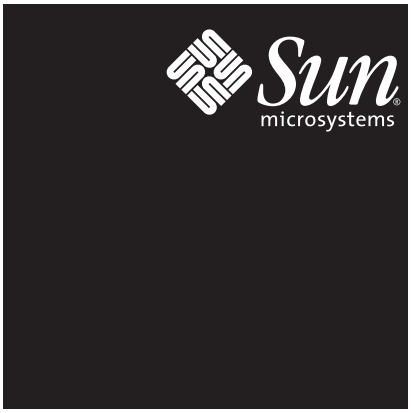
FireEngine—A New Networking Architecture for the Solaris™ Operating System

A Technical White Paper

Sunay Tripathi—sunay.tripathi@sun.com

November 2004



Please
Recycle

Table of Contents

Chapter 1
Overview

Introduction

The FireEngine networking stack for the Solaris™ Operating System (OS) is currently under development by Sun. Enhanced network performance and a flexible architecture to meet future customer networking needs are twin goals of FireEngine development. Addressing existing requirements, including increased performance and scalability, Disaster Recovery (DR), Secure Internet Protocol (IPSec), and IP Multiprocessing (IPMP), as well as future requirements—such as 10-gigabits per second (Gbps) networking, 100-Gbps networking, and TCP/IP Offload Engine (TOE)—are given equal priority.

Implemented in three phases, FireEngine's development stages are structured to provide increased flexibility and a significant performance boost to overall network throughput. Phase 1 has already been completed and these goals have been realized in services using TCP/IP. Web-based benchmarks show a 30- to 45-percent improvement on both SPARC® and Intel x86 architectures, while bulk data transfer benchmarks show improvements in the range of 20 to 40 percent. Phases 2 and 3 should deliver similar overall performance improvements. With increased flexibility and performance boosts of this magnitude, FireEngine is well on its way to reinforcing Sun's Solaris OS as the commercial standard for networking infrastructure.

Performance Barriers

The existing TCP/IP stack uses STREAMS perimeters and kernel adaptive mutexes for multithreading. As the current STREAMS perimeter provides per module, per protocol stack layer, or horizontal perimeters. This can, and often does, lead to a packet being processed on more than one CPU and by more than one thread, leading to excessive context switching and poor CPU data locality.

IP's PERMOD perimeter has long been known to cause connection and CPU scalability problems. The major problem is that certain operations are "exclusive" (preventing any other network activity) and this exclusivity is implemented using the non-scalable concept of PERMOD perimeter. With the PERMOD perimeter all access to any IP instance should go through a single `syncq` structure.

Network Performance

FireEngine introduces a new highly scalable, packet classification architecture called *Firehose*. Each incoming packet is classified early on, then proceeds through an optimized list of functions—the *Event List*—that makes it easy to add protocols without impacting the network stack's complexity, performance, or scalability.

FireEngine concentrates on improving the performance of key server workloads that have a significant networking component. The impact of network performance on these workloads, as well as benchmarks that describe overall workload performance, are described in Table 1.

Performance Metrics

Table 1. Workloads and Benchmarks

Workload	Network Performance Impact	Benchmark
Web Server	Highest	SPECweb99, SPECweb99_SSL
Application Server		ECPerf, SPECjAppServer2001
Collaborative Computing (HPC)		No networking benchmark available
Database	Lowest	TPC-SO

Applications often use networking in two distinct ways: To perform transactions over the network, or to stream data over the network. Transactions are short-lived connections transferring a small amount of application data, while streaming data is a transfer of large amounts of data during long-lived connections.

In the transaction case, performance is determined by a combination of the time it takes to get the first byte (first-byte latency), connection set up/tear down, plus network throughput (bits per second or bps). In the streaming case, performance is dominated by overall network throughput. These parameters impact performance in various ways, depending on the amount of data transferred. For instance, when transferring one byte of data, only first-byte latency and connection set up/tear down count. When transferring very large amounts of data, only network throughput is relevant.

Finally, there is the ability to sustain performance as the number of active simultaneous connections increases. This is often a requirement for Web servers.

A networking stack must take into account the host system's hardware characteristics. For low-end systems, it is important to make efficient use of the available hardware resources, such as memory and CPU. For higher-end systems, the stack must take into account the high variability in memory access times, as well as system resources that offload some functions to specialized hardware.

FireEngine focuses on these network performance metrics:

- Network throughput
- Connection set up/tear down
- First-byte latency
- Connection and CPU scalability
- Efficient resource usage

Table 2 shows the importance of each of these networking performance metrics as they relate to target workloads.

10-Gbps Performance

Table 2. Impact on Target Workloads of Network Performance (the number of •'s denotes relative performance improvement)

Function	Network Throughput	Connection Set Up/Tear Down	1st Byte Latency	Connection/CPU Scalability	Efficiency
Web Server	•••	•••	••	•••	•••
Application Server	•	•	•••	•	•
Database	••	•	•••	•	•
Collaborative or High-Performance Computing (HPC)	•	•	•••	•	•

10-Gbps networks are currently being deployed in initial commercial deployments. Today and for the foreseeable future, a 10-Gbps network can be saturated only when multiple CPUs are simultaneously pumping data into it. The Solaris OS is designed so that server network throughput grows linearly with the number of CPUs and network interface cards (NICs). This design feature enables FireEngine to take full advantage of the next generation of multiple CPUs on a die.

Solaris Network Cache and Accelerator

The Solaris Network Cache and Accelerator (Solaris NCA) caching server software upgrades the standard Solaris TCP/IP stack to deliver improved Web performance to the more recent versions of the Solaris OS. Solaris NCA technology is being merged into FireEngine development, with partial integration in phase one and full integration by phase two.

Chapter 2

FireEngine Design

The Solaris FireEngine networking performance improvement project adheres to these design principals:

- Data locality: Ensures that a connection is always processed by the same CPU whenever possible
- CPU modeling: Efficient use of available CPUs and interrupt/worker thread model. Allows use of multiple CPUs for protocol processing
- Code path locality: Improves performance and efficiency of TCP/IP interactions
- TCP/IP interaction: Switches from a message passing-based interface to a function call-based interface

FireEngine starts by improving TCP/IP interaction and IP `syncq` issues by merging TCP/IP into a single optimized, multithreaded module that is not dependent on STREAMS perimeter protection. This enables shorter code paths and shallower stack depths.

The previous networking stack, as well as the merged NCA networking stack, has multiple inefficiencies in areas such as thread management, locking, and per-connection synchronization. The FireEngine architecture uses a per-CPU synchronization mechanism, called *vertical perimeters*, inside the TCP/IP module. Vertical perimeters are implemented using a serialization queue abstraction, `squeue`.

As soon as an incoming packet reaches IP, connection lookup is done using the IP connection classifier. Based on that classification, the connection structure is identified. Since the lookup happens outside the internal perimeter, the connection is bound to an instance of the vertical perimeter (or `squeue`) when the connection is initialized. Subsequent packets for that connection are always processed on the same squeue that the connection is bound to, so that the system can achieve better cache locality and increased overall network performance.

The IP connection classifier also becomes a database for storing the sequence of protocol function calls necessary to process all inbound and outbound packets. This sequence—the Event List—processes packets for connections, and is the basis of a new framework that defines and processes network traffic.

By taking a database-like approach for these functions, reconfiguration of existing protocols and the implementation of new protocols can be done with greater ease. This approach allows the Solaris networking stack to be changed from the current STREAMS-based message passing interface to a Berkeley Software Design (BSD)-style function call interface.

FireEngine Implementation Phases

Because of the large number and dependent nature of changes required to achieve FireEngine goals, the development program is split into three phases:

Phase 1 Fundamental infrastructure implemented and a large performance boost realized. Application and STREAMS module developers see no changes other than better performance and scalability.

Phase 2 Feature scalability, offloading, and the new Event List framework implemented.

Phase 3 Other modules—such as IPSec, SCTP, and IPfilter—will be converted to the new Event List framework, further increasing performance and scalability.

Table 3 shows the feature set associated with each phase of FireEngine development and its corresponding Solaris OS release.

Table 3. FireEngine Release Phases

	Phase 1	**Phase 2**	**Phase 3 (Firehose)**
Release	**Solaris 10**	**Solaris 10 Updates / Solaris Express**	**Solaris 11**
Feature	IP MT	Full NCA merge UDP Performance	TLI/TPI
	IP Fan-out Classifier	Event List	Full Classifier
	Vertical Perimeters	Packet Event Framework	Other Offload
	Merged TCP/IP	Classifier Offload	
	Syscall Changes - `Accept` - `Connect` - `Socket` - `Sendfilev` - `Close`	TOE/RDMA Support Observability	

Figure 1 shows areas of the Solaris OS that are modified by FireEngine.

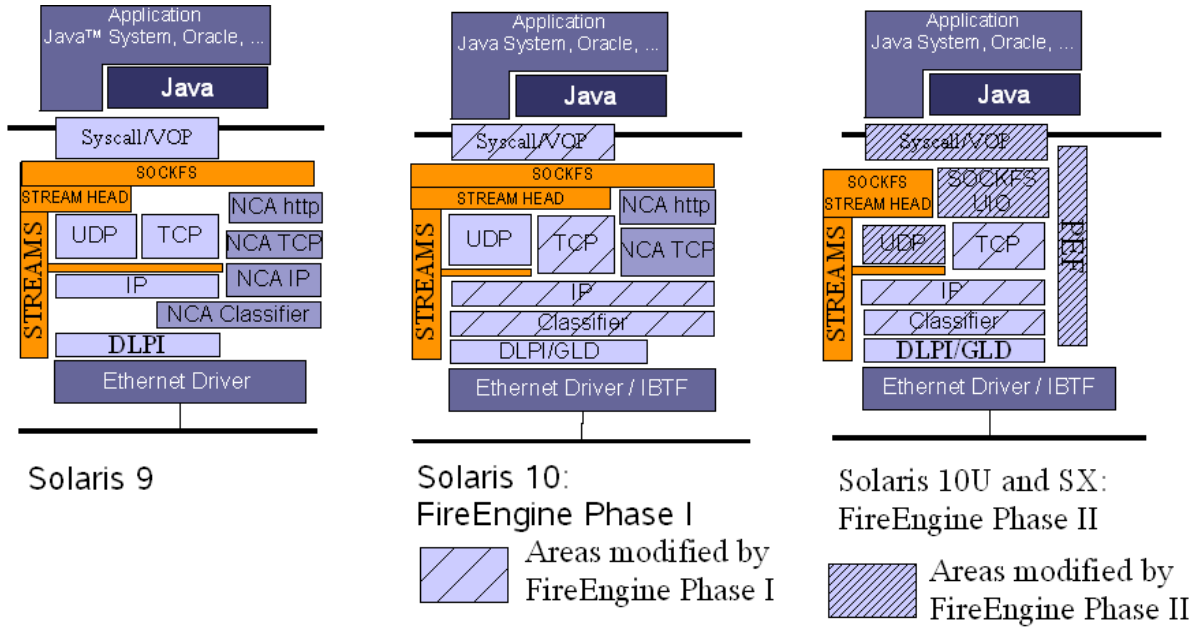


Figure 1. Network Stack Evolution

Table 4 shows expected improvement areas for all phases of FireEngine development.

Table 4. Improvement Areas for FireEngine Phases (the number of •'s measures the relative performance improvement)

Improvement Area	**Phase 1**	**Phase 2**	**Phase 3**
Performance Scalability	•••		
Interrupt Model Improvement	•••	•••	•
IP Fully Multithreaded	•••		
Call Stack Depth Reduction	••	••	
Connection Set Up/Tear Down Improvement	•••	•	
Better TCP/IP Interaction	•••		
Better UDP/IP Interaction		•••	
Feature Scalability		•••	
Stack Observability Improved		•••	
Hardware Offload Support		•••	
Feature Classifier			•••
NCA Stack Merge		•••	
NFS/RPC/TLI Performance		••	••

Tools and utilities are available for developers and administrators to monitor, measure, and tune FireEngine facilities.

Chapter 3

FireEngine Phase 1 Architecture

IP Classifier-Based Fan Out

When the Solaris IP receives a packet from a NIC, it classifies the packet and determines the connection structure and vertical perimeter instance that will process that packet[1]. New incoming connections are assigned to the vertical perimeter instance attached to the interrupted CPU. Or, to avoid saturating an individual CPU, a fan-out across all CPUs is performed. A NIC always sends a packet to IP in interrupt context, so IP can optimize between interrupt and noninterrupt processing, avoiding CPU saturation by a fast NIC.

There are multiple advantages with this approach:

- The NIC does minimal work, and complexity is hidden from independent NIC manufacturers.
- IP can decide whether the packet needs to be processed on the interrupted CPU or via a fan out across all CPUs. Processing a packet on the interrupted CPU in interrupt context saves the context switch, compared to queuing the packet and letting a worker thread process it.
- IP can also control the amount of work done by the interrupt without incurring extra cost. On low loads, processing is done in interrupt context. With higher loads, IP dynamically changes between interrupt and polling while employing interrupt and worker threads for the most efficient processing. In the case of a single high bandwidth NIC (such as 10Gbps), IP also fans out the connection to multiple CPUs.
- If multiple CPUs are applied, the connection is bound to one of the available CPUs[2] servicing the NIC. Worker threads, their management, and special fan-out schemes can be coupled to the vertical perimeter with little code complexity. Since these functions reside in IP, this architecture benefits all NICs.
- The DR issues arising from binding a worker thread to a CPU can be effectively handled in IP.

Vertical Perimeters

The Solaris 10 FireEngine project introduces the abstraction of a vertical perimeter, which is composed of a new kernel data structure, the `squeue_t` (serialization queue type), and a worker thread owned by the `squeue_t`, which is bound to a CPU.

Vertical perimeters or `squeues` by themselves provide packet serialization and mutual exclusion for the data structures. FireEngine uses a per-CPU perimeter, which is a single instance per connection. For each CPU instance the packet is queued for processing, and a pointer to the connection structure is stored inside the packet.

1.A connection is tied to a vertical perimeter instance, and all packets for that connection are processed on that vertical perimeter.

2.Based on round robin at present. More enhanced schemes are envisioned for the future.

The thread entering squeue may either process the packet immediately, or queue it for later processing. The choice depends on the squeue's entry point and its state. Immediate processing is possible only when no other thread has entered the same squeue.

A connection instance is assigned to a single squeue_t so it is processed only within the vertical perimeter. As a squeue_t is processed by a single thread at a time, all data structures used to process a given connection from within the perimeter can be accessed without additional locking. This improves both CPU and thread context data locality of access for the connection metadata, packet metadata, and packet payload data, improving overall network performance.

This approach also allows:

- The removal of per-device driver worker thread schemes, which are often problematic in solving system-wide resource issues.
- Additional strategic algorithms to be implemented to best handle a given network interface, based on network interface throughput and system throughput (such as fanning out per-connection packet processing to a group of CPUs).

IP Multithreading (MT)

To avoid the inefficiencies in the Solaris OS's previous IP operations during plumbing[3], most `xxx_set ioctl()` calls, some multicast operations, and some operations that require holding a global IP `syncq` lock, FireEngine makes the Solaris IP fully multithreaded, removing the requirement of thread exclusivity and switching from PERMOD to D_MP STREAM perimeter. FireEngine no longer uses STREAMS-provided PERMOD protection.

Merged TCP/IP

The TCP and IP stacks in version previous to Solaris 10 were separate STREAMS modules. They communicated with each other using STREAMS message-passing mechanisms. This presented several problems:

- Extra overhead from the `putnext()` call
- Potentially long call stacks
- Loss of context between modules
- Information gathered from incoming `mblocks` in one module had to be gathered again
- Poor code or data locality

Since the TCP and IP stacks embed a great deal of knowledge about each other, they become more efficient and less complex by making each of them call the other directly. This eliminates STREAMS-related processing overhead and saves significant CPU cycles.

Connection Set Up/Tear Down

Most socket-related system calls show performance improvements when using merged TCP/IP modules. The gains come from reducing the cost of plumbing the STREAM and introducing direct calls between TCP and IP.

The few system calls specifically targeted for improvement are mentioned below. The terms *context* and *perimeter* are used interchangeably to specify the vertical perimeter or squeue to which the connection is bound.

3.Opening IP and autopushing TCP on top.

Accept()

The FireEngine architecture establishes a connection in its own perimeter as soon as a SYN packet arrives, ensuring that packets always land on the correct connections.

The connection indication is still sent to the listener on the listener's STREAM, but the accept happens on the newly created acceptor STREAM[4], and the acknowledgment can be sent on the acceptor STREAM. This ensures that the listener doesn't become the bottleneck when a large number of new connection requests arrive simultaneously.

Connect()

Connect performance will improve significantly in FireEngine phase 2, when `sockfs` is integrated more tightly with TCP. However, `connect()` still benefits from phase 1 improvements because setup is more efficient and the overall FireEngine architecture improves efficiency.

In addition to the general improvements already mentioned, FireEngine improves IP bind logic by doing a bind with the classifier as a direct function call, eliminating the need to wait for an acknowledgment by the caller.

Socket()

With a merged TCP/IP module, FireEngine achieves almost a 25-percent improvement over the prior stack. The improvement in opening the acceptor STREAM—the big cost of setting an incoming connection—is approximately 80 percent.

Sendfilev()

Apart from the connection set up and tear down, both `sendfile()` and `sendfilev()` performance was increased with FireEngine. TCP flow control information was made available to `sockfs`, so in case the connection is flow controlled, `sendfilev()` can package the data to be better suited for larger reads and TCP consumption. A nonflow-controlled STREAM means that the data should be sent out immediately, avoiding queuing if possible. This change was important because the data should always be available when an acknowledgment (ACK) arrives and a send window opens up.

Close()

By taking the merged TCP/IP stack and a reference count-based architecture approach, the Solaris TCP's dependence on the queue is removed and a close can proceed immediately, while threads are still processing the TCP data structure. Table 5 shows a summary of how FireEngine improves `Libmicro` performance.

Table 5. Libmicro Results (Millions Per Second Per Call: msec/call)

Connection Type	Solaris 10 (Build 24)	FireEngine	Improvement
Connect()/Accept()	203 msec/call	146 msec/call	28%
Socket()	33 msec/call	25 msec/call	24%
Close()	88 msec/call	47 msec/call	47%

4.There is no need to allocate data structures for this STREAM.

Chapter 4
Summary

The Solaris FireEngine architecture is designed to make Solaris 10 OS-based networking significantly faster and more flexible. Phase 1 has been completed and integrated into Solaris 10. Significant flexibility and performance improvements have been realized in workloads that use TCP/IP. Web-based benchmarks on both SPARC and i386 architectures have been improved significantly, and bulk data transfer benchmarks have also improved. Test TCP (TTCP) and Netperf measurements show that FireEngine is more efficient than the previous Solaris networking infrastructure, and more efficient than Linux in CPU utilization (see benchmark web sites listed in Chapter 5)[5]. Similar improvements are expected for phases 2 and 3.

The following summary outlines the high-level issues driving overall FireEngine development.

- Move to a reference-based scheme for network processing
- Merge TCP/IP into a single module and create a function call-based interface
- Use a serialization mechanism (squeue) to protect the TCP data structure
- Make IP fully multithreaded, removing current STREAMS protection dependencies
- Use a connection classifier early in IP processing
- Bind each connection to a particular squeue
  - Packets for a particular connection are always processed on the same squeue
- Process the squeue only one thread at a time
  - Provides mutual exclusion for TCP data structures (vertical perimeters)
- Create a per-CPU squeue
- Bind each inbound connection to the squeue that is attached to the interrupted CPU executing the incoming connection
- Bind each outbound connection to the squeue that is attached to the CPU executing the application

With flexibility and network throughput a key component of the server value proposition, the Solaris FireEngine technology is well on its way toward re-establishing Sun as the commercial standard to which all other competitive network architectures are compared.

5.Solaris performance evaluation using Sun™ Enterprise 220R server with 2x450-MHz CPUs, Quad Fast Ethernet (QFE), and hme

Chapter 5

More Information

For more information, please visit these sites.

Solaris OS	sun.com/solaris
Solaris NCA	sun.com/software/whitepapers/solaris9/networkcache.pdf
Solaris Documentation	docs.sun.com
SPECweb99	specbench.org/osg/web99/
SPECweb99_SSL	specbench.org/osg/web99ssl/
ECPerf	java.sun.com/j2ee/ecperf/index.jsp
SPECjAppServer2001	specbench.org/osg/jAppServer2001/
TPC-SO	tpc.org
Netperf	netperf.org/netperf/NetperfPage.html
TTCP	pcausa.com/Utilities/pcattcp.htm

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN **Web** sun.com

The Network is the Computer™

Sun Worldwide Sales Offices: Argentina +5411-4317-5600, Australia +61-2-9844-5000, Austria +43-1-60563-0, Belgium +32-2-704-8000, Brazil +55-11-5187-2100, Canada +905-477-6745, Chile +56-2-3724500, Colombia +571-629-2323 Commonwealth of Independent States +7-502-935-8411, Czech Republic +420-2-3300-9311, Denmark +45 4556 5000, Egypt +202-570-9442, Estonia +372-6-308-900, Finland +358-9-525-561, France +33-134-03-00-00, Germany +49-89-46008-0 Greece +30-1-618-8111, Hungary +36-1-489-8900, Iceland +354-563-3010, India–Bangalore +91-80-2298989/2295454; New Delhi +91-11-6106000; Mumbai +91-22-697-8111, Ireland +353-1-8055-666, Israel +972-9-9710500 Italy +39-02-641511, Japan +81-3-5717-5000, Kazakhstan +7-3272-466774, Korea +822-2193-5114, Latvia +371-750-3700, Lithuania +370-729-8468, Luxembourg +352-49 11 33 1, Malaysia +603-21161888, Mexico +52-5-258-6100 The Netherlands +00-31-33-45-15-000, New Zealand–Auckland +64-9-976-6800; Wellington +64-4-462-0780, Norway +47 23 36 96 00, People's Republic of China–Beijing +86-10-6803-5588; Chengdu +86-28-619-9333 Guangzhou +86-20-8755-5900; Shanghai +86-21-6466-1228; Hong Kong +852-2202-6688, Poland +48-22-8747800, Portugal +351-21-4134000, Russia +7-502-935-8411, Saudi Arabia +9661 273 4567, Singapore +65-6438-1888 Slovak Republic +421-2-4342-94-85, South Africa +27 11 256-6300, Spain +34-91-596-9900, Sweden +46-8-631-10-00, Switzerland–German 41-1-908-90-00; French 41-22-999-0444, Taiwan +886-2-8732-9933, Thailand +662-344-6888 Turkey +90-212-335-22-00, United Arab Emirates +9714-3366333, United Kingdom +44-1-276-20444, United States +1-800-555-9SUN or +1-650-960-1300, Venezuela +58-2-905-3800, or online at sun.com/store

Printed in USA 06/04 XX0000-0/#K