Applications of the Split Protocol Paradigm

Bharat S. Rawal*, Ramesh K. Karne*, Alexander L. Wijesinha*,
Patrick Appiah-Kubi*, and Sonjie Liang*
Towson University, Towson, Maryland, USA

Abstract

A protocol is considered to be an *in-separable* entity between two or more communicating systems. In protocol splitting, a protocol is split into two sub-entities. The split protocol performs identical functions as the non-split protocol. Protocol splitting enables many innovations that are not possible with traditional non-split protocols. In this paper, we describe the split protocol concept and explore its use in implementing client server architectures, mini-cluster configurations, and migratory servers. Protocol splitting can be used to improve server performance, construct a variety of mini-cluster configurations to serve large workloads, build clusters without using expensive partitioning strategies, build simple migratory server systems, enrich client server architectures to accommodate client splitting, and achieve inherent server reliability. In particular, we consider the architecture, design and implementation of relevant split Web server and client applications. The primary purpose of this paper is to consolidate previous work on split protocols and discuss their broader impacts on the design and implementation of future communication system architectures.

Key Words: Bare machine computing, load balancing, server clusters, server migration, split protocols, web server performance.

1 Motivation

Network protocols enable interactions between two or more communicating entities via an exchange of messages. In many cases, two-way interactions between these entities are tightly coupled to process the messages following an event/action client-server model based on requests and replies. Examples of protocols that operate in this manner include HTTP, FTP, TLS, SMTP, and TCP. In general, such protocols are implemented on the client and server as ***in-separable*** entities.

This in-separable characteristic of protocols provides the motivation for a different approach to client-server interaction; that is, splitting a protocol so that conventional inseparability is broken. A protocol can be split in many ways as described in this paper. Splitting yields surprising results that can be useful in the design of future client-server systems, server clusters and migratory servers.

2 Introduction

We illustrate the split protocol concept by means of Web server and client applications that are designed and implemented using the bare machine computing (BMC) paradigm, which extends the dispersed operating system computing (DOSC) concept [16]. Bare PC applications are easier and simpler to use in the design and implementation of split systems. In principle, splitting could be done using conventional non-bare systems, but would be difficult to implement in the presence of an operating system (OS). In the BMC paradigm, the OS or kernel is completely eliminated and application objects (AO) [14] carry their own network stack and drivers to run independently on the bare machine. All BMC applications are self-supporting, self-managed and self-executable, thus allowing the application programmer to solely control the applications and their execution environments. Many client-server applications (for example [1, 9, 11]) have been built using bare PCs.

Web servers play a critical and central role in the Internet. Since Web servers only perform a certain (limited) set of functions and do not require elaborate user interfaces, they (and servers in general) are especially suited to be designed as bare PC applications. Furthermore, as opposed to bare PC Web servers, conventional Web servers require periodic maintenance, updates, and are prone to OS based attacks. Also, a bare PC Web server will have better performance, greater reliability, easier maintainability and more longevity than an OS-based server.

Many approaches to designing servers have been proposed in the literature. Google proposed its own lean Linux kernel [2]. Factored OS [30] targets scalability, bare-metal Linux [29] is a trimmed version of Linux, and SWILL [17] allows the addition of a simple Web server to applications. Exokernel [8] recommends moving network intensive processes from kernel to applications in order to speed-up processes, and the Fluke kernel [9] provides an API characterized by full interruptible and restartable ("atomic") kernel operations that has many advantages for applications. Tiny OS [27] suggests a small footprint for OS, OS Kit [20] provides system libraries for open systems, and IO-Lite [21] provided fast I/O calls bypassing system calls. Palacios and Kitten [18] are tools

* Department of Computer and Information Sciences. E-mail: (bsrawalkshatriya, rkarne, awijesinha, appiahkubi, sliang)@towson.edu.

designed to provide a thin layer over the hardware to support both full-based virtualization and native code bases. ChromiumOS [5] is an operating system designed for Web users. Many server systems disable the usual OS functions such as shell scripts, logs, and remote logins so they can gain performance. Commercial servers such as Apache and IIS focus their designs on optimization for increased performance and also provide an API to hardware resources instead of using system calls.

The above approaches require some OS layer, but the BMC paradigm proposes the avoidance of any OS, kernel, or centralized software to manage resources. In this paradigm, control is given back to the application programmer [16]. When a machine is made bare, it enables a programmer to develop applications that are completely independent of computing environments and only dependent on the underlying CPU architecture. However, the application programmer's job becomes more complex now, since it is necessary to deal with both application and system knowledge. On the other hand, the applications themselves are simple and easy to code once the programmer understands the underlying principles of the BMC paradigm. The design and implementation of a bare PC Web server are presented in [11], and the direct hardware interfaces for an application object (AO) are described in [15].

Web server reliability and load distribution are important problems that are still being addressed with a variety of techniques. In particular, load balancing techniques are used at various layers of the protocol stack to share the load among a group of Web servers [6]. In approaches such as Migratory TCP (M-TCP) [26], a client connection is migrated to a new server and an adaptation of TCP enables migration of the connection state. In contrast, the split technique when applied to TCP enables the splitting of a TCP connection between servers so that one server handles connection establishment and another handles data transfer. Splitting also allows a server to self-delegate a percentage of its connection requests to another server for data transfer and enables servers to share the load without a central control and without any client involvement.

To explain how a TCP connection is split by a BMC Web server, consider the message exchange in Figure 1. When an HTTP GET request arrives after the TCP connection is established, the BMC Web server sends a response (GET-ACK) to the client, and updates the client's state. The GET request is then processed by the server and the requested file is sent to the client during the data transfer. This process differs somewhat based on whether the HTTP request is static or dynamic. Although we address splitting only for static requests, the techniques apply to dynamic requests as well. Once the data transfer is complete, the connection is closed by exchanging the usual FIN and FIN-ACK messages. Although multiple GET requests can be sent using a single TCP connection, for ease of explanation, we only consider the case of a single GET per connection. A single HTTP request and its underlying TCP messages can thus be divided into connection establishment (CE), data transfer (DT), and connection termination (CT) phases.

The novel splitting concept presented here is based on splitting the HTTP request and underlying TCP connection into two sets {CE, CT} and {DT}. It allows one server to handle connections and the other to handle data without a need for too many interactions between servers. The data could reside on only one server or on both servers if reliability is desired. Splitting a client's HTTP request and the underlying TCP connection in this manner also provides the additional benefit of data location anonymity while enabling load sharing among servers. In the future, server machines optimized to handle only connection requests and others optimized to handle only data transfer could be built to take advantage of splitting.

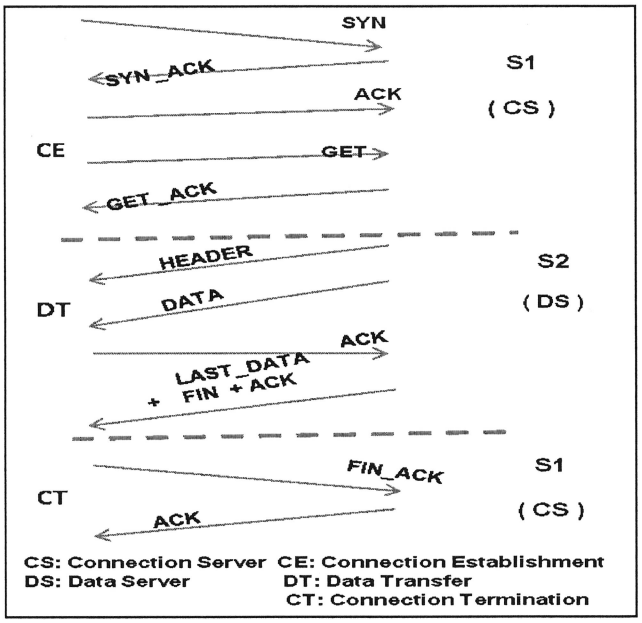


Figure 1: HTTP/TCP protocol interactions

Load balancing is frequently used to enable Web servers to dynamically share the workload. For load balancing, a wide variety of clustering server techniques [13] is employed. Most load balancing systems used in practice require a central control system such as a load balancing switch or dispatcher [28]. Splitting enables a new approach to load balancing whereby HTTP requests are split among a set of two to four servers, where one or more connection servers (CSs) handle TCP connections and may delegate a fraction (or all) requests to one or more data servers (DSs) that serve the data [22]. For example, the data transfer of a large file could be assigned to a DS and the data transfer of a small file could be handled by the CS itself. This approach is different from the approach introduced in [7] where TCP is spliced for URL-aware redirections.

As noted earlier, no client involvement is necessary for splitting unlike migratory or M-TCP [14]. In [22], splitting using a single CS and a DS was shown to improve

performance compared to non-split systems. Since the DSs are completely anonymous and invisible (they use the IP address of the delegating CS), it would be harder for attackers to access them. In particular, communication between DSs and clients is only one-way, and DSs can be configured to only respond to inter-server packets from an authenticated CS. We studied the performance of three different configurations of Web server clusters based on HTTP splitting by measuring the throughput (in requests/s) and also connection and response times at the client.

In real world applications, some servers may be close to data sources, and some servers may be close to clients. Splitting a client's HTTP request and the underlying TCP connection in this manner allows servers to dynamically balance the workload. The split concept has being tested in a LAN with multiple subnets connected by routers. In HTTP splitting, clients can be located anywhere on the Internet and interact with servers at a different location. However, there are security and other issues that require the CS and DS to be on the same LAN. These issues are discussed in more detail in [22].

The rest of the paper will describe the split protocol architecture, and the design, implementation and performance of several network applications. Since these applications were discussed in [22-25], we only consolidate and summarize the main concepts and results here.

3 Split-Protocol Server Architecture

The basic split protocol architecture used for the experiments described in [22] is reproduced here for illustration in Figure 2. A given request is split at the GET command between a CS and a DS. The CS handles the connections, and the DS handles the data transfer. In addition to connections, the CS also handles the data ACKs and the connection closing. The CS has complete knowledge of the requested file, its name, size, and other attributes, but it may or may not have the file itself. However, the DS has the file and serves the data to the client.

After the GET command is received by CS, it sends an ACK

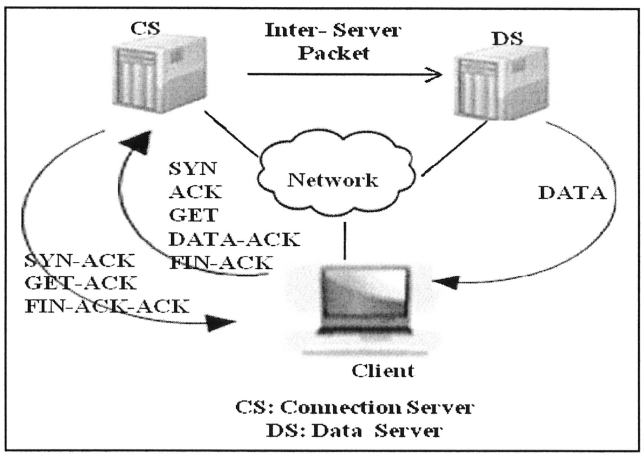


Figure 2: Split-protocol architecture

to the client and also sends a delegate message DM1 to DS. The message DM1 contains the state of the request that is stored in CS in the form of an entry in the TCP table (referred to as a TCB entry). When DM1 reaches the DS, it creates its own TCB entry and starts processing this request as if it was initiated in the DS itself. When a DS sends data to the client it uses the CS's IP. After the CS receives a FIN-ACK from the client to signify connection closing, it sends another inter-server packet referred to as DM2 to DS. The DM2 received by DS will close the state of the request in DS. These DM1 and DM2 inter server packets serve as the start and end of a request at DS. More details of the design and implementation can be found in [22].

The CS and DS architecture in Figure 2 allows for a variety of delegation configurations. A request received by CS can be either processed wholly at CS or delegated to DS. That is, some requests can be processed at CS and some can be delegated to DS resulting in a variation of the delegation ratio. As CS and DS are identical functional units, they can also perform the role of a CS or a DS. These novel splitting techniques and the associated Web server architecture could be used in distributed computing and for improving server reliability.

4 Mini-Cluster Configurations

The initial studies conducted on split servers and their performance led to the development of mini-cluster configurations. A variety of cluster configurations together with the architecture, design, implementation and performance of mini-cluster configurations were presented in [23]. Mini-cluster configurations also offer better reliability because of the interchangeable nature of CSs and DSs.

4.1 Cluster Configurations

In a split request, the client first sends the request to a CS, the CS sends an inter-server packet to a DS, and the DS sends the data packets to the client. Inter-server packets may also be sent during the data transfer phase to update the DS if retransmissions are needed. With partial delegation configuration, the CS delegates a fraction of its requests to DSs whereas in full delegation configuration, the CS delegates all its requests to DSs.

This mini Web server clustering technique uses two or more servers with protocol splitting. Configuration 1 in Figure 3 shows full delegation with one CS, one DS, and a set of clients sending requests to the CS. The DS and CS have different IP addresses, but the DS sends data to a client using the IP address of the CS. Configuration 2 in Figure 4 shows a single CS with two or more DSs in the system with partial or full delegation. In partial delegation mode, clients designated as non-split request clients (NSRCs) send requests to the CS, and these requests are processed completely by the CS as usual. The connections between the NSRCs and the CSs are shown as dotted lines. With full delegation, clients designated as split-request clients (SRCs) make requests to the CS, and these requests are delegated to DSs. For full delegation, there are no

NSRCs in the system. When requests are delegated to DSs, we assume they are equally distributed among the DSs in a round-robin fashion. It is also possible to employ other distribution strategies. Configuration 3 in Figure 5 shows two

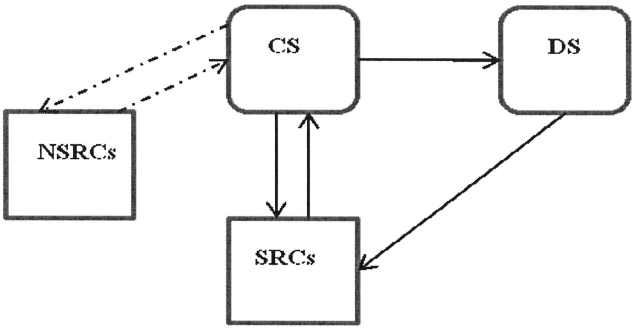


Figure 3: Split architecture configuration 1

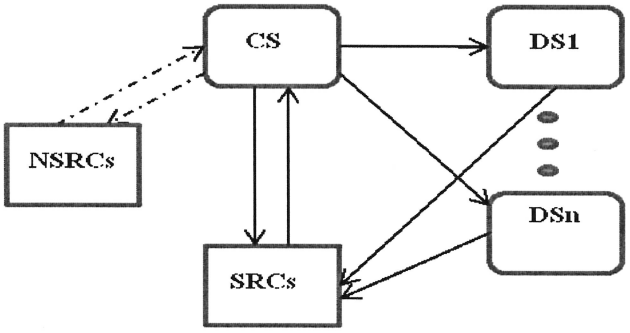


Figure 4: Split architecture configuration 2

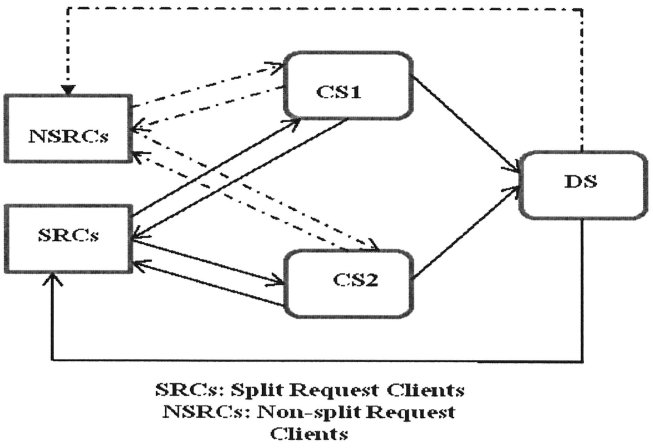


Figure 5: Split architecture configuration 3

CSs and one DS with both SRCs and NSRCs. For this configuration, we used small file sizes to avoid overloading the single DS. Although we have not done so, multiple DSs could be added as in Configuration 2.

4.2 Experimental Results

Some results of performance studies conducted on the mini-cluster configurations are given here. Figure 6 compares the throughput for split servers with full and partial delegation by varying the file size. The maximum throughput and a performance improvement of 25 percent are attained for 64 KB files with partial delegation. For 100 KB and 128 KB files, the performance improvements due to splitting are 17.7 and 12.5 percent, respectively with partial delegation. Notice that when two identical servers are used (no split), the maximum request/sec that can be achieved is about 2000 requests/sec for 64K resource file size, assuming there is no overhead for load balancing. In the case of split protocol servers however, the maximum request/sec that can be achieved is 2500 requests/sec, which is 25 percent more than theoretical maximum. This performance improvement shows the benefit of using the split protocol technique.

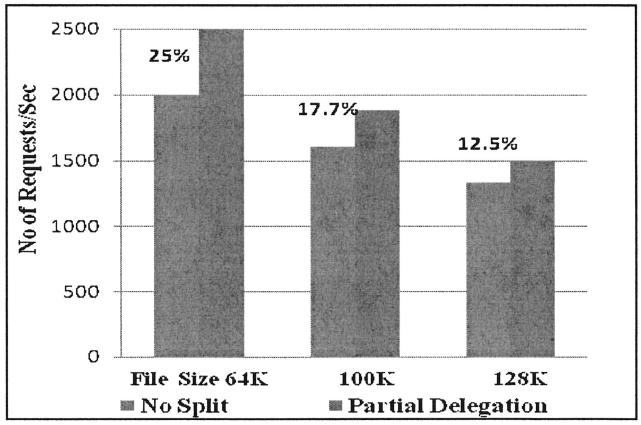


Figure 6: Throughput with full/partial delegation for varying file sizes (configuration 2)

Figure 7 shows the throughput for three servers with full and partial delegation using configuration 3. In this configuration, the throughput improvement over three non-split servers with full delegation is about 6.5 percent. However in the case of partial delegation, the throughput improvement is about 22 percent over three non-split servers.

The configuration studied here does not require a central load balancer or dispatcher, and is completely transparent to the client. The experimental results for this configuration suggests that scalable Web server clusters can be built using one or more split server systems, each consisting of 3-4 servers. More studies are also needed to evaluate the security benefits of split server clusters, and their scalability and performance with a variety of workloads.

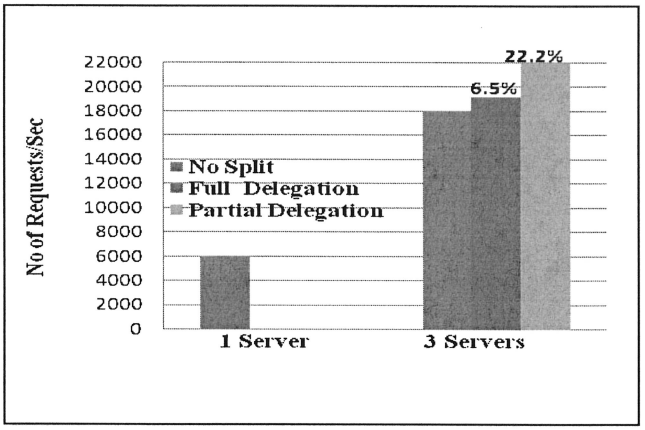


Figure 7: Throughput with full/partial delegation (configuration 3, file size 4KB)

5 Modified Client/Server Architecture

The nature of the interaction between the CS, DS and the client is evident on examining Figures 1 and 2. The CS handles all connections and it communicates with the client by sending and receiving packets. The DS only communicates with the client by sending data to it. The CS also sends an inter-server packet to DS to provide client's request and its state. The CS is connected to the client throughout its session and during the processing of a given request. In this architecture, one CS interfaces with one or more DSs to provide client services and thus becomes a bottleneck in a given mini-cluster configuration [23].

To address this problem, the client-server architecture was modified. In this new architecture [25], connections and data transfers are separated entirely as shown in Figure 8. The data servers (one or more) and their counterpart connection server can be on the same subnet or on different subnets within the same LAN. The CSs can be monitored for ongoing connections with the clients, while isolating the data servers

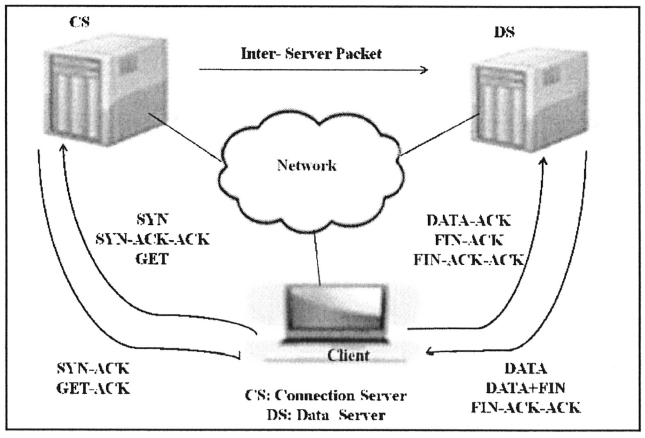


Figure 8: Split protocol architecture (modified client/server architecture)

from the clients. Splitting the connections among the CS and DS servers enables increased security to be provided at a server level. When a connection is established between a client and a CS, the CS will send an inter-server packet to a DS and terminate its connection processing; the DS will send the data and close the connection. Notice in Figure 8 that the client sends the SYN, SYN-ACK-ACK and GET to CS and CS sends SYN-ACK and GET-ACK to the client. After CS processes the connection, it sends a message to DS through an inter-server packet and eliminates this connection at CS. The rest of the connection and interactions related to DATA, ACK and FIN-ACK will be processed by DS. We have freed up CS completely after the GET is processed, but the client must be made aware of the DS as described in [25].

5.1 Configurations

Figure 9 shows a general configuration for connecting one CS, one or more DSs and one or more clients. A resource file size of 64K is used throughout our measurements. The CS will delegate all its requests to one or more DSs for data processing.

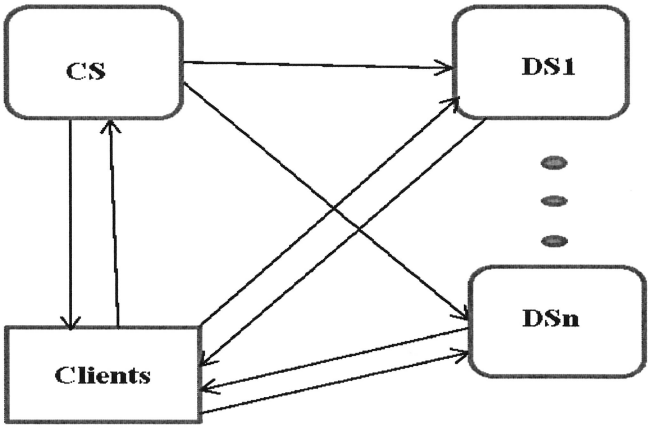


Figure 9: Split Architecture Configurations (Modified Client/Server Architecture)

5.2 Experimental Results

In this system configuration, as CS does not wait until the connection is closed, the CS utilization was very low, so more DSs can be added to the configuration to serve the clients. A linear performance improvement was achieved with up to 4 DSs added to the configuration. This linear performance gain is expected until the CS gets saturated. The linear performance is also expected because the CS causes no bottleneck and all DSs execute concurrently and independently to process client requests. The number of DS servers connected to a single CS server can be estimated to be 15, by extrapolating the observed results. This observation can be used when forming large cluster configurations with split servers. Although the Google architecture uses clusters [2], no protocol splitting is involved.

6 Server Migration

Novelties derived from the split protocol concept can be leveraged in developing server migration architectures [24]. Server migration is usually needed in catastrophic situations or to perform backups and recovery. The split protocol architecture has a minimum of two servers (CS and DS). These two servers have the same state and context of a given request. The CS has the state of a request until GET arrives and the DS has the same state of the request until the data is sent to the client. In essence, the DS acts as a shadow server for CS for a given request i.e., split protocol servers inherently support redundancy and reliability by design. There is no need for explicit shadow servers in split protocol server systems.

The bare server migration architecture proposed here is not same as process migration [19] as there is no process context in the state of a request in a bare implementation. Mobile agents, Aglets (Java based agent technology from IBM), mobile virtual desktop computing [3], and similar ideas have been investigated previously by other researchers. Mobile Web servers [4] provide Web services for mobile devices. Most of these earlier approaches are based on a computing environment where there is some sort of resident operating system. Typically, mobility is then implemented at an application level and the client may have to be aware of migration either through different code or by making a new connection to the server. None of these approaches can handle switch over within a single request.

In bare split protocol servers, the state of a given request is stored in a TCP table whose entries are referred to as TCB entries. Each TCB entry has the complete state of a given request, which consists of about 168 bytes. If this state is moved from a CS to another CS, the latter will simply resume execution from this current state. The 168 byte packet containing the TCB entry can be sent to another CS as an inter server packet with a special tag. When this special tagged packet arrives, the new CS saves the TCB entry in its own TCP table and takes over processing the request.

Migrating servers in this manner poses several networking challenges since there are no changes made to the client and the client has no knowledge of the migration process. One of the main difficulties is to deliver the client connection to the new server who needs to use the same IP as the old server when communicating with the client. This change needs to occur when the old server crashes, or it wishes to hand its connections over to the new server. Another difficulty is migrating existing requests in a CS on one network to a CS in another network. The number of requests residing in a given CS depends on the current client load and the remaining requests to be serviced. Both issues were addressed in a LAN environment by making minimal changes to the server code to handle the migration process. The following sub-sections give more details of the architecture, design, implementation and measurements relevant to migratory bare servers.

6.1 Migration Architecture

In order to implement server migration, a test LAN was set up as shown in Figure 10. The CS, DS and client operate as described in earlier sections. Section 6.3.1 provides more details of the network. The new server CS* to which CS will migrate is connected to the network, but it will not process any client requests and is in stand-by mode. Note that CS and CS* are on different subnets. The relevant IP and MAC addresses used in our case are shown in the figure. When the client sends a request to the CS, it is delegated to the DS and the DS serves the data to the client. In our network, the client request goes through Router#3 and Router#2 via Switch#0 to reach CS. The relevant routing table entry at Router#3 shows 10.55.12.0 → 10.55.10.10 since Router#2 is the next hop to reach the 10.55.12.0/24 network. Once CS* takes over, packets from the client to CS (with destination address 10. 55.12.241) will need to be sent by Router#3 to CS*, which resides on the 10.55.10.0/24 network.

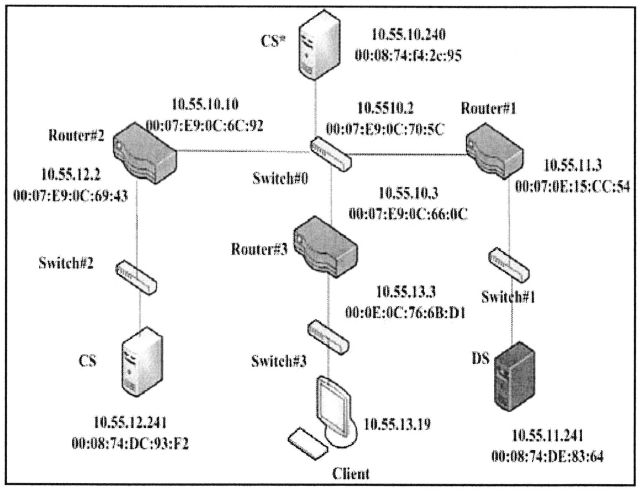


Figure 10: Test LAN for migration

6.2 Design and Implementation

As described in [24], the bare CS design is modified to enable migration capabilities in the server. When a CS is ready to migrate (as triggered by a timing point), it sends all of its pending requests to CS*, which takes over the role of CS. The CS has prior knowledge of CS* and its IP address. Prior to migration, inter-server packets are being sent from the CS to the DS as usual when each GET request arrives. Under large load conditions, it is possible that CS could have many unprocessed requests in the TCB table. In addition to these pending requests, new requests may still come in after the migration event is triggered. These requests will be lost and the client will retransmit them. The retransmitted requests will be processed by CS*. Before migration, CS sends a final inter-server packet to CS* to confirm it is handing over. After this, CS* will send an ARP broadcast to enable its MAC address to be bound to the IP address of CS. Minimal modifications were made to the current server design and inter-server packet code to implement server migration. In essence, the bare PC server split protocol design is easily adapted for migration.

An alternative configuration for migration is to have the CS send its pending requests to its DS before handing over. The DS receives the pending requests and stores them in its TCB. The DS may still have some previous data transfer requests to be processed and sent to clients, so it acts as a DS and CS at the same time until all the previous data requests are processed before turning into a CS (i.e., CS*). CSs and DSs are designed to take either a CS role, or a DS role, or both at any given point in time. This configuration is unique to the bare PC split protocol server design and can be implemented with no additional logic or complexity.

We now briefly describe the implementation details of server migration for our test LAN. Recall that CS and CS* are on different subnets (Figure 10). After CS* takes over from CS, until the new route from the client to CS* which is located at 10.55.10.240 is set up through the relevant routers, packets sent by the client to CS are not delivered. However, the client is unaware of server migration and continues to send packets using the TCP connection to CS. To set up the new route, the Linux script program in Figure 11 was written. It enables

```
# monitor routes
while [ ${_unl} -gt 0 ]
do
 if [ $( ping -c 1 ${_ip1} |grep -ic "100% packet loss" ) -gt 0 ];then
   if [ "${_interf}" = "10.55.10.10" ]; then
     if [ ${_flg0} -eq 0 ]; then
       ip r d ${_ip1} via ${_interf}
       if [ "$?" -ne 0 ]; then
         ip r d ${_ip1} via 10.55.10.3
       fi
       ip r a ${_ip1} via 10.55.10.3
       echo "New route added 10.55.10.3"
       _flg0=1
       _flg3=0
       _interf="10.55.10.3"
     fi
      elseif [ "${_interf}" = "10.55.10.3" ]; then
       if [ ${_flg3} -eq 0 ]; then
       ip r d ${_ip1} via ${_interf}
       if [ "$?" -ne 0 ]; then
         ip r d ${_ip1} via 10.55.10.10
       fi
         ip r a ${_ip1} via 10.55.10.10
       echo "New route added to 10.55.10.10"
        _flg3=1
        _flg0=0
        _interf="10.55.10.10"
     fi
   fi
  # avoid the dead loop
   let i=i+1
   if [ $i -eq 3 ]; then
    exit 1
   fi
  else
   sleep 2
  fi
  if [ $j -eq 10 ]; then
   j=0
   echo $j
  else
   echo $j
   let j=j+1
  fi
done
```

Figure 11: Snippet of Linux Script to monitor IP routes

Router#3 to monitor the status of its route to CS. If the router detects that CS is no longer reachable via Router#2, the program will automatically create the new route to CS*, which is now using the IP of CS. In our configuration, Router #3 constantly monitors eth0 (10.55.10.3) for connectivity with CS (10.55.12.241) by sending an ICMP message (ping request) to CS. After CS is no longer functioning, host unreachable messages with 100 percent packet loss are received by Router#3. The router then deletes the old route to CS and adds the new route to CS*. During this process, the client may lose connectivity for a short period (a few microseconds).

Implementation of the above was done in C/C++. The code sizes are similar to our previous designs [22-23] since only minimal changes were made. The implementation of the HTTP protocol and the necessary network protocols were based on a state transition diagram. The bare PC Web server runs on any Intel-based CPUs that are IA32 compatible. It does not use any hard disk, but uses the BIOS to boot the system. A USB is used to boot, load, and store a bare PC file system (raw files at this point). There was no need to change any hardware interface code during Web server modification to handle splitting or migration.

The software is placed on the USB and includes the boot program, startup menu, AO executable, and the persistent raw file system (used for resource files). The bootable USB containing this information is generated by a tool (designed and run on MS-Windows), which creates the bootable bare PC application for deployment.

6.3 Performance Measurements

6.3.1 Experimental Setup. The experimental setup in Figure 10 involved Dell Optiplex GX260 PCs with Intel Pentium 4, 2.8GHz Processor, 1GB RAM and Intel 1Gbps NIC card on the motherboard. The Ethernet switches were 1 Gbps 16-port Linksys. A Linux-based http_load [12] stress tool and a bare PC Web client were used for the experiments. Each http_load stress tool can run up to 1000 concurrent HTTP requests/sec. Each bare PC Web client can run up to 5700 HTTP requests/sec. A mixture of bare and Linux clients along with bare split servers were used to measure the performance of the split servers. Other popular browsers running on Windows and Linux (Internet Explorer and Firefox respectively) were also used to test the split servers for functionality.

6.3.2 Measurements. Our measurements focus on how many requests were pending when CS handed over, how long it took for the CS to migrate those requests to CS*, how long it took for those migrated requests to be processed at CS*, the relation between pending requests and the current CS load, and how much time was needed to migrate a server. As the split protocol servers are implemented on a bare PC, data collection was much easier.

Figure 12 shows a plot of the number of requests against the server load. The server load was measured in terms of the number of requests/sec for a 4KB resource file. As the load increases, there will be more requests waiting to be processed or delegated to DS. In our system, there are small number of pending requests at any given point in time as most of the requests were already delegated to DS as soon as their GET arrived. An inter-server packet with all the needed information is sent to DS for each delegated request. The experiments showed that about 1-138 requests were still pending in the server for a load variation of 1000 – 7000 requests/sec. These pending requests increase rapidly after

5000 requests/sec since it is getting saturated (higher CPU utilization).

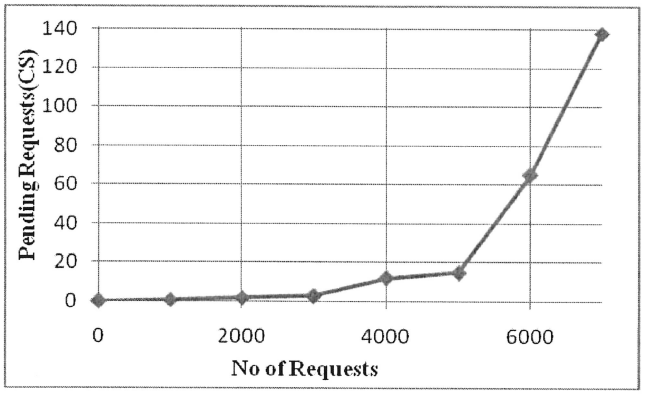


Figure 12: Pending requests at CS with 4KB resource file

In Figure 13, a graph showing the time spent in migrating pending requests from one CS to CS* is drawn. It takes between 1-13 milliseconds to migrate these pending requests from CS to CS*. Under heavy load conditions there is only a time lapse of 13 milliseconds to migrate all pending requests between the two CSs which are on different networks.

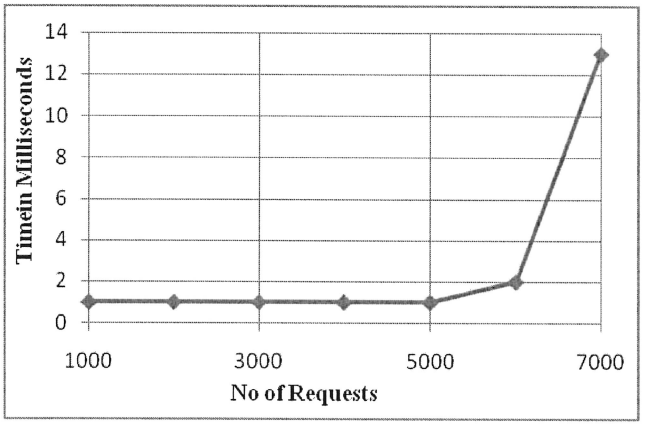


Figure 13: Time to migrate pending requests at CS with 4KB resource file

Figure 14 shows the number of pending requests for variable file sizes (4K to 128K) and a constant load of 500 requests/sec. Notice that the pending requests increase sharply up to 260 for a 128K file size. Larger file sizes have higher connection times (time to process the whole request) and thus more requests will be pending in the server as expected.

Figure 15 shows the actual time spent in migrating requests from CS to CS*. The times measured, range from 1 – 37 milliseconds for file sizes up to 128K. As the resource file size increases, more time is needed to migrate all pending requests as there are more requests in the server that are unprocessed. The migration time depends on the number of requests/sec (load) and the average resource file size.

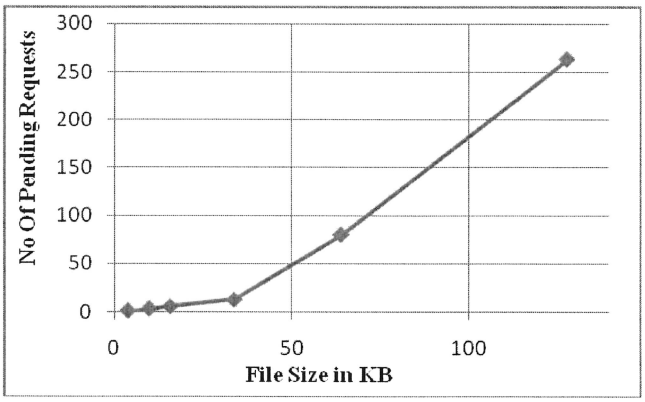


Figure 14: Pending requests at CS with 500 requests/sec load

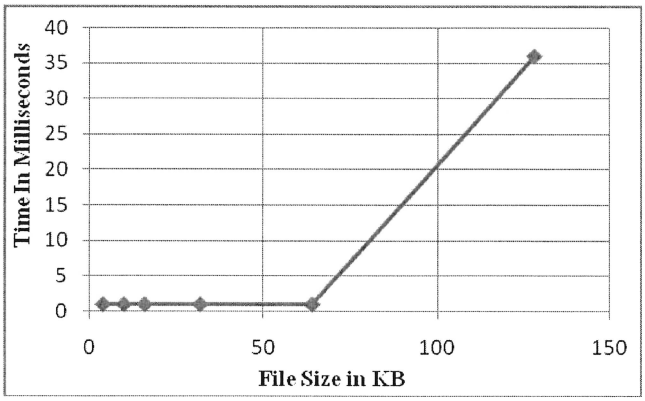


Figure 15: Time to migrate pending requests at CS with 500 requests/sec load

To get further insight into migration delays, the pending requests in CS were simulated by artificially delaying request processing and collecting the migration time as shown in Figure 16. In this case, the load on the server is kept constant. Notice that the migration time is linear as expected.

Figure 17 illustrates the connection time and response time with and without migration for split protocol servers (CS and

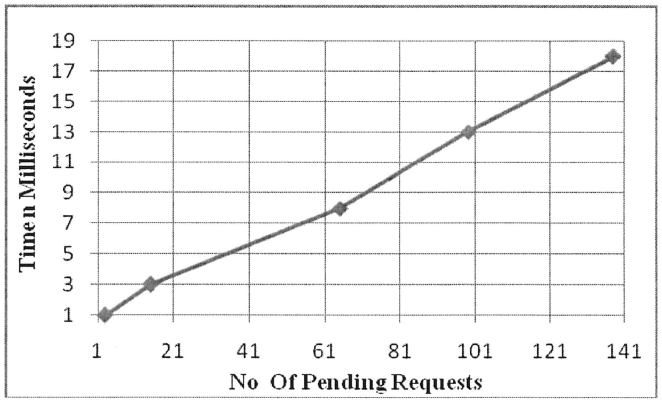


Figure 16: Time taken to migrate pending requests at CS

DS) with varying loads (requests/sec). CT and RT are the times with normal operation and CT-M and RT-M are the times with migration. The graph shows the degradation in connection and response time due to migration. It is seen that the increase in connection and response time is very small for loads of up to 3000 requests/sec. These times increase rapidly from this point due to the large load and the migration delay due to the increased overhead. A maximum load size of 5000 requests/sec was used in these experiments.

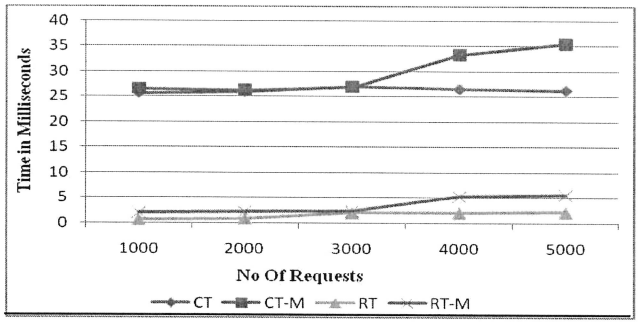


Figure 17: Connection time and response times (with and without migration)

Figure 18 shows the CPU utilization on the split protocol servers CS and DS. Notice that CS and DS reach over 90 percent CPU utilization for large loads (7000 requests/sec) with 4K resource file size. As shown in the graph, CS reaches saturation at 7000 requests/sec load. Further experiments are needed using migration with mini-clusters and splitting as in [23] to overcome the server saturation problem and to demonstrate the scalability of mini-clusters in the presence of migration.

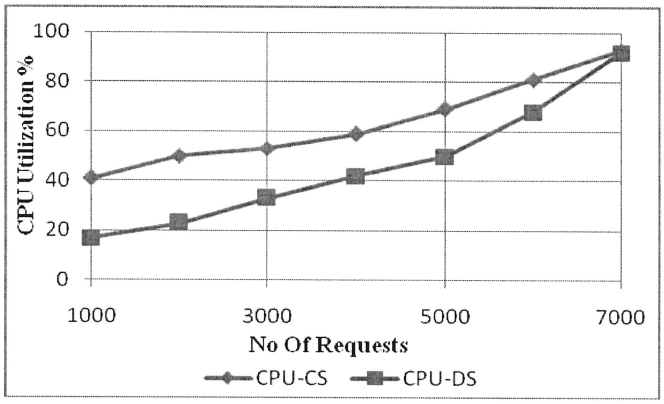


Figure 18: CPU utilization

Figure 19 illustrates the number of lost requests due to server migration. About 14-56 requests were lost for varying loads up to 7000 requests/sec. These requests are lost since it is not possible to process incoming requests during migration. The current server code was not enhanced to handle these requests; it is however possible to reduce lost requests by simply adding them into a pool of pending requests. In the current split design, pending requests only consist of requests with received GET commands from the client, so a different split request point is required to recover all lost requests. Once the migration process is complete, all lost requests will be reissued to CS*. Even with the current design, the above performance measurements demonstrate that the split protocol concept is well-suited for constructing reliable server clusters with inherent redundancy.

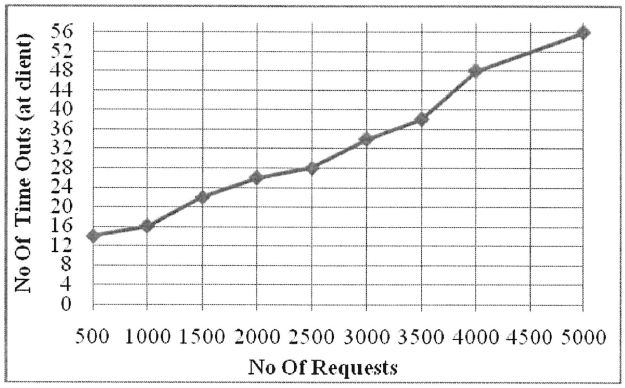


Figure 19: Lost requests at client

7 Significance of Split Protocol Concept

Splitting is a general approach that can be applied in principle to any application protocol that uses TCP (it can also be applied to protocols other than TCP to split the functionality of a protocol across machines or processors). In particular, splitting the HTTP protocol has many impacts in the area of load balancing. We discuss some of these impacts below.

- Split protocol configurations can be used to achieve better response and connection times, while providing scalable performance. Splitting also eliminates the need for (overhead/cost associated with) external load balancers such as a dispatcher or a special switch.
- Split protocol Configuration 2 (with one CS, one DS) and partial delegation achieves 25 percent more performance than two homogeneous servers working independently. This performance gain can be utilized to increase server capacity while reducing the number of servers needed in a cluster.
- Split server architectures could be used to distribute load based on file sizes, proximity to file locations, or security considerations.
- The results obtained (using specific machines and workloads) indicate that mini-cluster sizes are in the single digits. More research is needed to validate this hypothesis for other traffic loads. However, if we assume that mini-clusters should contain a very small number of nodes, they would be easier to maintain and manage (compared to larger clusters). Using mini-clusters, it is possible to build large clusters by simply increasing the number of mini-clusters.
- Splitting protocols is a new approach to design server

clusters for load balancing. We have demonstrated splitting and built mini-cluster configurations using bare PC servers. However, the general technique of splitting also applies to OS-based clusters provided additional OS overhead can be kept to a minimum (and that undue developer effort is not needed to tweak the kernel to implement splitting).

- When protocol splitting uses two servers (CS and DS), it dramatically simplifies the logic and code in each server (each server only handles part of the TCP and HTTP protocols unlike a conventional Web server that does both protocols completely). Thus, the servers are less complex and inherently have more reliability (i.e., are less likely to fail).
- Splitting can also be used to separate the "connection" and "data" parts of any protocol (for example, any connection-oriented protocol like TCP). In general, connection servers can simply perform connections and data servers can provide data. It can also be used to split the functionality of any application-layer protocol (or application) so that different parts of the processing needed by it are done on different machines or processors. Thus, a variety of servers or Web applications can be split in this manner. This approach will enable new ways of doing computing on the Web to be investigated.
- Split protocol servers used in migratory servers will provide reliability and simplicity which is inherent in the design.
- Migratory architectures designed on the basis of split protocol models can prevent expensive shadowing, backup and recovery techniques.

7.1 Security Issues

Protocol splitting and migration raise some security issues even though the servers are on the same LAN. When splitting is used, the DS sends packets back to the client using the source IP address of CS. This means that DS is allowed to spoof the IP address of CS in its outgoing packets. Likewise after migration, CS* will send and receive packets from clients using the IP address of CS. If CS* and CS are on different subnets, CS* is using an IP address whose prefix does not match the IP prefix of its subnet. Also, in our test LAN, we did not consider the impact on local clients and routers in subnets affected by giving CS's IP address to CS*. These local routers, for example, would need to be informed of the change so that they can route CS's packets to CS* (which is using CS's IP address in a different subnet). To minimize the security impact of splitting and migration, it is therefore best that CS, DS and CS* be located on the same subnet. If CS and CS* are located on the same subnet, the ARP broadcast by CS* is all that is needed to ensure that the network operates correctly after migration (it is not necessary to run the script for a route change). It is also necessary to authenticate inter server packets.

8 Conclusions

In this paper we discussed the novel concept of split protocols and their applications. Mini-cluster configurations and modified client server architectures were presented. Server migration and its impact on migration time and lost requests were measured. Full delegation and partial delegation concepts were used to achieve high performance. It is shown that split protocol servers inherently provide reliability, simplicity and scalability. Split protocol designs and migratory servers can be used to build high performance server clusters in the future.

References

[1] P. Appiah-Kubi, R. K. Karne and A. L. Wijesinha, "Design and Performance of a Webmail Server on Bare PC," *HPCC 2010*, pp. 521-526, 2010.

[2] L. A. Barroso, J. Dean, and U. Urs Hölzle, "Web Search for a Planet: The Google Cluster Architecture," *IEEE Micro 23*, 2:22-28, 2003.

[3] R. Baratto, S. Potter, G. Su, and J. Nieh, "MobiDesk: Mobile Virtual Desktop Computing, *Mobicom 2004*, Philadelphia, PA, 2004.

[4] G. Canfora, G. Di Santo, G. Venturi, E. Zimeo and M. V. Zito, "Migrating Web Application Sessions in Mobile Computing," *Proceedings of the 14th International Conference on the World Wide Web*, pp. 1166-1167, 2005.

[5] The Chromium Projects, available at http://www.chromium.org/, accessed: Feb. 3, 2014.

[6] G. Ciardo, A. Riska and E. Smirni, "EquiLoad: A Load Balancing Policy for Clustered Web Servers," *Performance Evaluation*, 46(2-3):101-124, 2001.

[7] A. Cohen, S. Rangarajan, and H. Slye, "On the Performance of TCP Splicing for URL-Aware Redirection," *Proceedings of USITS'99, The 2nd USENIX Symposium on Internet Technologies & Systems*, October 1999.

[8] D. Engler, *The Exokernel Operating System Architecture*, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, October 1998.

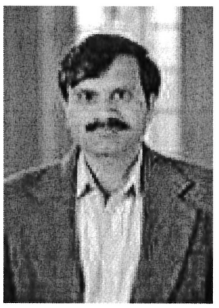
[9] B. Ford, M. Hibler, J. Lepreau, R. McGrath, and P. Tullman, "Interface and Execution Models in the Fluke Kernel," *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, USENIX Technical Program, New Orleans, LA, pp. 101-115, February 1999.

[10] G. H. Ford, R. K. Karne, A. L. Wijesinha, and P. Appiah-Kubi, "The Performance of a Bare Machine Email Server," *21st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2009)*, IEEE/ACM Publications, So Paulo, SP, Brazil, pp. 143-150, 28-31 October 2009.

[11] L. He, R. K. Karne, and A. L. Wijesinha, "Design and Performance of a Bare PC Web Server," *International Journal of Computer and Applications*, Acta Press,

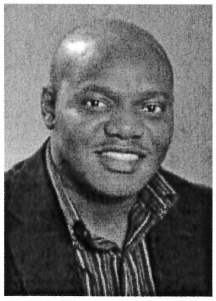
15:100-112, June 2008.
[12] http_load, available at http://www.acme.com/software/http_load, accessed: Feb. 3, 2014.
[13] Y. Jiao and W. Wang, "Design and Implementation of Load Balancing of a Distributed-System-Based Web Server," *3rd International Symposium on Electronic Commerce and Security (ISECS)*, pp. 337-342, July 2010.
[14] R. K. Karne, "Application-Oriented Object Architecture: A Revolutionary Approach," 6th International Conference, HPC Asia 2002, Center for Development of Advanced Computing, Bangalore, Karnataka, India, December 2002.
[15] R. K. Karne, K. V. Jaganathan, and T. Ahmed, "How to Run C++ Applications on a Bare PC," SNPD 05, 6th *ACIS International Conference*, IEEE, pp. 50-55, May 2005.
[16] R. K. Karne, K. V. Jaganathan, T. Ahmed, and N. Rosa. "DOSC: Dispersed Operating System Computing," OOPSLA 05, *20th Annual ACM Conference on Object Oriented Programming, Systems, Languages, and Applications*, Onward Track, ACM, San Diego, CA, pp. 55-61, October 2005.
[17] S. Lampoudi and D. M. Beazley. "SWILL: A Simple Embedded Web Server Library," FREENIX Track: 2002 USENIX Annual Technical Conference, Monterey, California, June 2002.
[18] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen, and R. Brightwell, "Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing," *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2010)*, pp. 1-12, April, 2010.
[19] D. S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler and S. Zhou, "Process Migration," *ACM Computation Surveys*, 32(3):241-299, September 2000.
[20] The OS Kit Project, "School of Computing," University of Utah, Salt Lake, UT, http://www.cs.utah.edu/flux/oskit, June 2002.
[21] V. S. Pai, P. Druschel, and W. Zwaenepoel. "IO-Lite: A Unified I/O Buffering and Caching System," *ACM Transactions on Computer Systems*, ACM, 18(1):37-66, February 2000.
[22] B. Rawal, R. Karne, and A. L. Wijesinha, "Splitting HTTP Requests on Two Servers," The Third International Conference on Communication Systems and Networks: COMSNETS 2011, Bangalore, India, January 2011.
[23] B. Rawal, R. Karne, and A. L. Wijesinha, "Mini Web Server Clusters for HTTP Request Splitt," 13th International Conference on High Performance Computing and Communication, HPCC-2011, Banff, Canada, Sept. 2-4, 2011.
[24] B. S. Rawal, R. K. Karne, A. L. Wijesinha, S. Ramcharan, and S. Liang, "A Split Protocol Technique for Web Server Migration," International Workshop on Core Network Architecture and Protocols, (ICNA), 2012.
[25] B. S. Rawals, R. K. Karne, and A. L. Wijesinha, "Split Protocol Client/Server Architecture," IEEE Symposium on Computers and Communications, (ISCC), 2012.
[26] K. Sultan, D. Srinivasan, D. Iyer and L. Lftod, "Migratory TCP: Highly Available Internet Services using Connection Migration," *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pp. 17-26, July 2002.
[27] "Tiny OS," Tiny OS Open Technology Alliance, University of California, Berkeley, CA, I http://www.tinyos.net/, 2004.
[28] S. Vaidya and K. J. Chritensen, "A Single System Image Server Cluster using Duplicated MAC and IP Addresses," *Proceedings of the 26th Annual IEEE Conference on Local Computer Network (LCN'01)*, pp. 206-214, 2001.
[29] T. Venton, M. Miller, R. Kalla, and A. Blanchard, "A Linux-Based Tool for Hardware Bring Up, Linux Development, and Manufacturing," *IBM Systems Journal*, IBM, NY, 44(2):319-330, 2005.
[30] D. Wentzlaff and A. Agarwal, "Factored Operating Systems (FOS): The Case for a Scalable Operating System for Multicores," *ACM SIGOPS Operating Systems Review*, 43(2):76-85, April 2009.

Bharat Rawal is an Assistant Professor in the Department of Computer Science and Information Systems at Shaw University. He has a doctorate in Information Technology and an MBA from Towson University, and a BSc in Physics from South Gujarat University, India. His research interests are in split protocol systems and bare machine computing.



Ramesh K. Karne is a Professor in the Department of Computer and Information Sciences at Towson University. He obtained his Ph.D. in Computer Science from the George Mason University. Prior to that, he worked with IBM at many locations in hardware, software, and architecture development for mainframes. He also worked at the Institute of Systems Research at the University of Maryland, College Park as a research scientist. His research interests are in bare machine computing, networking, computer architecture and operating systems.

Alexander L. Wijesinha is a Professor in the Department of Computer and Information Sciences at Towson University. He holds a Ph.D. in Computer Science from the University of Maryland Baltimore County, and both a M.S. in Computer Science and a Ph.D. in Mathematics from the University of Florida. He received a B.S. in Mathematics from the University of Colombo, Sri Lanka. His research interests are in computer networks including wireless networks, VoIP, network protocol adaptation for bare machines, network performance, and network security.



Patrick Appiah-Kubi earned a doctorate in Information Technology from Towson University, an MS in Electronics and Computer Technology from Indiana State University, and a BS in Computer Science from Kwame Nkrumah University of Science and Technology, Ghana. He is an Assistant Professor in the Department of Electronics and Computer Engineering Technology at Indiana State University. His research interests are in bare machine computing systems, networks, and security.

Songjie Liang obtained his doctorate in Information Technology at Towson University. He is also an independent IT consultant, working for Lockheed Martin, Federal Government DOC, IRS, and USPS. His research interests include bare machine computing, file management systems, USB drivers, software engineering, database management systems, systems administration and Linux operating systems.