

CSP: A Novel System Architecture for Scalable Internet and Communication Services

Hemal V. Shah, Dave B. Minturn, Annie Foong, Gary L. McAlpine, Rajesh S. Madukkarumukumana, and Greg J. Regnier

Enterprise Architecture Lab

Intel Corporation

5200 N.E. Elam Young Parkway
Hillsboro, OR 97124

{hemal.shah, dave.b.minturn, annie.foong, gary.l.mcalpine, rajesh.sankaran, greg.j.regnier}@intel.com

Abstract

The boundary between the network edge and the front-end servers of the data center is blurring. Appliance vendors are flooding the market with new capabilities, while switch/router vendors scramble to add these services to their traditional transport services. The result of this competition is a set of ad-hoc technologies and capabilities to provide services at the network edge. This paper describes the *Comm Services Platform* (CSP); a system architecture for this new ‘communication services tier’ of the data center. CSP enumerates a set of architectural components to provide scalable communication services built from standard building blocks that utilize emerging server, I/O and network technologies. The building blocks of CSP include a System Area Network, the Virtual Interface Architecture, programmable network processors, and standard high-density servers.

1. Introduction

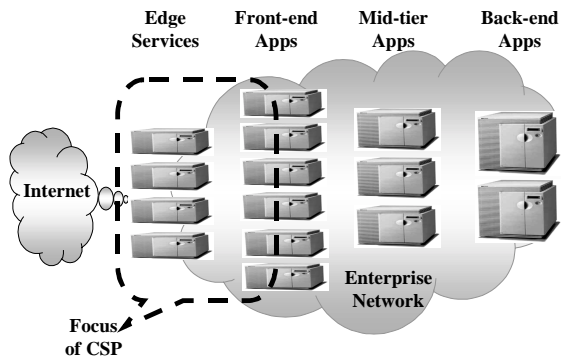


Figure 1: The N-Tier Datacenter

The three-tier data center model is well known. The first tier consists of front-end servers that provide web, messaging and various other services to clients on a network, the middle tier handles transaction processing and implements the data center business logic, while the back-end consists of databases that hold persistent state. We see a fourth tier emerging in this model between the network and the front-end servers. This ‘communication services tier’ provides an ever-rich set of functions. These services operate on network traffic at and above the network

layer, with well-known examples such as load balancing (at multiple levels), security (firewall and SSL), caching and others. These services intend to increase the responsiveness and throughput of the data center by distributing and offloading these functions from the server farm at the front end of the data center.

This paper describes a novel system architecture for a scalable, high-performance communication services tier based on emerging server and network technologies that are, or will become, standard building blocks. These technologies include System Area Network (SAN) technology, the Virtual Interface Architecture [20], programmable network processors, and standard high-density servers. We also describe a new core service, the SAN Proxy service that offloads and decouples the Transmission Control Protocol /Internet Protocol (TCP/IP) processing from the front-end servers, thus improving the performance of all of the services built on top of TCP/IP.

The organization of the rest of the paper is as follows. Section 2 discusses observations and motivations behind our research. Section 3 provides an overview of the CSP system architecture. Solutions and techniques developed in CSP are presented in Section 4. Initial evaluation of CSP using a system prototype and simulation results are provided in Section 5.

Finally, Section 6 summarizes our findings thus far and outlines possible future work.

2. Observations and Motivations

Evolution at the data center network edge has been driven primarily by demands to make web services more scalable, efficient and available. The most common solution applied to the scaling problem is to construct systems comprised of a large number of front-end servers (web server farms) behind load or content distributors. These systems typically use commodity servers for web services and specialized appliances, commonly termed “web-switches”, as distributors of web service requests. The connectivity between web-switches and front-end servers is predominately on a commodity LAN fabric, a.k.a. Ethernet, with Internet Protocol (IP) running over it.

Over the past few years, there has been an extensive amount of research on the functionality, design, and usage of web-switches. This research, along with a proliferation of commercial products [28, 29, 30, 34, 35], has mostly focused on the following areas:

- Web request distribution based on server load, content locality, cache affinity, or client session affinity [1, 14, 26, 28].
- Optimizations of server TCP connection management [4, 24, 34].
- Mechanisms for inter-connecting client and server TCP connections (TCP-splicing) [3, 12, 19].
- Offloading server CPU intensive operations, such as secure sockets layer (SSL) operations.

Each of these approaches has been based on optimizing and distributing web transactions by transparently intercepting and modifying the Hyper Text Transfer Protocol (HTTP)/TCP/IP packet dialog between the client and the server. All of these approaches operate under the premise that the web switch or appliance device must be interconnected to the front end servers, or other web appliances, using the standard TCP transport, even though the client end to end TCP connections are terminated at the web-switch. As a result, the web-switch is required to maintain two separate TCP sessions, one to the client and another one to the server. Typically, each client connection is paired with an associated server connection. Some web-switches have added an optimization that allows multiple client sessions to be multiplexed on a set of persistent web-switch to server TCP connections [34]. This reduces the server TCP connection setup and teardown overheads but

adds the complexity of session management and TCP data buffering to the web-switch.

We have observed an evolution of web-switches from stateless load distributors into intelligent devices that have control over both the client and server sessions. When co-located in a data center environment, along with front-end servers and other web appliances, they function more as peers in distributed Internet end-point applications environment. Given this observation, we propose that the data center web deployments are good candidates for the application of SAN technologies. SAN technologies, specifically the Virtual Interface (VI) Architecture and emerging Infiniband™ Architecture [9], were developed to eliminate inefficiencies incurred when using general-purpose network interfaces with transports such as TCP, for high-speed inter-process communications and I/O.

We propose that a systematic approach can be applied using SAN building blocks to solve the following issues with today’s data center systems:

1. The high software overheads of using general-purpose network interfaces significantly reduce the capacity of the front-end servers. Operating system related inefficiencies incurred in network protocol processing, such as user/kernel transitions, data copies, software multiplexing, and reliability semantics reduce both CPU efficiency and overall network throughput.
2. The knowledge gained by TCP/IP packet processing in a given component is not shared with other components in the system leading to excessive redundant processing and poor integration. For example, a TCP session establishment and subsequent session identification are performed on both the web-switch and the front-end server.
3. General-purpose network protocols are not suitable for low-latency, high throughput IPC between functional components. As a result, scalability and additional functionality is limited to costly and proprietary “in the box” solutions. For example, web-switches predominately use proprietary internal interconnects and inter-process communication (IPC) to add incremental functionality.
4. Web switch appliances tend to be proprietary closed systems that do not allow the development of new services by a large community of developers.

5. Rapidly increasing network speeds and usage demands create a greater need for an efficient and general-purpose method to independently scale packet switching capacity, packet processing, and network services.

Software overhead in server operating system network stacks, specifically sockets and TCP/IP, has been identified and studied extensively over the last ten years. Efficiencies have been added by tuning the server resident software and protocol stacks and by incremental offloading of computation intensive portions of the network stack into specialized hardware on the Network Interface Card (NIC) [25, 27, 33]. These optimizations, along with increases in server processing speeds, have improved network overheads but at too slow of a rate compared to increasing network throughput and Internet application demands [15].

To overcome these inefficiencies, we investigated the use of a Virtual Interface (VI) Architecture enabled SAN fabric as the network interconnect between the web-switch and the front-end servers. This investigation was prompted partially by the results of our previous VI Architecture related research showing that use of VI Architecture provided significant performance and efficiency benefits over standard server networking interfaces, even when used with higher level socket interfaces [16]. As an experiment, we ran a simulated web transaction performance test in order to re-evaluate TCP/IP and VI Architecture performance characteristics on modern software and hardware. The test comprised of iteratively sending a 256-byte message, simulating an HTTP *get* request from a client, followed by receiving a reply message of varying sizes from the server. The transaction latency and the server host CPU cycles spent per transaction were measured.

In the experiment, we used Gigaset cLAN product [31] as the native VI Architecture (nVIA) and M-VIA [11] over Gigabit Ethernet as the emulated VI Architecture (eVIA). TCP/IP related tests were run over Gigabit Ethernet. The results of this experiment are shown in Figure 2. As the graph illustrates, for the same reply size, nVIA and eVIA are able to achieve 2-2.25x and 1.3-1.6x better latencies than TCP/IP respectively. Furthermore, the CPU cycles spent per transaction for nVIA and eVIA were significantly lower than for TCP/IP (for large messages, an order of magnitude).

Another important characteristic of VI Architecture enabled SAN fabrics, is that they provide the

mechanisms needed to enable efficient low latency IPC between distributed components in a system. By using these mechanisms, which consist of the standard VI Architecture interface (VIPL) and underlying reliable SAN wire transport, a lightweight protocol can be constructed to efficiently exchange control information and data between components.

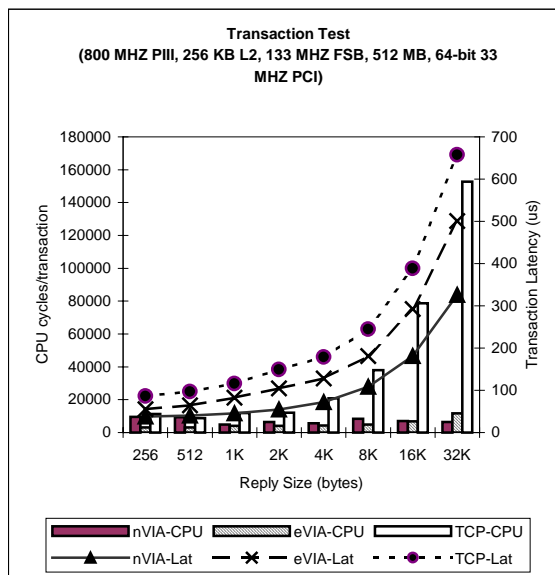


Figure 2: Transaction Test Results

Our idea was to research the use of VI Architecture enabled SAN fabric as the framework to create a distributed system architecture, termed CSP system architecture that would enable systems to be constructed entirely out of commodity building block components. The goals of this research being to address the issues raised earlier on the limited scalability, redundant packet processing, restricted functionality, and proprietary nature of existing web-switching systems deployed in today's data center.

3. System Architecture Overview

The CSP system architecture consists of multiple functional elements, or nodes, interconnected with a VI Architecture enabled SAN fabric. These elements can be enumerated to enable the construction of CSP systems with varying levels of functionality, scaling and performance. The decomposition of the CSP platform into a functional pipeline of building blocks allows scaling of each pipeline stage independently. Figure 3 illustrates the CSP system architecture. The functional elements of the CSP system architecture are described as follows:

System Area Network (SAN) and the Virtual Interface Architecture (VI Architecture): the SAN in the CSP system architecture provides the interconnecting fabric for all other elements in the system. CSP nodes attach to the SAN using the VI Architecture specification [20] compliant interface and the VIPL API [21]. A lightweight IPC mechanism, termed the “CSP Transport” is used to communicate control information and exchange data between the elements.

Network Node (NN): provides the LAN and/or WAN interface function in the CSP architecture. It performs the first level of processing in the functional pipeline of the CSP system by processing LAN/WAN packets and then forwarding them to other SAN nodes for further processing. Examples of NN packet processing include line-rate layer-3 packet forwarding, layer-4 load balancing, and TCP flow classification. The network node is optimized for fast packet processing and constructed using programmable network processors, such as the Intel® IXP1200 [32]

Proxy Node (PN): acts as the proxy between remote clients and high-level CSP services residing above the network transport layer. Proxy nodes perform the next level of processing in the CSP pipeline by terminating all client TCP sessions and subsequently communicating the associated data streams over the CSP transport between proxy and application nodes. Proxy nodes essentially de-couple network transport protocol processing cycles from application node compute cycles in order to enable independent scaling of packet and application processing. The architecture of proxy nodes in the CSP system enables them to perform various higher-level functions on the application data. Examples of these functions might include HTTP proxy services, web content transformation, such as SSL, and support for content-based distribution of web services.

Application Node (AN): hosts well-known applications, such as a web, mail or directory services. The application node is built using standard high-density server hardware and runs standard operating systems and applications. The application node utilizes the CSP transport to bypass the kernel-resident network stack during communication with proxy nodes (and/or possibly other SAN-enabled nodes).

Management: The CSP architecture defines a management function that provides dynamic resource discovery and configuration services, along with network policy management. As a central location

for resource and policy information, the management function is assumed to be configurable in a redundant manner. An example of the CSP management function would be a pair of nodes in a CSP system that acts as a repository and distribution center for system configuration and packet processing policy.

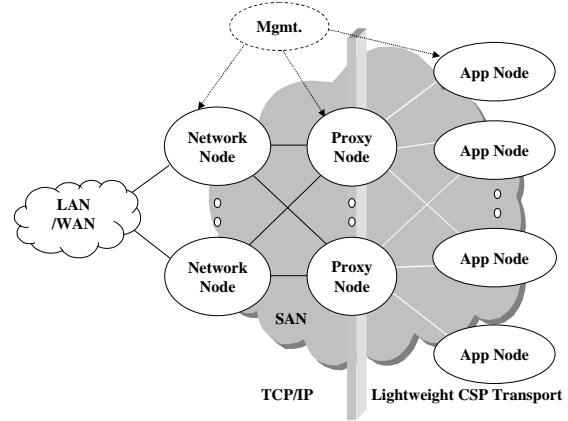


Figure 3: CSP System Architecture

4. CSP Solutions

The main focus of CSP is to provide solutions to the network edge related problems described in Section 2. This section describes main techniques and solutions developed for the CSP architecture.

4.1 SAN Tunneling

In a distributed functional pipeline, one function of the network node is to forward packets between the external network (LAN or WAN) and the SAN. In the CSP architecture, packets are communicated between a network node and a proxy node through a SAN tunnel (VI) based IPC mechanism. This is accomplished by encapsulating network packets with SAN packet header information. This feature is important in order to allow the proxy node to process TCP/IP packets at user level using VIs. Furthermore, SAN tunneling enables communication of additional information in a VI packet header that can be used in TCP/IP header compaction, TCP flow identification, and TCP checksum off-loading.

4.2 Flow Labeling

In order to eliminate redundant packet header processing and to enable efficient lookup for TCP packet flows, we developed a VI Architecture based TCP flow labeling technique. This technique is comprised of tunneling encapsulated TCP packets over VI connections, a simple TCP flow-identifier-

based lookup scheme, and TCP/IP header compaction. The use of TCP flow labels enables simple indexing based lookup (constant time lookup) schemes on the network, proxy, and application nodes. On the proxy node, flow labels eliminate the need to maintain a separate TCP transmission control block (TCB) cache for lookup, enabling it to outperform other TCP TCB lookup schemes used in practice. On the network node, TCP flow labels are used on outbound TCP packet flows to simplify packet forwarding information lookup.

In contrast to other flow labeling schemes in use by the web switches, this scheme carries the TCP flow identification information all the way to the end stations (proxy nodes and application nodes) and thus reduces redundant TCP/IP header processing. TCP/IP header compaction is used to allow space for additional VI packet header bytes in the encapsulated TCP packet without having to fragment the packet or reduce the typical TCP maximum segment size (MSS) for like media, i.e. Ethernet.

During the initial TCP connection establishment, a flow identifier is generated by the proxy node and communicated to the network node as a part of an outgoing TCP SYN/ACK or TCP SYN packet. During the connection tear down, the proxy node informs the network node about the termination of the flow in the outgoing TCP FIN/ACK or last TCP ACK packet. For the incoming traffic with known TCP flows, the network node performs the lookup and TCP/IP header compaction (optionally), sets the flow identifier field, and tunnels the rest of the packet to the appropriate proxy node. Upon receiving a compacted packet for a known TCP flow from the proxy node, the network node performs the flow identifier based lookup, constructs IP header, and transmits it on an appropriate LAN port. The proxy node maintains a table of pointers to TCBS. It uses the flow identifier as an index to the table and this allows constant time TCB lookup for an incoming TCP packet with the flow identifier.

4.3 Distributed Network Services

The CSP architecture enables two levels of network services. At the first level, network nodes can perform layer-2 to layer-4 (L2-L4) processing and at the next level proxy nodes can perform layer-5 to layer-7 (L5-L7) processing. Compared to existing 'in-the-box' solutions offered today [26, 28, 29, 30, 34, 35], the advantages of CSP distributed network services are:

- They provide scalable, open, and cost-effective solutions as they are built using VI Architecture interfaces, programmable network processors, and standard rack-mounted servers.
- Distribution of service functionality allows the components to be independently scaled and optimized.

An example of this type of services is distributed network load balancing, where the network nodes perform L4 load balancing across the proxy nodes either using Network Address Translation (NAT) or IP tunneling and the proxy nodes perform application level load balancing (L5-L7) across the application nodes. Thus, the network nodes along with the proxy nodes can be viewed as a distributed network switch performing L4-L7 load balancing.

4.4 SAN Proxy Service

We developed a new CSP core service on the proxy nodes called *SAN Proxy* to mitigate the OS based TCP/IP processing bottleneck on the application nodes. The SAN Proxy is a transport layer proxy service developed for decoupling TCP/IP processing from application processing. The SAN Proxy independently manages TCP/IP connections with the network clients and SAN channels (VIs) with the application nodes, relays TCP/IP byte streams arriving from the network clients to the application nodes using a lightweight protocol, and communicates the lightweight protocol data arriving from the application nodes to the appropriate network clients using TCP/IP. In this service, TCP flows are terminated on the proxy nodes and only the data above the TCP layer is communicated to/from application nodes over reliable VI connections. As the network nodes and proxy nodes communicate TCP/IP packets over SAN tunnels, the proxy nodes can process TCP/IP packets at the user-level without any OS intervention.

This isolation and decoupling of TCP/IP processing from application processing provides the following advantages:

- The control traffic generated due to TCP/IP processing no longer interferes with the applications because it does not propagate beyond the proxy nodes. The protocol processing overheads on the application nodes can be significantly reduced because of the use of lightweight protocol (CSP transport) to communicate data.

- Since the proxy nodes are not constrained by the legacy API (sockets), OS environment, and the hardware platform, they can be optimized to meet both TCP/IP and SAN protocol processing demands and can be scaled independently from application nodes.
- One or more application nodes can share the proxy nodes. Thus, proxy nodes can be used to perform L5-L7 policy execution across the application nodes in addition to off-loading protocol processing.
- Because the SAN Proxy handles all packets moving through the system, it is the logical place to add higher level services such as caching, firewall, and content transformation.

Figure 4 shows the flow of packets for a HTTP/1.0 transaction on the CSP with TCP/IP termination and protocol translation at the proxy node, illustrating the simplified control traffic at the application node.

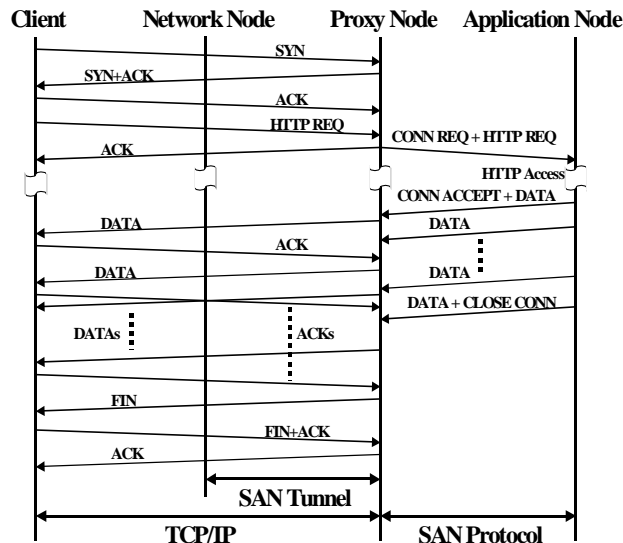


Figure 4: Example of a HTTP/1.0 Transaction with TCP/IP Termination on the Proxy Node

4.5 Application Support

Application support focuses on developing an efficient communication interface for front-end Internet applications. Internet applications commonly use a socket-based programming paradigm for communications. Our solution involves work in two major directions: i) support for existing legacy applications that use the BSD sockets API; ii) understand and overcome the communication

constraints imposed by legacy API. DASockets [5] and Windows Sockets Direct Path (WSDP) [23] are efforts with similar goals in mind.

5 CSP System Evaluation

A combination of system prototyping and simulation was being used for evaluation of the CSP performance, scalability, and for architectural validation. The combination provided the proof-of-concept as well as proof-of-architectural implementation for CSP. In this methodology, prototyping with off-the-shelf hardware and software components was focused on dealing with the real world problems of implementing a CSP system with existing technology and on providing details on SAN protocols, message formats, and timing information. The prototype results were used in conjunction with the simulation models to evaluate overall system performance, identify bottlenecks, evaluate messaging flows, test flow control mechanisms, and validate improvements.

5.1 Prototype Implementation

This subsection describes a prototype implementation of the CSP. The prototype implementation provides practical experience in building a CSP system and provides a functional demonstration vehicle. The following subsections describe in detail each component of our CSP prototype.

5.1.1 SAN and CSP Transport

In order to construct the prototype using existing technologies, we used Gigabit Ethernet as the switched interconnect between the elements of the system. To provide a VI Architecture compliant interface within each element of the system, we used the Modular-VI Architecture (M-VIA) [11] emulation software developed at Lawrence Berkeley National Labs. The M-VIA software emulates the VIPL 1.0 API semantics in kernel software in the absence of native VI Architecture hardware support. This allows us to develop the services and interfaces independent of the underlying SAN fabric, as long as it supports the VIPL interface.

It is noted here that for our prototyping and analysis purposes, we assume that switched Ethernet provides the reliable link semantics normally supplied by true SAN implementations. Also note that the emulated VI Architecture over Ethernet does not provide the same

level of performance as a true SAN with native VI Architecture hardware support.

Our prototype CSP transport is a lightweight protocol optimized for reliable SAN interconnects. The CSP transport was derived from our earlier stream sockets prototype over VI Architecture [16]. It was modified to allow proxy nodes to access internal VI descriptors and register/deregister buffers. Furthermore, the ability to wait or poll on particular values of the immediate data field of a VI descriptor was added for supporting multiplexing of sockets over a VI on the application nodes. The CSP transport manages VI resources and buffers used for communication, and performs credit-based flow control. The use of a single VI connection between an application node and a SAN proxy can have serialization and single-point of failure problems. On the other hand, use of a VI per socket limits scalability. In the CSP prototype, application level communication endpoints (sockets) were multiplexed over a pool of VI connections. A cache of registered buffers was maintained by the CSP transport in order to reduce the cost of memory registration and de-registration.

5.1.2 Network Node

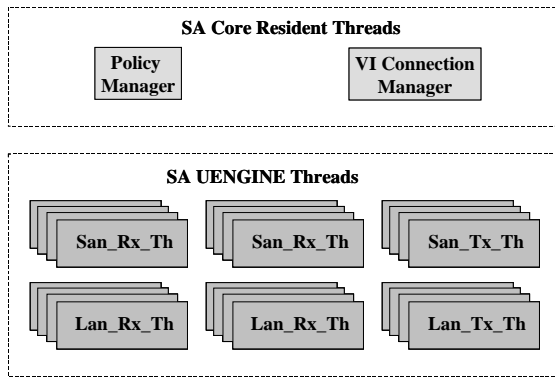


Figure 5: An IXP1200 based Network Node

The prototype CSP network node used an Intel® IXP1200 network processor [32]. The IXP1200 is a single chip network processor with interfaces to external memories and media access devices. The internal architecture of the IXP1200 consists of the following functional units; six multi-threaded micro-coded RISC engines (UENGINES), one StrongARM® core processor (SA_CORE), SRAM and SDRAM memory interface units and an external bus interface unit (IXBUS) which is used to connect

media access controllers. The prototype IXP1200 configuration used in the CSP prototype consisted of a single IXP1200 chip, 32 Megabytes of external SDRAM, 2 Megabytes of external SRAM, one eight port (quad) 10/100 Mbps Ethernet MAC for attaching to the LAN, and one two port Gigabit Ethernet MAC for attaching to the SAN.

The network node software architecture consists of twenty-four UENGINE threads for packet processing and two SA_CORE resident management threads. The SAN interfaces were made VI Architecture compliant by porting and running the M-VIA stack on the UENGINE threads. The UENGINE threads perform one of four packet processing operations; reception and processing of packets received on the LAN interface (Lan_Rx_Th), reception and processing of packets received on the SAN interface (San_Rx_Th), transmission of packets onto the LAN interface (Lan_Tx_Th), and transmission of packets onto the SAN interface (San_Tx_Th). Each of these operations is multi-threaded and the threads are distributed onto multiple UENGINES in order to provide wire speed packet forwarding between LAN and SAN ports. Figure 5 shows the IXP1200 based network node.

5.1.3 Proxy Node

In our prototype, we used standard rack mounted 800 MHz Pentium® III processor based servers running Linux OS (version 2.2.14-20) as the proxy nodes. The Proxy node uses user-level TCP/IP (derived from [10]) to communicate with LAN/WAN clients. On the behalf of clients, it interacts with the application nodes using the CSP transport. In our prototype, we only implemented the SAN Proxy service and a simple L4 load-balancing scheme on top of it.

In our implementation, the SAN Proxy service is a user-level multithreaded application that uses a pool of worker threads. Each worker thread is specialized for performing a subset of functions associated with TCP/IP processing, protocol translation/decoupling, SAN protocol processing, initial discovery mechanism, and L4-L7 policy execution. The SAN Proxy maintains two separate pools of registered buffers (one for incoming traffic and another for outgoing traffic). By maintaining a memory buffer structure that tracks the state of each registered buffer, the SAN Proxy achieves zero-copy translation. Figure 6 shows an architectural view of a proxy node along with a network node and an application node running a legacy application.

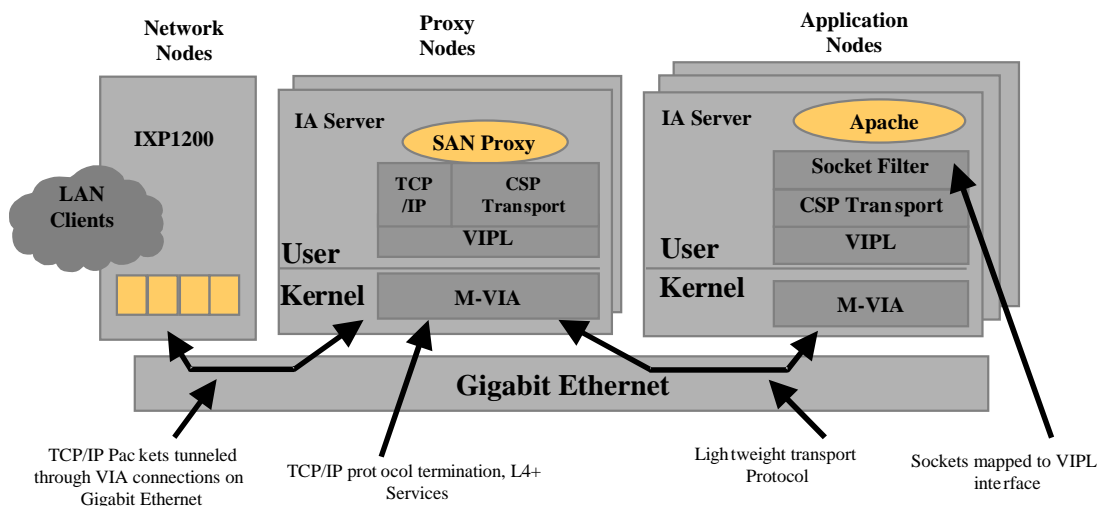


Figure 6: TCP/IP Termination Using Proxy Node

5.1.4 Application Node

Our initial approach was to enable existing applications without modifying the OS or applications. In our prototype, we introduced a *Socket Filter* module to intercept all socket-related function calls made by the legacy applications, and map them into appropriate CSP transport messages. In an environment such as Windows™ NT, the socket filter module could be loaded dynamically as a dynamically loadable library (DLL). However, due to the lack of DLL support in Linux, the application has to be recompiled and linked to the socket filter library.

We assumed that a majority of Internet server applications follow a predictable logic flow for socket-related calls, and we can consistently map them to the CSP transport messages. To simplify prototyping, we worked with those versions of applications that are multi-threaded, since VI resources cannot be shared across multiple processes. To date, we have successfully used the socket filter to enable an ftp server (betaftpd-0.0.7) and two web server applications (thttpd -2.16 and Apache-2.0a-dev3).

In order for a legacy application to preserve its host OS descriptors on the application nodes, the socket filter partitions 16-bit file descriptor (*fd*) space into system *fds* and transport *fds*. Furthermore, a *fd* for listening on a well-known port is obtained from host OS. The socket filter uses flow identifiers for both

actively and passively opened sockets. For passively opened sockets, flow identifiers supplied by the proxy nodes are used as new socket *fds*. For an actively opened socket, a mapping between a socket *fd* and the corresponding flow identifier is used as the socket *fd* needs to be generated prior to the connection establishment.

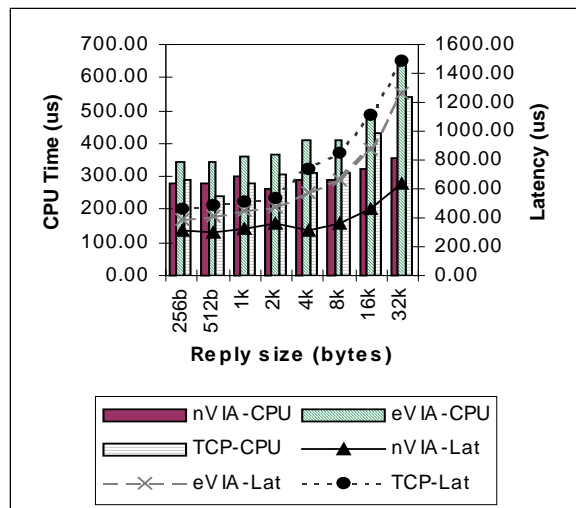


Figure 7: Performance of Apache over Various Transports

To ensure that real applications can truly benefit from the use of lightweight CSP transport, we determined the performance of the application node in isolation. Figure 7 compares the performance of an application node responding to HTTP/1.0 GET requests of

various file sizes using Apache over TCP/IP, eVIA, and nVIA. A simple client was used to drive sequential, single-stream HTTP GET requests to Apache over TCP. The same requests were issued to Apache over eVIA and Apache over nVIA hardware by using a simulated proxy (and client) over the CSP transport. The simulated proxy replicates the behavior of the proxy node and issues the necessary CSP transport messages. To ensure a fair comparison, the maximum packet size was set to 1514 bytes (including headers) and non-blocking writes were used in all cases. It should be noted that larger Maximum Transfer Sizes (MTS) up to 64K (as used in our experiments related to Figure 2) are more commonly used in SANs, thereby allowing for better performance. Furthermore, our prototype has to work under the restrictions imposed by legacy applications with substantial overheads added by the socket filter module. For the file sizes we considered, Apache over nVIA achieved 1.5-2.9x improvement in the latency. Furthermore, nVIA incurred less CPU overhead than TCP/IP.

5.1.5 Management Node

The main functions of the management node in our prototype is to maintain a centralized CSP information database, set policies on other nodes, and facilitate initial discovery operations. Other nodes register with the management node and inform it of their capabilities. As a part of initial discovery, the management node provides the proxy nodes and application nodes information about the network nodes and proxy nodes respectively. L2-L4 policies on the network nodes and L4-L7 policies on the proxy nodes are set after initial registration phase.

5.2 CSP Simulation Model

Development of the CSP simulation model and initial CSP system simulation studies were performed in parallel with the development of the actual CSP prototype. This allowed us to do early “pre-prototype hardware” studies of SAN fabric and CSP system configurations. The SAN fabric simulations were focused mainly on developing the SAN fabric infrastructure that would support the interconnect requirements of the CSP architecture, would scale to clusters of at least 64 nodes, and would allow configuring the SAN for either Ethernet or Infiniband™ Architecture operation. This enabled testing various CSP configurations with various SAN protocol and fabric topology options. The CSP system simulations were focused on evaluation of the

CSP web services performance on various system configurations and workload scenarios. Since we didn't have all the critical CSP prototype software tuned for performance and test results at the time this paper was written, we used a three-step process to characterize as much of a range of performance, scaling, and configuration information as we could.

The test (CSP) configuration used for our system simulations consisted of; two network nodes, each with a Gigabit Ethernet link to the external client network, two proxy nodes with two SAN ports each, four application nodes with one SAN port each, and a 10 port SAN switch interconnecting all the nodes. The SAN links were modeled as 100 meter, full-duplex links, at 1.25 Gbps for Ethernet and 2.5Gbps for Infiniband™ Architecture. SAN messages were segmented into 1514 byte frames for Ethernet and 282 byte frames for Infiniband™ Architecture.

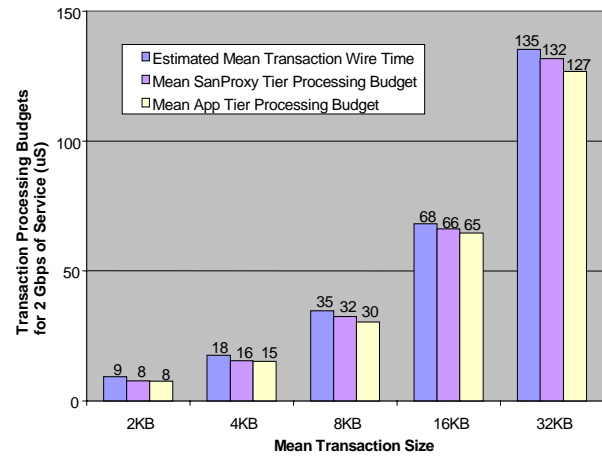


Figure 8 CSP Processing Time Budgets

For the first step in characterization, system simulation was used to empirically determine the processing budgets for the proxy and application tiers of the test configuration. These budgets were compared with the calculated mean wire time for each transaction size on the two output links from the network nodes to the clients (see Figure 8). The budgets were derived using constant streams of HTTP/1.0 GET requests for power-of-two transaction size from 2KB to 32KB. Each budget shows the mean transaction processing time the test configuration must achieve in the proxy or application tier to fully utilize the bandwidth of the two Gigabit Ethernet links to the client network, given a constant stream of the corresponding transaction size.

The second step was to establish the 800 MHz Pentium® III Processor based server we use in our

prototype system as a basic unit of measure of processing capacity (P) for characterizing the application tier (we have not yet acquired the data we need to do this for the proxy tier). In Figure 9, we graphed the application tier budgets from Figure 8 and the mean transaction processing times for the native VI Architecture case in Figure 7. Then we divided the native VI Architecture based processing times by the application tier processing budgets to create the curve showing the equivalent number of P required in the application tier to achieve the maximum throughput for each mean transaction size. It shows that it would take the equivalent of $33P$ to sustain the full 2 Gbps throughput with a stream of 2KB web transactions or $2.8P$ with a stream of 32KB transactions.

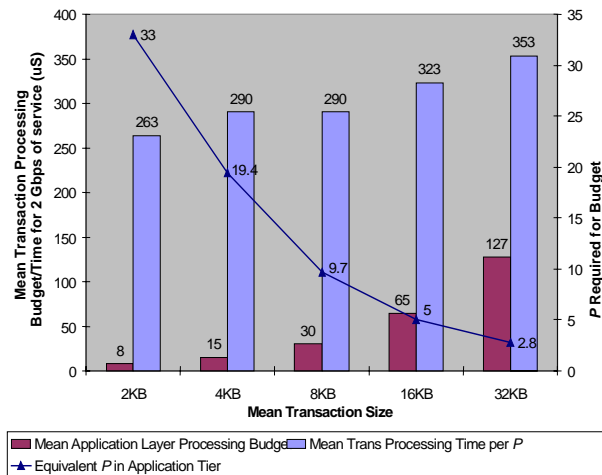


Figure 9: Equivalent P in Application Tier for 2 Gbps of Service

The third step was to input the appropriate processing times into our test model to simulate CSP configurations with the equivalent processing capacity of each of the points graphed on the curve in Figure 9. For each of these equivalent-processing capacities, we applied a web transaction workload consisting of transaction sizes randomly selected from the SPECweb99 static distribution (a mean transaction size of 14,384 bytes). Requests were generated by the clients in an exponential distribution with a mean rate of 20000 requests per second. Each test was run until 50,000 transactions were completed. The results of these simulations were used to produce the chart shown in Figure 10. The bars graph the total system throughput in transactions per second (TPS) and the curve graphs the TPS per P equivalent processing capacity.

The chart in Figure 10 shows that the performance of our test configuration peaks at the equivalent processing capacity of about $10P$ for 2 Gbps of service. It also shows that, although the system throughput drops significantly when the processing capacity is reduced to the equivalent of $\sim 3P$, the TPS per P is still increasing. If we consider only the cost/performance of the application tier, this suggests the best cost/performance CSP configurations for this workload will have the equivalent of $1P$ to $2P$ per gigabit of service. However, if such things as the cost of the links to the clients network are considered, one may need anywhere from $3P$ to $5P$ per gigabit of service to maximize the cost effectiveness of each network link. Additional processing loads such as dynamic objects, SSL processing, or content transformations will also drive up the processing capacity requirements.

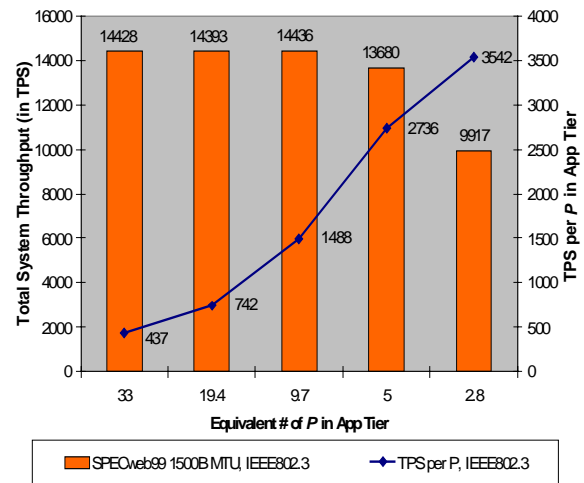


Figure 10: SpecWeb99 Static Distribution Performance

By including complete benchmark processing loads in the performance simulations and by collecting the critical performance data for each prototype node, we will be able to fully characterize the CSP in the future and enable easily optimizing configurations for various applications and cost/performance tradeoffs.

6 Conclusions and Future Work

In this paper, we described the CSP system architecture for scalable Internet and communication services. The distinguishing features of the CSP architecture being; SAN as the system interconnect and VI Architecture as a low-overhead interface to SAN, use of commodity building block components, decomposition of network services into distributed functional pipelines, and the de-coupling of network

protocol processing from end-point application processing. We described several CSP solutions, specifically the use of SAN tunneling and TCP flow labeling to reduce redundant packet processing, the CSP transport to provide an efficient “out of the box” IPC mechanism, the SAN proxy service that terminates client TCP/IP flows and maps them to the CSP transport, and the distribution of network services to allow scalability by service function.

Our prototype implementation and system simulation results have offered us significant insights into the viability, performance and scalability of web service systems based on the CSP architecture. The current prototype, which is functionally complete and operational, but not yet tuned for optimal performance, has allowed us to understand the difficulties and limitations imposed when constructing CSP systems out of available commodity building block components. The CSP simulation models have allowed us to extend past these limitations in order to gain the knowledge needed to architect future interconnect and component technologies better suited for CSP like systems. The following is a summary of the insights gained and issues raised during both the CSP prototype and modeling efforts:

- It is possible to construct a CSP type system using commodity-based components that can be scaled by function and processing capability to achieve a desired system throughput and cost point. To date, our CSP simulation studies have shown significant evidence that the CSP systems can be built from commodity building blocks and provide the necessary characteristics and scalability to be competitive with more specialized solutions. Unfortunately, at this time we have not acquired all of the critical prototype data needed to complete our analysis and fully characterize and contrast our solution against others. We plan to do so in future.
- Prototype benchmarking and simulation results showed that the use of a light weight CSP transport had performance benefits over traditional network transports, especially on components running traditional operating system stacks, such as application nodes. These benefits are largely due to the underlying VI Architecture and SAN interconnect technologies. The challenge lies in how to balance these benefits with the constraints imposed. For example, our early investigations showed that significant performance benefits could be achieved by using VI Architecture interfaces for web type transactions. Later prototyping showed that

utilizing the VI Architecture interface efficiencies while still maintaining application transparency and compatibility of existing Internet application interfaces, i.e. BSD sockets, was very challenging. In particular, the *select()* paradigm, used by socket-based applications, did not map efficiently to the queued real time signals available in message passing primitives provided by VI Architecture interfaces.

- The current CSP prototype and system model use a proxy node architecture that required all TCP control and data traffic be stored and forwarded through it. This presents challenges in how to balance memory capacity, I/O throughput, and processing capacity with price/performance and functionality on the proxy node.
- CSP System simulations have shown very aggressive processing budgets are needed for both the proxy and application node components to achieve maximum system capacity. Current generation components are not optimized for CSP traffic patterns or processing budgets and therefore require higher degrees of CSP node replication or component cost. By developing a set of server silicon components that are optimized for the CSP architecture and developing key hardware functionality, we believe a much better cost/performance solution can be achieved.

Going forward, we plan to address the constraints imposed by the legacy sockets API as well as explore legacy-free application interfaces that can take full advantage of the CSP transport optimizations (such as user level I/O, asynchronous I/O, and zero copy semantics). Ultimately, we hope to develop a set of OS independent high-performance APIs to facilitate this transaction. To address the proxy node store and forward issue, we plan to investigate alternative control and data flow methods, such as “third party data transfers”.

Other areas for future work are extending the CSP architecture to support SAN-aware file systems as defined in the direct access files system (DAFS) specification [6] and also extending the CSP architecture to interface into more traditional SAN-aware back-end applications, such as distributed databases.

To date we have focused heavily on basic web traffic. In the future, we would like to explore more communication intensive applications focused on a

mixture of voice, rich media and data with additional infrastructure services such as SSL and Firewalls. We believe the CSP architecture is a relevant step in the evolution of the converged data center networks of the future.

References

1. Apostolopoulos et al. "Design, Implementation and Performance of a Content-Based Switch", *In Proc. of IEEE INFOCOM 2000*, 2000.
2. Camarda et al. "Performance Evaluation of TCP/IP Protocol Implementations in End Systems", *IEE Proc. Comput. Digit. Tech.* Vol. 146, Jan 1999.
3. A. Cohen et al. "On the Performance of TCP Splicing for URL-Aware Redirection", *Proc. of the 2nd USENIX Symp. on Internet Technologies & Systems*, 1999.
4. Edith Cohen et al. "Managing TCP Connections under Persistent HTTP", *Proc. of the Eighth International World Wide Web Conf.*, 1999.
5. DASockets Protocol Specification, Version 0.6, Network Appliance, Inc., 2000.
6. Direct Access File System (DAFS). <http://www.dafscollaborative.org/>.
7. D. Dunning, G. Regnier, G. McAlpine et al. "The Virtual Interface Architecture", *IEEE Micro*, Vol. 3, No. 2, pp. 66-76, 1998.
8. A. Fox et al. "Cluster-Based Scalable Network Services", *Proc. of the sixteenth ACM symp. on Operating systems principles*, pp. 78-91, 1997.
9. Infiniband™ Arch. Spec. Vol 1 & 2. Rel. 1.0. www.infinibandta.org/download_spec10.html.
10. KA9Q NOS TCP/IP. <http://people.qualcomm.com/karn/code/ka9qnos>.
11. M-VIA : A High Performance Modular VIA for Linux. <http://www.nersc.gov/research/FTG/via>.
12. D. Maltz and P. Bhagwat. TCP Splicing for application layer proxy performance. Technical Report RC 21139, IBM, March 1998.
13. D. Maltz and P. Bhagwat. "Improving HTTP Caching Proxy Performance with TCP Tap", *In Proc. of HIPPARCH'98*, pp. 98-103, 1998.
14. V. Pai et al. "Locality Aware Request Distribution in Cluster-based Network Servers", *In Proc. Architectural Support for Programming Languages and Operating Systems*, 1998.
15. Greg Regnier, Dave Minturn, et al., Internet Protocol Acceleration Techniques, Intel Developer Forum, February 1999.
16. H. Shah, C. Pu, and R. Madukkarumukumana. "High Performance Sockets and RPC over Virtual Interface (VI) Architecture", *In Proc. Third Intl. Workshop on Communication, Architecture, and Applications for Network Based Parallel Computing*, pp. 91-107, 1999.
17. Evan Speight, Hazim Abdel-Shafi, and John K. Bennett. "Realizing the Performance Potential of the Virtual Interface Architecture", *In Proc. of the 13th ACM-SIGARCH International Conference on Supercomputing*, June 1999.
18. Junehwa Song et al. "Design Alternatives For Scalable Web Server Accelerators", *In Proc. of the IEEE International Symp. on Performance Analysis of Systems and Software*, April 2000.
19. Oliver Spatscheck et al. "Optimizing TCP Forwarder Performance", *In IEEE/ACM Tran. of Networking*, Vol. 8, No. 2, April 2000.
20. Virtual Interface Architecture Specification, Version 1.0, <http://www.viarch.org/>.
21. Virtual Interface Architecture Developer Guide, Intel Corporation, <http://developer.intel.com/design/servers/vi/>.
22. G. Welling, M. Ott, and S. Mathur. "CLARA: A Cluster-Based Active Router Architecture", *Hot Interconnects 8*, pp. 53-60, 2000.
23. Windows Sockets Direct Path for System Area Networks. Microsoft Corporation, 2000.
24. C. Yang and M. Luo, "Efficient Support for Content-Based Routing in Web Server Clusters", *Proc. of the 2nd USENIX Symposium on Internet Technologies & Systems*, 1999.
25. Alacritech, Alacritech Server Network Adapters. <http://www.alacritech.com/html/products.html>.
26. Alteon WebSystems, Alteon Web Switching Products. <http://www.alteonwebsites.com/products/>.
27. Alteon WebSystems, Next Generation Adapter Design and Optimization for Gigabit Ethernet. <http://www.alteonwebsites.com/products/whitpapers/adapter>
28. ArrowPoint Communications, ArrowPoint Content Smart Web Switches. <http://www.arrowpoint.com/products/index.html>.
29. Cisco Systems, Cisco Local Director. http://www.cisco.com/public/product_root.shtml.
30. F5 Networks, BIG-IP Products. <http://www.f5labs.com/f5products/bigip>.
31. Giganet, Inc., Giganet cLAN Product Family. <http://www.giganet.com/products/>.
32. Intel® IXP1200 Network Processor. <http://developer.intel.com/design/network/products/npfamily/ixp1200.htm>.
33. Interprophet Corporation. <http://www.interprophet.com/>.
34. NetScaler, WebScaler Internet Accelerator. <http://www.netScaler.com/products.html>.
35. Resonate, "Central Dispatch 3.0 – White Paper", <http://www.resonate.com/>.