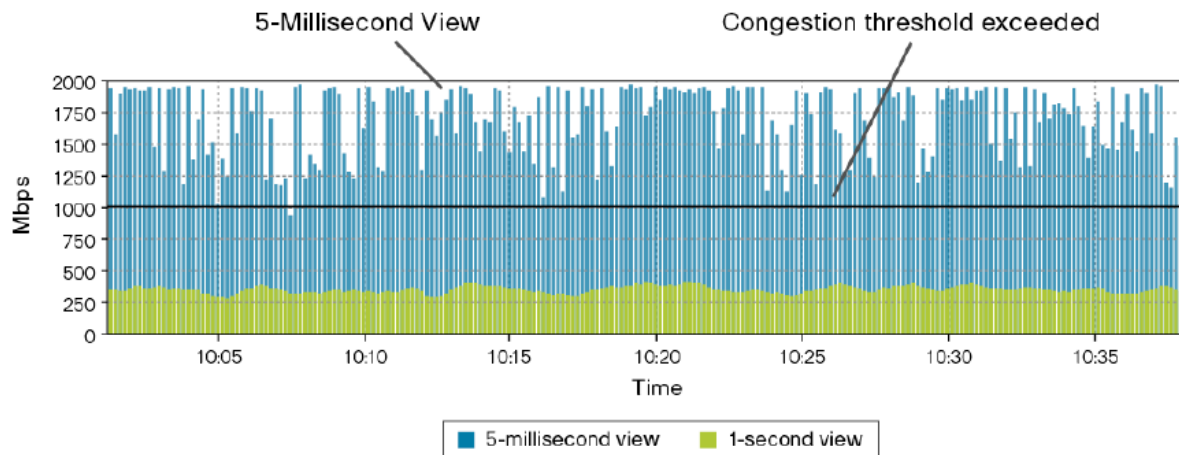
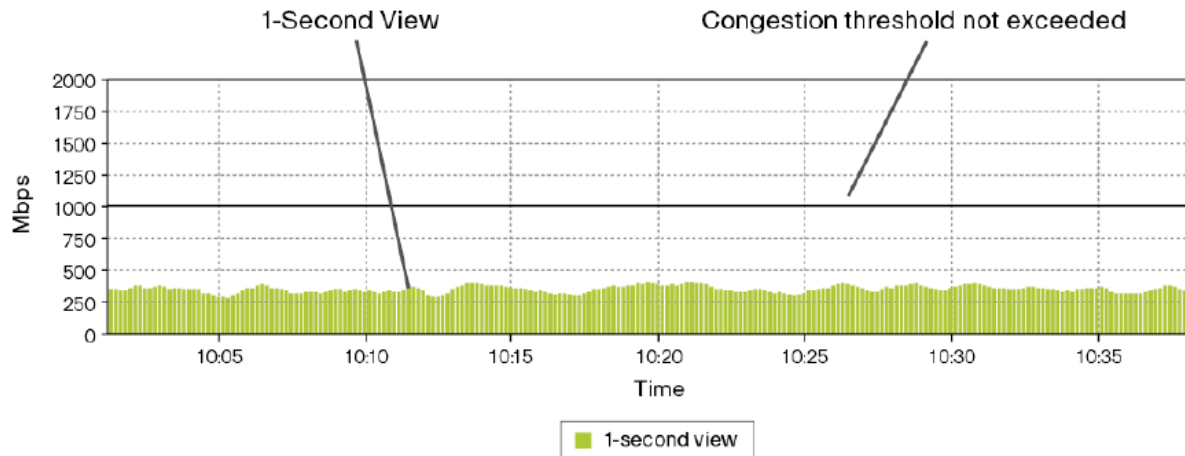
Increasing levels of sophistication to reduce latency to the minimum
Need to match this investment to the resulting business benefit

Increasing cost and complexity



- Optimal carrier selection to optimize propagation latency
- Select high line speed to reduce serialization
- Collapse network to reduce number of network devices in path
 - Increase L2 size and span, reduce # of L3 hops
- Optimize selection of network devices
 - Cut through switches
 - Low latency routers with good Non Drop Rates (NDR)
- Plan to handle microbursts
- Managing drops and discards to reduce jitter





Courtesy of Cisco

- Do not run under any hypervisor
- Where a choice is possible select the lowest latency Operating System platform
 - Linux, Solaris, Windows
- Use fastest cores available
 - i.e. Currently x86 rather than SPARC
 - Consider Turbo boost or sacrificing cores e.g. Everest
 - Decide where to trade between speed and reliability
 - E.g. Water cooled X5680 running at 4.4Ghz
- Use fastest network connection to reduce serialization delays and reduce likelihood of dropped packets - 1Gbp/s minimum
 - Consider InfiniBand as alternative to Ethernet between critical servers
- Minimize network contention between critical servers to reduce likelihood of dropped packets and flatten networks by reducing hops
- OS's are tuned by default for throughput, re-tune for low latency
- Tune the TCP/IP network stack or bypass it altogether
- Optimize program runtime
 - Prioritize critical applications and minimize running system by disabling unwanted services
 - Interrupt masking and allocation
 - Pinning to Cores and locking memory

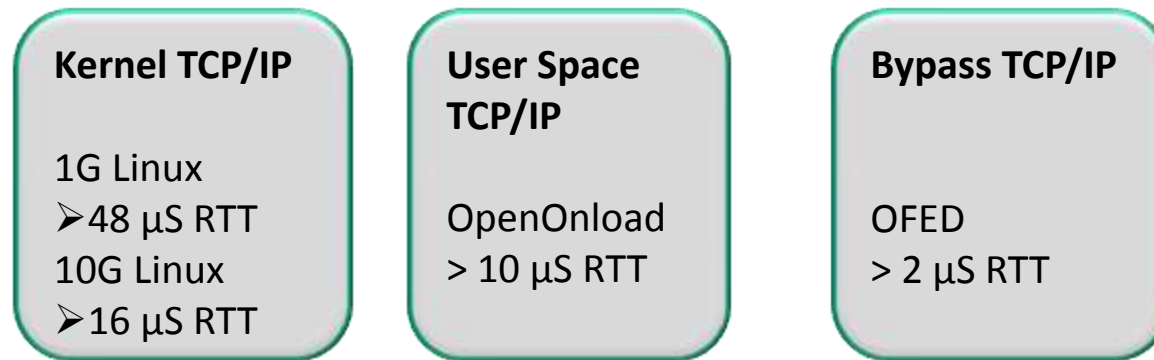
- Minimize to actual required Services, saves memory, swap page and occasional re-scheduling events
 - also has security advantages by reducing attack envelope
- Don't start X11 and desktop services
 - Best option is don't install
 - e.g. Desktop Gnome RPM on Red Hat
 - Often left on for troubleshooting and making changes
 - Make init level 3 the default rather than level 5 which is usually used to start this /etc/inittab
 - *id:3:initdefault:*
- Other services you (probably) don't need are:
 - iptables, ip6tables, yum-updates, sendmail, bluetooth, cups, irda, atd, autofs, hidd, kudzu, smmbfs.....
- Remove unnecessary cron jobs and make sure others don't impact critical Trading periods

- Minimize jitter and improve average latency
- Solaris
 - Assign latency critical processes into RT Scheduling class
 - Provides fixed priority higher than all other classes
 - Keep < # of logical CPU's to ensure OS services not completely stalled
- Linux
 - Enable kernel pre-emption
 - CFS fair queuing scheduling algorithm, merged into 2.6.23
 - Timeline based future task execution with nanosecond granularity accounting
 - Set critical tasks with lower decay factor to retain priority or assign to RT sched class

- Avoid it, or at least move it out of critical code paths
 - Achieve durability by replicating critical data across the network, 100's of times quicker than a disk I/O
 - In Memory caches
 - Apache Memcached – read only, for subsequent reads
 - Cacheonix , Oracle Coherence, GemStone, GigaSpaces, RNA networks, TIBCO ActiveSpaces, Terracotta
- Tune Storage subsystem
 - Heavily cached storage for random I/O and write mostly
 - Linux, disable elevator re-order algorithm
 - Boot time parameter - **elevator == noop**
 - FLASH drives primarily for random I/O
 - PCIe card bases solutions provide better b/w (e.g. Fusion-IO, ioExtreme)
- Tune File System
 - Select the most appropriate FS type
 - EXT3 is a good all-rounder
 - XFS for large files
 - Consider specialist file systems such as:
 - OCFS32, HSF2, BTRFS

- Newer tends to be better for low latency
 - Preemptible kernel first appeared in 2.5 but took a long while to appear in mainstream
 - Make sure CONFIG_PREEMPT enabled
 - `grep /boot/config-`uname -r` CONFIG_PREEMPT`
 - OFED RDMA – mainstreamed in kernel 2.6.14
 - CFS scheduler – mainstreamed in kernel 2.6.24
 - High resolution timer (nanoclock) in kernel 2.6.16 and high resolution events in 2.6.21
 - Hard Real time support is available as a patch for 2.6.24 and later, or pre-packaged as RedHat MRG or SuSE SLERT
 - <http://www.kernel.org/pub/linux/kernel/projects/rt/>
 - CONFIG_PREEMPT_RT
 - RH MRG and Novell SLERT both include RT patch and OFED (although not latest versions)
 - Personally saw big latency wins when I moved from RH 5.3 using 2.6.18 to a 2.6.27 kernel
 - OFED is pushed back into kernel but vendors prefer you to use their “value added” stack which replaces this.
 - Latest OFED build tends to have best latency and has more bug fixes, particularly for Ethernet RDMA and iWARP

- Be selective about NIC's
 - Big differences on latency and drops, caused by both hardware and quality of driver
- Consider your network stack



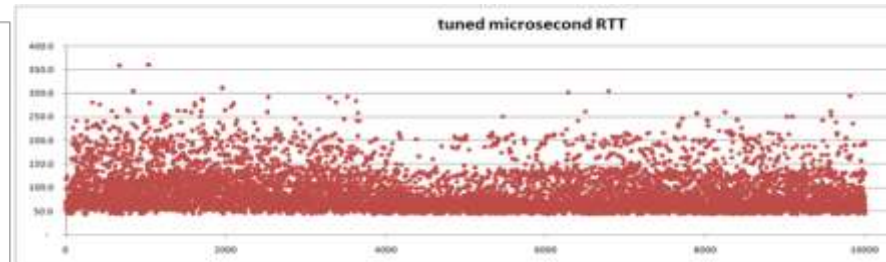
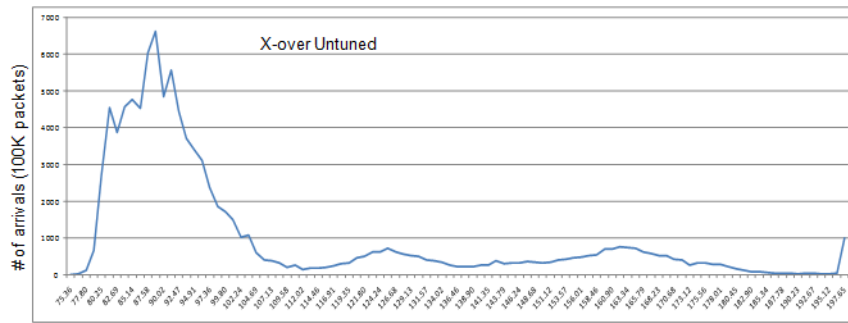
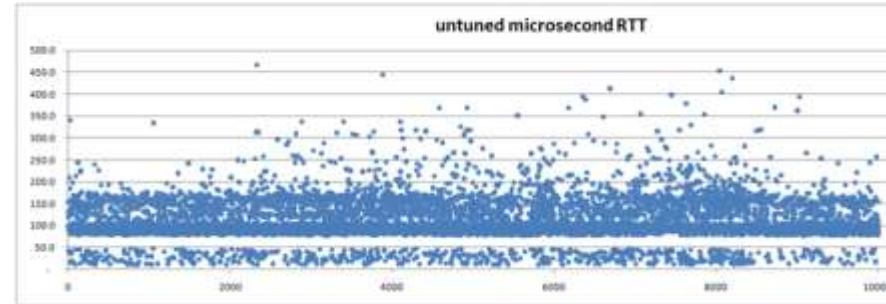
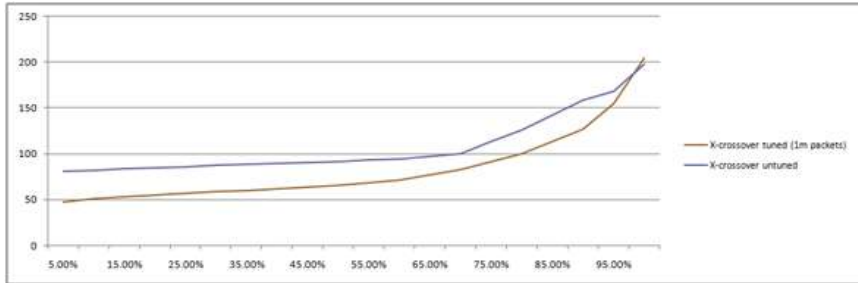
- Tune NIC driver
- Tune TCP/IP stack
- Note the tradeoffs – lower latency increases CPU overhead, power and complexity

- Zero copy of files using **sendfile(2)**
- Bypass TCP if possible by using OFED
- Pre-fault (i.e. touch) all memory and then pin all application virtual memory into physical pages with **mlockall(3c)**.
- Disable Nagle using **TCP_NODELAY** option on socket. Good practice to do this even if Nagle is disabled at the system level for code portability.
- Solaris specific
 - Early bind or link statically all libraries to prevent delays loading dynamic libraries during runtime – *setenv LD_BIND_NOW=1*
 - Run in a processor group which is masked from interrupts and pin memory using **psrset(1m)**
 - Set to a fixed priority 60 using FX scheduling class using **prcntl(1m)**. This will keep it above all TS, IA, and FSS processes.
 - Zones can be used for additional isolation since they add no detectable latency
 - Avoid application lock contention issues – identify with **mpstat(1m)** , **prstat(1m)** , **plockstat(1m)** and **dtrace**
- Linux specific
 - CPU groups and interrupt binding with **cpuset**
 - Stop CPU slow down , designed to save power - **cpuspeed**
 - Detect issues and tune with **mpstat**, **memprof**, **strace**, **ltrace**, **blktrace**, **valgrind**, **latencytop**, **tcptract**,
 - Kernel trapping with **systemtap**, communities third attempt, similar to **dtrace**

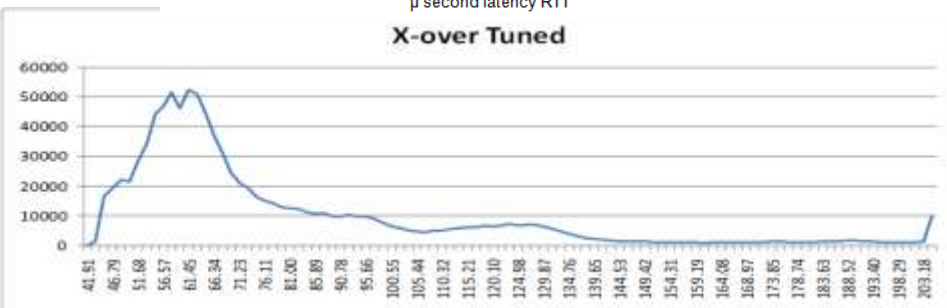
Example Linux Server Tuning

	min	average	median	max	Std Dev
X-crossover tuned (1m packets)	41.91	79.52	65.51	419	35.45
X-crossover untuned	75.359	104.8	91.5	561	30.16

29% improvement of median RTT



Results of initial Linux OS measurement and tuning



Continuous Refinement and Technology maturity

- Continuous research into new technologies
- Structured investigation and testing

	10ge	InfiniBand	40ge	100ge	Openonload	RoCEE	FPGA	GPU	Optical switching	Quantum	Photonics	Probability chips
References available	●	●			●	●	●	●	●			
Management Tested	●	●	●		●	●	●	●	●			
Performance Tested	●	●	●		●	●	●	●	●			
Functionality Tested	●	●	●		●	●	●	●	●			
Constraints clearly documented	●	●	●	●	●	●	●	●	●			
Can be shown to work	●	●	●	●	●	●	●	●	●			
Prototypes Available	●	●	●	●	●	●	●	●	●			●
Under development	●	●	●	●	●	●	●	●	●	●	●	●
Plausible	●	●	●	●	●	●	●	●	●	●	●	●

- Low latency design and Consultancy for FS clients
- Latency modelling and prediction
- Vendor independent and able to advise on technology and product selection
- Consultancy, design, performance reviews and education

www.informatix-sol.com

richard@informatix-sol.com

Workshop Modules available including:

[Low latency tuning for Solaris and Linux](#)
[Time Synchronization](#)
[Low latency monitoring and reporting](#)
[Programming with OFED](#)
[Ethernet v. InfiniBand](#)

[InfiniBand Overview](#)
[Long Distance InfiniBand](#)
[InfiniBand Management and Observability](#)
[InfiniBand diagnostics and troubleshooting](#)
[InfiniBand Product Selection](#)