Survey of Bare Machine Computing Systems and Applications

Ramesh K Karne
Computer Information Sciences
Towson University
Towson, USA
rkarne@towson.edu

Alexander L Wijesinha
Computer Information Sciences
Towson University
Towson, USA
awijesinha@towson.edu

Nirmala Soundararajan
Computer Information Sciences
John Massey School of Business
Southeastern Oklahoma State University
Durant, USA
nsoundararajan@se.edu

***Abstract*—For over 30 years, a bare machine computing paradigm and its applications were researched as a practical means to build computer systems and applications. A bare machine is a computing device without any software. It contains no persistent storage and no operating system. The machine cannot run unless a system application is loaded from an external device. Bare machine computing is a general-purpose computing paradigm-it is not a dedicated application, a bare metal concept, or an embedded system approach. It reduces obsolescence and is inherently secure by design. This computing paradigm was the basis for building a variety of bare machine systems and applications, resulting in twenty-seven doctoral dissertations and over 70 publications. The purpose of this survey is to consolidate the research in one place to serve as a basis for future exploration of the paradigm. It describes the work in several application domain areas highlighting the novelties in architecture, design, and implementation. The computer system industry and researchers could investigate using bare machine computing principles to build all computer and information systems. Reducing obsolescence is essential to building systems that last for a long time. Designing systems based on a bare machine computing paradigm will help to achieve this goal and build systems characterized by simplicity and improved security.**

Keywords—Bare Machine Computing, Obsolescence, Bare Internet, Domain-Specific Applications, Direct Hardware Interfaces, Cybersecurity.

I. Introduction and Motivation

Over a period of many years, Dr. Karne observed that current computer and information systems get obsolete rapidly and create waste in people's skills, products, and services, resulting in recurring costs to customers in hardware and software. He witnessed and experienced obsolescence first-hand in his hardware and software careers spanning over fifty years. Today, to build computer and information systems, consistent engineering principles are not followed, although these are used in constructing monuments and other notable structures. Developers also do not follow object-oriented principles to their fullest extent. This frustration motivated the author to address these issues and resulted in his first paper on "Object-oriented Computer Architecture for New Generation of Applications [1]," in 1995 followed by papers on the application-oriented object architecture (AOA) concept. This early work laid a foundation to develop the bare machine computing (BMC) paradigm and build real-world systems and applications based on the paradigm. Subsequent research based on it resulted in twenty-seven doctoral dissertations, and close to 70 publications at Towson University. Dr. Wijesinha joined him to pursue this monumental work, bringing his network and network security expertise into building bare computer systems and applications. Many doctoral students worked relentlessly and made significant contributions to this research. The combined efforts resulted in demonstrating the feasibility and value of the BMC paradigm. BMC is a general-purpose computing paradigm; it is not intended for use in the Internet of Things (IoT) and Graphics Processing Units (GPU's). BMC servers can interact with any other OS based clients with standard TCP/IP protocols. Currently, BMC applications can run on any Intel based platforms as the device drivers are only written for Intel architecture.

This paper surveys BMC history, research and publications. It serves as a foundation for building future computer and information systems characterized by simplicity, longevity, and inherent security.

II. Overview of the BMC Paradigm

The BMC paradigm is a computing paradigm based on bare machines. In the 1960's, computers were essentially bare machines, and thus, the paradigm is not completely a new idea. In this paradigm, applications run directly on bare machines without requiring an operating system (OS), centralized kernel, or any intermediary software. A bare machine does not run until external software is loaded. It is important to note that a BMC system is different from a bare-metal or embedded system [2, 3], which are dedicated applications. The BMC paradigm is a general-purpose computing paradigm that can be used to build any computer system or application. In addition, it is designed to reduce obsolescence and achieve high levels of security. It is also different from the bare-metal server concept in cloud computing because they have some form of an OS [3]. Fig. 1 (a) and Fig.1(b) show the essential differences between conventional and BMC systems.

The BMC paradigm is a two-pronged approach. First, a computer device is bare, which means it does not have any pre-installed software or persistent storage. A bare computing device has only the fundamental computing components: the central processing unit (CPU), main memory, and input/output (I/O)

peripheral devices. There is no OS, kernel, middleware, or other intermediary software running on a bare device. Bare machine applications are stored on a mass storage device such as a flash drive. Unlike conventional OS-based systems, the bare machine has no ownership, and it has no valuable resources in the box. In the BMC paradigm, the bare machine serves as a public platform for running BMC applications.

Second, BMC is an application-oriented approach, whereas conventional systems are based on evolutionary and environment-centric platforms. In the BMC approach, computer applications directly communicate with hardware without any middle layer. Developing an end-user BMC application is therefore different from developing applications that run on top of an OS or an embedded system. In this paradigm, the programmers serve as both system and application programmers as they need to understand important technical aspects related to the underlying hardware. The BMC applications directly communicate with and control the hardware using direct hardware API (HAPI), which is public. Applications are written in C/C++ with a small amount of assembly code while the boot, loader, and interrupt code are written in assembly code.

BMC applications are designed to be self-contained, self-managed, and self-controlled entities. They include all necessary components within their code, such as device drivers, network protocols, and support functions enabling them to operate independently. The necessary hardware drivers and network protocols are part of the application and integrated with it. It is also an application-to-application (A-to-A) model. Most of the details for users to communicate are hidden in the application. One or more BMC applications are compiled as an application suite to generate a single monolithic executable. No external software or modules are used to compile and link bare applications. There are no OS-related functions, and each application only contains the necessary minimal functionality. More specifically, the BMC concept is a minimalist approach to achieve simplicity, smaller code sizes, and inherent security. There is no privileged mode in a BMC system since applications only run in user mode. As application code is statically compiled, it is not possible for attackers to alter BMC program flow during the execution.

A domain specific application contains a particular application that runs on a bare machine. For example, a VoIP application communicating between two users is a domain specific application. Likewise, a Web server providing services to clients is a domain specific application. A text processor could also be viewed as a domain specific application. These domain specific applications are end user applications that are unique, and they are entities that live for an extended period (due to higher level abstractions and object-oriented methodologies). When one or more domain specific applications is needed for an end user, a few domain-specific applications can be combined into a bare application suite. Only one application suite runs at a given time.

As noted above, all applications are currently written in C++/C and assembly for Intel's x86 and x64 instruction set architectures. The assembly code is minimal in BMC applications (e.g. a USB 2.0 driver has twenty-four lines of assembly code, including comments, and it has only two functions). A system based on the BMC paradigm is a closed system, and it has no layers.

In Fig. 1(a), conventional systems need some sort of middleware, or OS to communicate with the hardware. They explicitly or implicitly use OS system calls or libraries to communicate with hardware. As such, application programs cannot run without some form of middleware. The conventional approach assumes a machine can run any application, which is run by this middleware. At a given point, the system is running many applications in addition to the OS. Application programmers have no control of execution flow. In Fig. 1(b), BMC systems have no middleware or OS. A bare application suite runs by itself without any middleware. It carries all the necessary code for a given application suite and nothing else. Application programmers have total control of execution flow. An application suite only has the necessary direct standard hardware interfaces in it. At any given point, a user does not need all applications to run.

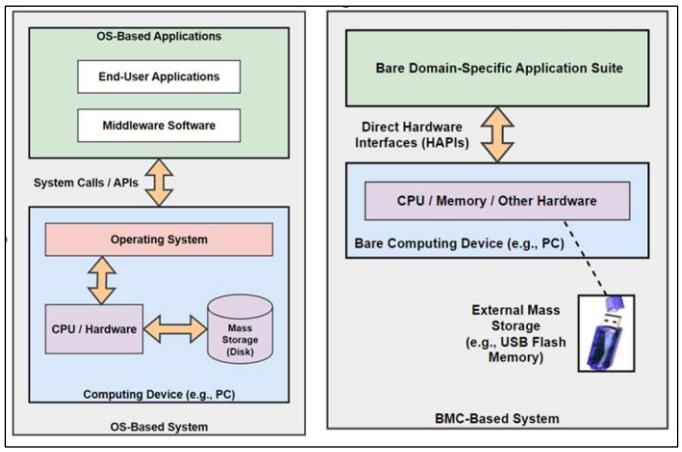

Fig. 1. (a) OS-based system. (b) BMC-based system

III. Obsolescence

Obsolescence is studied in many disciplines such as engineering, economics, business, and computer and information sciences. The comprehensive survey in [4] summarizes key types of obsolescence and gives taxonomy. The most significant classes of obsolescence include design [5, 6, 7], technology [8, 9], planned [10, 11], appearance, and skills. Most people are familiar with product obsolescence associated for example with PCs, laptops, smartphones, TVs, cameras, and electronics. The obsolescence problem is also seen in houses, materials, drugs, vaccines, commodity goods, education, curriculum, and more. Bill Gates stated a business strategy that explains one aspect of obsolescence [12]: *"In three years, every product my company makes will be obsolete."* As products and technology become obsolete, it has a ripple effect on people's skills, longevity, and their value to society [13]. In effect, technology today is designed to become obsolete in a short time. As noted in the literature above, making things obsolete is a part of our culture, business, and society.

As reducing obsolescence is the focus of the BMC paradigm, our research considers this as a main requirement in building computer and information systems. The rest of this paper continues as follows. Section IV narrates related work. Section

V describes steps taken to build a comprehensive list of bare systems that demonstrate the feasibility of BMC systems and applications. This section also describes the crucial components needed to build these systems and applications. Section VI summarizes the architectural, design and protocol novelties explored in BMC research. Section VII provides a summary and gives the conclusion of this paper including some possible future investigations.

IV. Related Work

In the beginning of the computer era, machines had an array of switches and buttons to interact with them. Computer applications were primitive and small, enabling them to contain all necessary software to run on the hardware. Hardware was expensive and software was cheaper. During the evolution of computing, OSs played a crucial role in making the software independent of hardware. However, software was still dependent on the underlying OS and system calls. There is a platform dependency in most software today. OSs became complex and large (20-50 million lines of code), and researchers started moving OS components into applications. Early examples include Exokernel [14], Microkernel [15], Tiny-OS [16], IO-Lite [17], OS-Kit [18], Palacio and Kitten [19], and RIOT [20]. These examples illustrate the research trends to reduce OS/kernel size and complexity such as giving some direct hardware access to applications and moving part of OS functions into user space. The approaches used differ from BMC in that they were not intended for general purpose computing. Moreover, BMC methodologies focus on reducing obsolescence and designing systems resilient to security vulnerabilities. More related work are in the BMC publications referenced in this paper.

V. Survey of BMC Systems and Applications

This section provides a chronology of BMC systems. Each step played a significant role in achieving our objectives and resolving unexpected hurdles encountered on the way. BMC systems and applications evolved over a 30-year period during which they were explored through many doctoral dissertations and graduate projects.

A. Boot and Load Process

To run applications on bare machines, a given application must have its own boot and load mechanisms. We have chosen to build boot and load processes using assembly code that runs in real mode in Intel processors. We have selected a flash drive to contain the boot and load programs, and an application suite. The AOA concept described in [21, 22] has more details on these basics. The C++/C/ASM programming environment chosen to write applications and run on a bare PC are briefly described in [23]. AOA was referred to briefly as dispersed operating system computing (DOSC) [24], which implied that OS functionalities were dispersed into applications. However, this did not correctly represent the approach, and so applications were described as running on a bare PC, interchangeably called a bare machine. This led to the bare machine computing (BMC) paradigm described in [25]. The paradigm gives rise to a revolutionary approach spurring further research into studying and validating it. The essential ideas in BMC are briefly summarized in [26].

B. Device Driver Issues

Because OSs are driven by rapid changes in I/O-technology, vendors and programmers write device drivers for a given OS platform and version. As OS versions change frequently (sometimes every year), device drivers must also change. This causes issues with building BMC applications as there is no OS. This challenge forced us to write our own bare device drivers that work with bare machines. Initially, we wrote an Ethernet driver for a 3-COM network interface card (NIC), which became obsolete in Dell computers. Later, Dell machines used Intel chips as in the Intel 82540 EM NIC in OptiPlex GX 260 models, which was part of its motherboard. We wrote a bare device driver for this NIC and used it as a base enabling Dell models to run BMC applications. We developed our own expertise to build other Ethernet, audio, and USB 2.0 device drivers for Dell computers with Intel 32-bit processors. As expected, processor versions and Dell models become obsolete faster than we could catch up with their changes [9]. We had to stay with the Dell OptiPlex GX260 models for a long time to keep our device drivers working with BMC applications. Some later work on Ethernet drivers [27, 28] and USB drivers [29] enabled BMC applications to continue running on bare machines.

We also found an external NIC for Intel PCs, which was equivalent to the Intel 82540 EM models, enabling us to build a driver to work with other Dell OptiPlex models. However, when Intel changed the processors to 64-bit, the Intel Controller Hub (ICH) became obsolete and a new chip set for the I/O controller was used to build applications. This caused more obsolescence in our device drivers, thus forcing us to stay with old machines. This is a clear illustration of design and planned obsolescence that cripples older systems and products. Most systems become obsolete before their useful lifetime! Unless we write new drivers and replace the older systems, it is not possible to get support for and maintain the machines. Also, older products may not be operational as they cannot interface with the newer versions.

Fortunately, the chosen OptiPlex GX260 model was stable, operational, and helped us to conduct BMC research over a 30-year period. This indicates that hardware can be built to last a long time given a requirement to reduce obsolescence. The preceding work leads us to several observations: USB interfaces are standard, and USB can be a global interface protocol which in turn unifies device drivers in BMC and eliminates all device driver heterogeneity. Application programmers can use direct USB hardware interfaces (HAPIs) in application programs.

C. VoIP

As an example of a domain specific application, we first consider a bare VoIP application over UDP built using a bare NIC driver and a bare audio driver [30]. This application enabled us to demonstrate VoIP communication using two bare machines connected to a LAN environment and later over the Internet. This communication used RTP over UDP. Following this work, a SIP server and user agent were implemented [31], and SIP server performance was studied [32, 33]. For secure VoIP, SRTP was implemented, and its performance was evaluated in [34]. Secure VoIP was also implemented without using SRTP in [35]. VoIP studies used both IPv4 and IPv6 [36].

The VoIP real-world application established a basis to further investigate other domain specific applications described later.

The significant contributions of this work include demonstrating single thread processing and interaction between two clients on a network; installing boot code, loader and application on a USB drive; running real world applications on a bare PC; running SIP servers and user agents and running these applications on Dell OptiPlex 260s.

D. Web Server

Web servers and clients connected over a LAN, or the Internet represent core models used in other applications. The bare Web server domain specific application has many novel architectural, design, and implementation characteristics that were later used in other BMC systems and applications. The Web server also has multitasking and other complex functions that demonstrate the feasibility of bare systems.

1) Client Server Interactions: In BMC, protocols can interact without following traditional layering. In effect, TCP, IP, and other protocols such as HTTP that may be needed by the application are intertwined with the application code. For the case of the bare Web server, the server and client interactions are consolidated to make design and implementation simple. In this approach, Web server functionality is summarized as follows. A client in the form of an ordinary OS-based browser makes a request, and the bare Web server provides a response. However, this simple transaction goes through HTTP and TCP at higher levels, and IP and Ethernet at lower levels. Although messages are sent by the bare Web server so that interoperability with OS-based browsers is preserved, implementation code is not based on layers. Figure 2a illustrates protocol interactions in a typical message exchange that involves TCP and HTTP. This interaction involves connection establishment, data transfer, and connection termination. The figure only shows a simplified exchange to illustrate the concepts. This complexity in protocol interaction as shown in Figure 2a is implemented by using state transition diagrams.

2) Client State Data Structure: Each client request is uniquely identified by its IP address and source port number. The state of each request is stored in a table and addressed with a hash index. The status table that stores the state of each client request is indexed with a hash. This table is referred to as the TCP table or transport control block (TCB). The hash index to address the table is derived from a simple XOR function. In a bare Web server, we created a state table of 20,000 entries and a hash table of 1000 entries. About 14,000 client requests per second can be handled with this TCB in a LAN environment. Testing tools such as HTTP load running on Linux are used to validate and evaluate Web server performance.

3) Bare Web Server Task Structure: An optimized task structure is designed to build high performance Web servers. The same structure is used in other BMC systems and applications. It gives a simple way to implement multi-tasking in BMC systems. This task structure for a Web server is shown in Fig. 2b. The main task runs all the time in this model. When a packet arrives, it runs the receive (RCV) task.

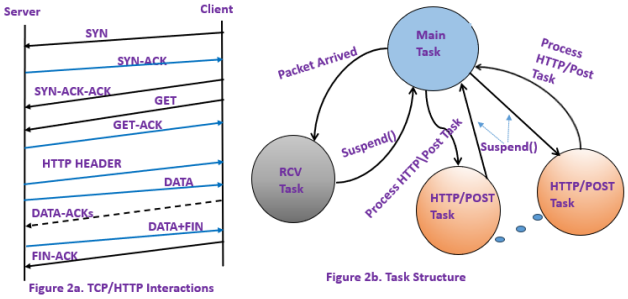

Fig. 2. TCP interactions and BMC task structure

The receive (RCV) task reads a packet from the Ethernet buffer, processes it, updates the TCB and returns to the main task. If a GET request comes from a given client, the RCV task also starts an HTTP task and returns to the main task. A circular list is used to maintain all tasks running in the machine. These tasks simply give more priority to the RCV task and less to the HTTP tasks. There is only one RCV task and one main task in the system. A given application suite may redefine the HTTP task as a general-purpose task and change its behavior as needed. There are no other task types in this simple Web server system. The RCV task runs as a single thread of execution and HTTP tasks are triggered by the RCV task for a GET or POST request. The HTTP and POST tasks are event-driven. The task structure helps to implement the code that handles packets exchanged between clients and the Web server including DATA-ACKS, and timeouts. This novel approach to handle tasks is generic and simple to implement in BMC systems and applications.

4) General Design Issues: A bare Web server system has many designs, and implementation features unique to BMC applications. The following are handled differently than in OS-based systems: applications communicating directly to the Ethernet device driver; multi-threading to handle multiple client requests; a data structure (TCB) to maintain the state of client requests; handling TCP/IP protocols for a client request's state changes; parallelizing client requests; efficient thread/task management; techniques to debug and test parallelism of client requests; optimizing performance; and testing the bare Web server using OS-based stress tools etc. Some of the above resulted in issues that were eventually solved. Thus, successfully implementing a robust Web server on a bare PC posed a daunting challenge to our doctoral students and faculty. At the end, sometimes by trial and error, a variety of novel and appropriate techniques were developed. Web server performance and design issues are discussed in [37]. It is of interest to note that this Web server system was compiled with the Visual Studio 6.0 compiler and Turbo ASM for the boot and loader programs. The Web server was also extended to support dynamic content of a Web site, which involved connecting to a MySQL database to obtain dynamic data (POST requests) [38]. The server used 32-bit application programs and processors. Later, this Web server was upgraded to compile with the Visual Studio 12.0 compiler to run on newer 32-bit machines [39]. The 32-bit Web server models led us to develop other novel Web servers based on the TCP protocol which splits into connection and data phases. The connection phase is managed by one server and the data transfer by a different server. The connection and data transfer servers are connected in series to do the work,

although the data transfer server is hidden from the clients. This novel split TCP protocol was introduced in [40] and its architecture was described in [41]. Based on the split concept, server clusters were designed and shown to achieve high performance [42]. Generic concepts of split servers and their applicability are discussed in [43]. This is one example showing that it is easier to design, implement, and test novel concepts with BMC systems than with conventional systems.

The significant contributions of this work include: studying the design, implementation, and performance of the bare Web server; testing it to verify correct operation and deploying it in real-world systems and applications; developing a novel split protocol concept (resulting in almost linear performance using split Web server clusters) that is applicable to other protocols; laying a foundation to continue further research into bare Web servers and to develop bare Web clients; and showing that the Web server task structure and TCB data structure is applicable to other BMC applications.

E. Email

In this section, we summarize the work done in developing the email domain specific application using the BMC paradigm. Initially, bare email servers and bare clients were used to demonstrate the feasibility of exchanging mail between bare machines. The design and implementation of a bare email server was described in [44]. In this system, SMTP and POP3 were seamlessly integrated with the bare TCP code. The email server was then tested with bare and commercial email clients. The implementation also supported MIME for attachments. The performance of the bare email server was evaluated with standard tools and compared with commercial email servers [45]. It was found that the bare email server outperformed the commercial servers. This research shows the possibility of using bare email servers to communicate with both bare and commercial email clients to support real-world email applications.

The significant contributions of this work include extending the existing bare Web server design and implementation with ease to build a bare SMTP email server; showing the capability of bare systems to interoperate with commercial systems; paving the way to deploy bare email servers and clients in future as isolated systems enabling small groups of users to communicate more securely than with current global email systems.

F. Web-based Email

Further research was conducted to extend the email server design and build a Web-based email domain specific application based on the BMC paradigm. The goal was to use this server to provide mail services to bare clients and commercial email clients over the Web. In [46], the design and implementation of a bare Web-based email server is described and the interoperability of the server with bare and conventional browsers is demonstrated. In [47], performance of the bare Web-based email server was measured using conventional stress tools. In [48], the TLS protocol was added to this server enabling email to be exchanged securely with clients. Performance of the TLS-based system was also evaluated. This design required that a TLS task be added to the Web-based email server task structure to support the TLS handshake and concurrently process HTTP and TLS requests. The TCB table was enhanced to deal with TCP and TLS request states and their operations. A lean version of TLS implemented earlier on the bare Web server was used as the basis for this server.

The significant contributions of this work include extending the existing bare email and bare TLS Web server applications to enable Web-based email with TLS and demonstrating the feasibility of building a bare Web-based secure email system for custom users.

G. Web Client

By inverting the TCP and HTTP packet exchanges between a bare Web server and commercial Web browsers, a simple novel bare Web client was designed for text-based browsing in [49]. The text-based bare browser was used to perform stress tests on Web servers. This research demonstrated the ability to communicate with ordinary commercial Web servers using a bare Web-based text browser. During this research, we also identified and specified direct hardware interfaces based on the BMC paradigm that are necessary to build bare machine systems and applications [50]. It should be possible in future to incorporate these interfaces on a processor chip and eliminate device drivers in computer and information systems. The significant contributions of this work include demonstrating bare text-based browsers and specifying direct hardware interfaces for BMC.

H. File Systems and Database Systems

In [51], a bare USB file system design and implementation based on the FAT32 file specification is described. Also, instead of building our own bare database management system, an OS based SQLite database engine was transformed to run on bare PCs [52]. The transformation process consisted of simply removing the system calls and replacing them with bare direct hardware interfaces [53]. The bare FAT32 interfaces are used to access the bare file system. This process was a daunting challenge because of the need to transform the SQLite code in C, which is about 150K lines of code, without understanding its internals. The bare mass storage system and the interoperability of bare SQLite are described in [54] and [55] respectively. Multiple USBs were used to demonstrate large mass storage systems. This work required the efforts of several doctoral students to integrate the bare file system, bare SQLite, and bare mass storage. In [56], integration of the bare file system with a novel bare email server using bare SQLite was demonstrated. This research used an application suite consisting of three bare applications with the bare SQLite database for storage and retrieval of email messages.

The significant contributions of this work include demonstrating the feasibility of integrating bare file and bare SQLite database systems, and building complex systems based on the BMC paradigm by using the application suite concept.

I. Other BMC Work

In this section, we describe other BMC work including some BMC systems characterized by novel architectures and protocols to illustrate the flexibility of the BMC paradigm.

1) Bare-to-Bare Systems: If we limit domain specific applications to only bare-to-bare systems, bare users can only communicate with other bare users. It is also possible to go further and develop a bare Internet model, where all the nodes on the Internet are bare. This model can incorporate novel architectures, protocols and methodologies that will simplify the design and implementation of computer and information systems. In addition, it allows designers to be more flexible and customize solutions to suit user needs within their respective domains.

Today's computing systems use multicore processor architectures. Design issues relevant to running Web servers on multicore architectures are described in [57]. These issues include (but are not limited to) booting, multiprocessor configurations (MP Architecture), and load balancing. For example, because PCs usually have only one Ethernet network card interface, all cores may try to access packets from the network at the same time. This causes synchronization problems in the multicore system. Locking mechanisms to address synchronization will have an impact on the scalability of performance. In a four-core processor, one core is used for booting and all cores may share the workload. However, performance does not scale in multicore systems due to the bottleneck of a single network card or chip in network applications.

To take advantage of communication involving bare-to-bare domain applications, a new UDP-based protocol is used between the bare Web server and bare clients. The protocol is much simpler than the conventional TCP-based protocol shown in Fig. 2a. It is not the same as the popular QUIC protocol over UDP that supports HTTP/3 and integrates the TLS 1.3 handshake. In the bare protocol, a new 16-byte control header is introduced in each packet to make the server and client designs simple [58]. The packet exchanges for this protocol and the 16-byte control header are shown in Fig. 3. In this design, a four-core processor is used to implement the Web server. The boot processor (BSP) and application processors (AP1, AP2, AP3) all share the workload. However, the BSP also manages sending and receiving Ethernet packets. In addition to simplifying server design, the 16-byte header with ACKs and Data packets helps to achieve reliability as clients and servers exchange this header to use in their respective state machines. The attributes in this 16-byte header simplify client and server implementations.

The BSP became a bottleneck limiting performance scalability. Data buffers are used to hold client requests in all cores, but they did not help to achieve better performance due to the BSP bottleneck. In multicore systems, synchronization and locking mechanisms add complexity to design and implementation. These design issues provided further research avenues to continue this work. In conventional systems, thread level parallelism or data level parallelism is used to improve performance, but there are only user threads in BMC systems and bare Web server applications do not parallelize data. The threads in BMC are a single thread of execution for each request. Thus, for these applications, the single network interface card caused a bottleneck.

Also, processor technology has moved from 32-bit to 64-bit architectures. The shift to 64-bit processors has ripple effects on compilers, applications, OSs, tasking, addressing, booting, loading, and more. This is an example of obsolescence at hardware, firmware, and software levels. To adapt BMC to these changes, considerable efforts were required to enable a BMC application to move to a 64-bit platform.

The BMC applications referred to above were based on a 32-bit Intel architecture. The boot and loader code were written in TASM assembly. All the multicore design issues previously identified were solved to run on 64-bit processors. The boot and loader code were rewritten in NASM assembly. Interrupts and tasking were modified in accordance with the 64-bit architecture. Other compilation and multicore issues were also resolved. In [59], the new architecture was demonstrated and the feasibility of running the previous bare UDP Web server [58] in a 64-bit model was investigated.

A completely updated version of the bare UDP Web server was studied in [60]. The optimized client/server protocol used in this work is shown in Fig. 3. It has two options. One with a last ACK and the other with no last ACK. There are also some differences in data transmissions. With a last ACK, the server sends all the data to a client, and the server waits for the last ACK. With no last ACK, instead of sending all the data at once, the server sends a small number of packets (N) and does not wait for any ACKs. N is usually 4 – 8 packets. When a client needs more data, a subsequent GET with the next starting packet number is sent. The 16-byte control header described above is also used in this protocol to simplify designs on both ends.

Using the above protocols, techniques to improve performance such as PEEKING (each core peeks the network card buffer), locking, and buffering were tried with little success. When all cores are peeking for packets from the same network card, it causes bottlenecks and adds complexity to the design and implementation. Results with and without a last ACK also did not show any significant performance improvements.

After experimenting with many approaches as discussed above to improve multicore UDP Web server performance, a different design was tried that made the server stateless. This new design was used to investigate the performance issues in multicore systems with 64-bit cores. The key feature used in the stateless server is quite simple. Instead of all cores contending for the network, the BSP is used for network communication as shown in Fig. 4. The other cores process the client requests. The BSP receives and sends packets on behalf of the other processors. It does not process any client requests but simply delegates them to the other cores in a round-robin fashion. When a request arrives at an AP, it buffers the packet until the core is ready to process it. Packets to be sent are placed in a buffer until the BSP is ready to send them. This approach avoids all concurrency control issues and locking mechanisms. It also simplifies the multicore server design. Design and implementation details with performance measurements for the stateless multicore bare UDP Web server are given in [61]. The results indicate that for 1 to 2 cores the performance increased by 21.5%, and for 1 to 3 cores by 44.7%. Thus, the speedup is not linear with respect to adding more cores to process HTTP requests. However, these results are better than the previous results for Web server performance in [58] and [60]. This is the

optimal design for bare UDP Web servers with multicore processors. In this design, the BSP became saturated, which limited its optimal performance.

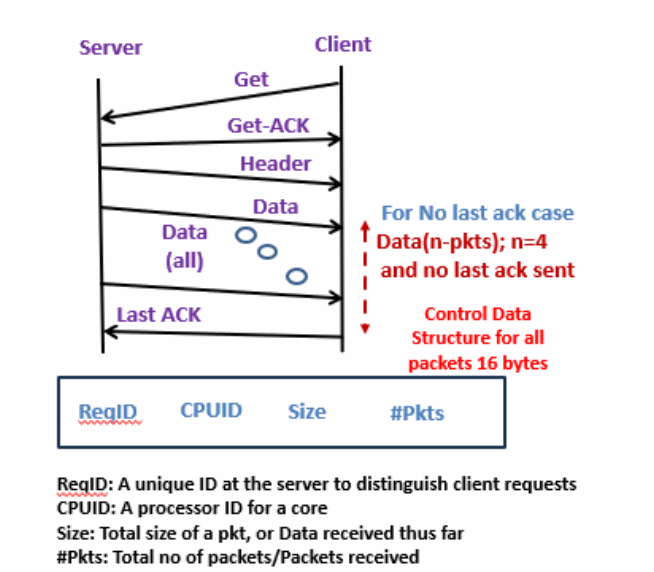

Fig. 3. UDP-based Web server protocols with control data header

The significant contributions of this work include: illustrating that it is possible to design custom protocols and techniques that are unique to a group of users; using a 16-byte control data header in packets to simplify protocol design and client/server interactions; implementing a UDP-based protocol where a client can only send HTTP GET messages to the server to improve security; sending only 4-8 packets at a time to the client making the server stateless and harder to attack; using subsequent GET requests to obtain the remaining data; and dedicating one core to send/receive packets to simplify multicore design and improve performance.

2) Chat Application: The bare chat domain specific application [62] has novel characteristics that make it more secure than conventional chat systems. In this chat application, the server and clients run on bare systems. It was demonstrated that the bare chat application could be used by small groups to communicate securely within their own domain. After checking their credentials, the bare users were given their USB to ensure physical security.

The context-based secure keys are generated by a bare server, which is physically secured. The context is a previous conversation in text form saved at the chat server. Note that there is no online key distribution. It is not possible to achieve secure communication on the global Internet with billions of users.

In March 2025, inappropriate use of the Signal Messenger application resulted in a US Department of Defense security issue. Security issues with messaging are typically not due to weaknesses in the security mechanisms they use. For example, both Signal Messenger and WhatsApp messaging use the secure Signal protocol for end-to-end encryption. However, there are some security concerns with implementing end-to-end encryption and the XChat messaging application as discussed in [63].

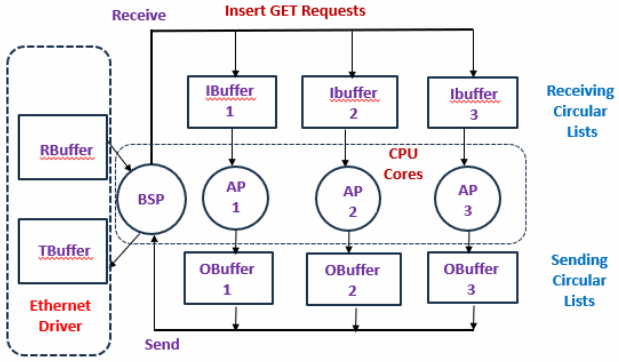

Fig. 4. Multicore organization in UDP-based Web server

Signal and WhatsApp are open source making their internals accessible to cryptographers (which is good) and attackers (which is bad). Furthermore, conventional messaging applications can be compromised by OS vulnerabilities. They also save chat messages on the cloud. In contrast, the bare chat application in [62] was architected, designed, and implemented to enable secure communication between bare users.

The significant contributions of this work include: demonstrating a chat application for bare users; proposing a bare Internet in which all nodes and ISPs are bare (a bare Internet is thus a subset of the Internet); and using a bare server-centric approach, where all client communication is tracked and controlled by the server including creation and control of physically distributed context-based keys.

3) Security Evaluation of the BMC Paradigm: A detailed evaluation of BMC systems focusing on their unique characteristics and security strengths is done in [64]. It shows that they can withstand twenty-two well known attacks on conventional systems. However, BMC systems may be still vulnerable to other types of attacks including physical security attacks and attacks by trusted bare users exploiting their assigned role and privileges. Conventional versus BMC guidelines and characteristics were also compared in this study. The twenty BMC guidelines include system approach, closed systems, local (not global) focus, user controlled secured USBs, limited access to authorized bare users, limited access to information on bare internals, no layers, bare Internet base, no heterogeneity, lower priority for user convenience, training for bare users, no online installation, no Wi-Fi (or other wireless connections), no script and batch files, no attachments, no social media access, no advertisements, no unsolicited Web sites, no automation tools, and no cookies. These guidelines are severely restrictive, but are necessary to significantly reduce the attack surface, ensure secure communication, and isolate the bare Internet from the global Internet used for other applications.

The significant contributions of this work include demonstrating the potential to build secure systems using BMC guidelines and characteristics as security requirements in system design; identifying principles for building inherently secure systems that are resilient to obsolescence; and showing

how to design computer and information systems that can spur novel architectures, designs, and protocols.

4) Miscellaneous BMC Systems: Many other BMC systems and applications were explored as referenced in [65-74], which are similar to applications described previously in this section.

After showing that the BMC paradigm could be used to design and build a variety of complex bare systems and applications as discussed above, research was conducted into transforming existing OS based applications to run on bare machines. First, the SQLite database engine was transformed to run on bare machines using a source code transformation in [52] as described earlier. In the binary level transformation technique, system calls are replaced with their bare equivalents in the direct hardware API (HAPI). This transformation technique does not require the programmer to understand the complexity of the binary code. Design issues in binary transformation, challenges, and progress made are described in [75].

The BMC paradigm is for designing and building general purpose computing applications running on all computing devices including smart phones. For example, a bare smart phone and a flash drive (USB device) could be used to boot and run personal domain-specific applications. This will reduce the number of smartphones needed since not everyone needs their own smartphone. This concept was explored to some extent in [76, 77, 78].

The significant contributions of this work include introducing novel bare systems that demonstrate use of the BMC paradigm as a general-purpose computing paradigm applicable to any computer or information system and noting that the BMC concept is relevant to smartphones and other pervasive devices.

VI. BMC Research and Development Contributions

BMC research and development contributions discussed in Section V illustrate the novelties and potential of the BMC paradigm. These contributions include (but are not limited to): architectures, protocols, and design methodologies; integrating a direct hardware API into the processor; a customized message control header to simplify design; making Internet nodes bear with mechanisms for more physical security; making obsolescence reduction a design requirement; defining a single programming language for computing; eliminating concurrency control mechanisms; isolating domain specific applications with limited users; and using proper abstractions and object-oriented principles.

This survey highlights the diversity and breadth of previous BMC research. It is intended as a source of information for preserving BMC technology, architecture, design, implementation, and systems approach that served as a foundation to build a variety of end user applications. We believe that the BMC paradigm is a means to reduce obsolescence and build highly secure systems, serving as an alternative to current evolutionary approaches that create more obsolescence and increase security vulnerabilities.

VII. Conclusion

We provided a comprehensive survey of BMC systems and applications, including the relevant literature, and research and development efforts. We gave a brief overview of the BMC paradigm and discussed how it differs from related approaches. While there is similar work, the BMC paradigm is unique in that it focuses on designing systems that reduce obsolescence and improve security. This survey represents our work primarily driven by twenty-seven doctoral dissertations and several graduate projects that were dedicated to building systems based on the BMC paradigm. During this work, many novel features were discovered such as architecture, designs, implementations, protocols, requirements, and guidelines that simplify computer and information systems. Overall, this survey is intended to preserve the knowledge and skills gained in building BMC systems and applications, and to provide a basis for further exploration of the BMC paradigm in the future.

References

[1] R. K. Karne, "Object-oriented Computer Architectures for New Generation of Applications," Computer Architecture News, December 1995, Vol. 23, No. 5, pp. 8-19.

[2] MisCircuitos. Difference between Bare Metal vs. Embedded Linux. miscircuitos.com/difference-between-bare-metal-vs-embedded-linux/, Accessed July 23, 2025.

[3] IBM. What is a bare metal server? ibm.com/topics/bare-metal-dedicated-servers, Accessed July 23, 2025.

[4] M. A. Mellal, "Obsolescence – A review of the literature", *Technol. Soc*. **2020**, *63*, 101347. [CrossRef]

[5] M. Zallio, D. Berry, "Design and Planned Obsolescence. Theories and Approaches for Designing Enabling Technologies" *Des. J.* 2017, *20*, S3749–S3761. [CrossRef]

[6] B. Dipert, Microsoft embraces obsolescence by design with Windows 11, EDN, Sept. 7, 2021.

[7] L. Bisschop, Y. Hendlin, and J. Jaspers, designed to break: planned obsolescence as corporate environmental crime, Crime, Law and Social Change 78, 2022.

[8] Song Ma, Technology Obsolescence, Nov 15, 2021, Last Revised Feb 1, 2025, papers.ssrn.com/sol3/papers.cfm?abstract_id=3964128, Accessed July 11, 2025.

[9] D. Naraharisetti, R. K. Karne, A. L. Wijesinha, J. Weymouth, In Proceedings of the 2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA), Orlando, USA, 23-25 May 2023; pp. 110–115.

[10] GreenMatters. Planned Obsolescence Exposed at Apple and Microsoft, in Light of New French Regulations. bbc.com/news/world-europe-42615378, Accessed March 27, 2024.

[11] Blake Van Jacobs, LiveGreen: Planned obsolescence, University of Nebraska, Medical Center, LiveGreen | August 26, 2020. unmc.edu/news.cfm?match=26100, Accessed July 11, 2025.

[12] Bill Gates Quote: "In three years, every product my company makes will be obsolete. The only question is whether we will make them obsolete or somebody else will."

[13] P. Hudomiet and R. J. Willis, Computerization, Obsolescence, and the Length of Working Life, Labour Econ. 2021 May 24;77:102005. pmc.ncbi.nlm.nih.gov/articles/PMC10079279/, Accessed July 11, 2025.

[14] D. R. Engler, The Exokernel Operating System Architecuture, Ph.D. thesis, MIT, 1998.

[15] I. Odun-Ayo et al, "An Overview of Microkernel Based Operating Systems", IOP Conference Series: Materials Science and Engineering, 1107 012052, 2021.

[16] TinyOS Home Page, tinyos.net/, Accessed Oct. 15, 2022.

[17] V. S. Pai, P. Druschel and W. Zwaenepoel, "IO-Lite: A Unified I/O Buffering and Caching System", ACM Transactions on Computer Systems, Volume 18, Issue 1, February 2000.

[18] The OSKit Project, cs.utah.edu/flux/oskit, Accessed Oct. 15, 2022.

[19] J. Lange, et. al, "Palacios and Kitten: new high performance operating systems for scalable virtualized and native supercomputing", 24th IEEE International Parallel and Distributed Processing Symposium, 2010.

[20] RIOT – The friendly Operating System for the Internet of Things, https://riot-os.org, Accessed Oct. 15, 2022.

[21] R. K. Karne, R. Gattu, R. Dandu, and Z. Zhang, "Application-oriented Object Architecture: Concepts and Approach", 26th Annual International Computer Software and Applications Conference, IASTED International Conference, Tsukuba, Japan, NPDPA 2002, October 2002.

[22] R. K. Karne, "Application-oriented Object Architecture: A Revolutionary Approach", 6th International Conference, HPC Asia 2002, December 2002.

[23] R. K. Karne, K.Venkatasamy, T. Ahmed, "How to run C++ applications on a bare PC", 6th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel / Distributed Computing (SNPD), Towson, Maryland, May 2005.

[24] R. K. Karne, K. Venkatasamy, T. Ahmed, "Dispersed Operating System Computing (DOSC)", Onward Track, OOPSLA 2005, San Diego, CA, October 2005.

[25] U. Okafor, R. Karne, A. Wijesinha, and P. Appiah-Kubi, "Eliminating the Operating System via the Bare Machine Computing Paradigm", 5th International Conference on Future Computational Technologies and Applications, Future Computing, May 27- June 1, Valencia, Spain, 2013.

[26] R. K. Karne, A. L. Wijesinha, and G. Ford, Opinion-- Stay on course with an evolution or choose a revolution in computing, ACM SIGARCH Computer Architecture News, Volume 36, Number 4, September 2008, pp. 1-6.

[27] F. Almansour, R. Karne, A. Wijesinha, R. Almajed, and H. Alabsi, "An Upward Compatible Ethernet Device Driver for Bare PC Applications", 11th International Conference on Information and Communication Systems (ICICS), 2020.

[28] F. Almansour, R. K. Karne, A. L. Wijesinha, and B. Rawal, "Ethernet Bonding on a Bare PC Web server with Dual NICs", The 33rd ACM/SIGAPP Symposium on Applied Computing, SAC2018, Pau, France, 2018.

[29] R. K. Karne, A. L. Wijesinha, and S. Liang, "A Bare PC Mass Storage USB Driver", International Journal of Computers and Their Applications, March 2013.

[30] G. Khaksari, A. L. Wijesinha, R. K. Karne, L. He, and S. Girumala, "A Peer-to-Peer Bare PC VoIP Application", IEEE Consumer Communications and Networking Conference, Seamless Consumer Connectivity, CCNC 2007, Los Vegas, Nevada, January 2007.

[31] A. Alexander, A. L. Wijesinha, and R. Karne. "Implementing a VOIP Server and a User Agent on a Bare PC", The Second International Conference on Future Computational Technologies and Applications, Future Computing 2010, November 21-26, Portugal, Lisbon.

[32] A. Alexander, A. L. Wijesinha, and R. Karne. "A Study of Bare PC SIP Server Performance", The Fifth International Conference on Systems and Networks Communications. ICSNC 2010, August 22-27, Nice, France.

[33] A. Alexander, R. Yasinovskyy, A. Wijesinha, and R. Karne, "SIP Server Implementation and Performance on a Bare PC", International Journal in Advances on Telecommunications, vol. 4, no. 1 and 2, 2011.

[34] A. Alexander, A. L. Wijesinha, and R. Karne. "An evaluation of Secure Real-Time Protocol (SRTP) Performance for VoIP", 3rd International Conference on Network and System Security (NSS), 2009.

[35] R. Yasinovskyy, A. Alexander, A. L. Wijesinha and R. K. Karne, "Bare PC SIP User Agent Implementation and Performance for Secure VoIP", International Journal on Advances in Telecommunications, vol 5 no 3 & 4, year 2012, pp. 119-119.

[36] R. Yasinovskyy, A. L. Wijesinha, R. K. Karne, and G. Khaksari, A Comparison of VoIP Performance on IPv6 and IPv4 Networks, The 7th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-2009), Rabat, Morocco. May 10-13, 2009.

[37] L. He, R. K. Karne, A. L. Wijesinha, and A. Emdadi, "Design and Performance of a Bare PC Web Server", International Journal of Computers and Their Applications (IJCA), June 2008.

[38] L. He, R. K. Karne, A. L.Wijesinha, and A. Emdadi, "A Study of Bare PC Web Server Performance for Workloads with Dynamic and Static Content", The 11th IEEE International Conference on High Performance Computing and Communications (HPCC-09), Seoul, Korea, June 2009, p494-499.

[39] H. Chang, R. K. Karne and A.L. Wijesinha, "Migrating a Bare PC Web server to Multi-core Architecture", IEEE 40th Annual Computer Software and Applications Conference, COMPSAC, Atlanta, Georgia, 2016, p216-221.

[40] B. Rawal, R. K. Karne, and A. L. Wijesinha. "Splitting HTTP Requests on Two Servers", The Third International Conference on Communication Systems and Networks: COMSNETS 2011, January 2011, Bangalore, India.

[41] B. S. Rawal, R. K. Karne, A. L. Wijesinha. "Split Protocol Client Server Architecture", Seventeenth IEEE Symposium on Computers and Communications (ISCC'12), July 1 -4, 2012, Cappadocia, Turkey.

[42] B. Rawal, R. K. Karne, and A. L. Wijesinha. "Mini Web Server Clusters for HTTP Request Splitting", 2011 IEEE International Conference on High Performance Computing and Communications, Banff, Canada, p94-100.

[43] B. Rawal, R. Karne, A. Wijesinha, P. Appiah-Kubi, and S. Liang, "Applications of the Split Protocol Paradigm, International Journal of Computers and Their Applications", Volume 21, No. 2, June 2014, p83-94.

[44] G. H. Ford, R. .Karne, A. L.Wijesinha, and P. Appiah-Kubi. "The Design and Implementation of a Bare PC Email Server", 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2009), Seattle, Washington, July 2009, p480-485.

[45] 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, 28-31 October 2009, São Paulo, SP, Brazil, pp. 143-150.

[46] P. Appiah-Kubi, R. K. Karne, and A. L. Wijesinha. "The Design and Performance of a Bare PC Webmail Server", The 12th IEEE International Conference on High Performance Computing and Communications, AHPCC 2010, Sept 1-3, 2010, Melbourne, Australia, pp 521-526.

[47] P. Appiah-Kubi, R. K. Karne, and A. L. Wijesinha. "A Performance Study of Conventional and Bare PC Webmail Servers", The Seventh International Conference on Networking and Services, ICNS 2011, p.280-285.

[48] P. Appiah-Kubi, R. K. Karne, and A. L. Wijesinha. "A Bare PC TLS Webmail Server", International Conference on Computing, Networking and Communications, ICNC 2012, Maui, Hawaii, January 2012, p.156-160.

[49] S. Almutairi, R. K. Karne and A.L. Wijesinha, "A Bare PC Text Based Browser", 2019 Workshop On Computing, Networking and Communications (ICNC), Honolulu, Hawaii, February 2019.

[50] S. Almutairi, R. K. Karne, A. L.Wijesinha, H. Chang, R. Almajed, H. Alabsi, W. Thompson, and N. Soundararajan "An API for Bare Machine Computing Applications", IEEE SoutheastCon, April 2019.

[51] W. Thompson, R. K. Karne, A. L. Wijesinha, H. Alabsi, and H. Chang, "Implementing a USB File System for Bare PC Applications", ICIW 2016: The Eleventh International Conference on Internet and Web Applications and Services, p58-63.

[52] U. Okafor, R. K. Karne, A. L. Wijesinha and B. Rawal, "Transforming SQLITE to Run on a Bare PC", In Proceedings of the 7th International Conference on Software Paradigm Trends, pp 311-314, Rome, Italy, July 2012.

[53] U. Okafor, R. K. Karne, A. L. Wijesinha, and P. Appiah-Kubi, "A Methodology to Transform an OS-based Application to a Bare Machine Application", The 12th IEEE International Conference on Ubiquitous Computing and Communications (IUCC-2013), July 16-18, Melbourne, Australia, 2013.

[54] W. V. Thompson, H. Alabsi, R. K. Karne, S. Liang, A. L. Wijesinha, R. Almajed, and H. Chang, "A Mass Storage System for Bare PC

Applications Using USBs", International Journal on Advances in Internet Technology, vol 9, no 3 and 4, year 2016. p63-74.

[55] W. Thompson, R. K. Karne and A. L. Wijesinha, "Interoperable SQLite for a Bare PC", 13th International Conference Beyond Database Architectures and Structures (BDAS'17), 2017, pp 177-188.

[56] H.Alabsi, R. K. Karne, A. Wijesinha, R. Almajed, B. Rawal, and F. Almansour, "A Novel SQLite-Based Bare PC Email Server", 15nth International Conference, BDAS 2019, Ustron, Poland, May 28-31, 2019, pp 341-p353.

[57] N. Soundararajan, R. K. Karne, A. L. Wijesinha, N. Ordouie, H. Chang, "Design Issues in Running a Web server on Bare PC Multi-Core Architecture", IEEE 44th Annual Computers, Software,and Applications Conference (COMPSAC), 2020

[58] N. Soundararajan, R. Karne, A. Wijesinha, N. Ordouie, B.S.Rawal, "A Novel Client/Server Protocol for Web-based Communication over UDP on a Bare Machine", 18th IEEE Student Conference on Research and Development (SCOReD), 2020.

[59] N. Ordouie, N. Soundararajan, R. Karne, A. Wijesinha, "Developing Computer Applications without any OS or Kernel in a Multi-core Architecture", IEEE ISNCC November 2021.

[60] N. Ordouie, R. K. Karne, A. Wijesinha, and N. Soundararajan, "A Simple UDP-Based Web Server on a Bare PC with 64-bit Multicore Processors: Design and Implementation", International Conference on Computing, Networking and Communicationssi (ICNC), 2023.

[61] F. Alotaibi, R. K. Karne, and A. L. Wijesinha, "A Stateless Bare PC Web Server", 19th International Conference on Web Information Systems and Technologies, WEBIST 2023, Rome, Italy, pp 406-413.

[62] F. Alotaibi, R. K. Karne, A. L.Wijesinha, and N.Soundararajan, "A Chat Application on a Bare Internet", 2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC), Osaka, Japan, pp 2411-2416.

[63] Matthew Green, A Bit More on Twitter/Xs New Encrypted Messaging, blog.cryptographyengineering.com/2025/06/09/a-bit-more-on-twitter-xs-new-encrypted-messaging/, Accessed July 9, 2025.

[64] F. Alotaibi, R. K. Karne, A. L. Wijesinha, N. Soundararajan, and A. Rangi, "An Evaluation of the Security of Bare Machine Computing (BMC) Systems against Cybersecurity Attacks", J. Cybersecur. Priv. 2024, 4(3), 678-730.

[65] N. Kazemi, A. L. Wijesinha, and R. K. Karne. "Evaluation of IPsec Overhead for VoIP using a Bare PC", 2nd International Conference on Computer Engineering and Technology (ICCET), 2010.

[66] N. Kazemi, A. L. Wijesinha, and R. Karne. "Design and Implementation of IPsec on a Bare PC", 2nd International Conference on Computer Science and its Applications (CSA), 2009.

[67] A. Emdadi, R. K. Karne, and A. L. Wijesinha. "Implementing the TLS Protocol on a Bare PC", ICCRD2010, The 2nd International Conference on Computer Research and Development, Kaula Lumpur, Malaysia, May 2010.

[68] R. G. Eyer, R. K. Karne, A.L. Wijesinha, "A Mechanism for Secure Delivery of Bare Machine Computing Software", 27th International Conference on Software Engineering and Data Engineering (SEDE 2018), October 2018, New Orleans, LA, USA, p134.

[69] A. Loukili, A. Tsetse, A. Wijesinha, R. Karne, and P. Appiah-Kubi, "Performance of an IPv6 Web Server under Congestion", 12th International Conference on Networks (ICN), 2013.

[70] A. Loukili, A. L. Wijesinha, R. K. Karne, and A. K. Tsetse, "TCP's Retransmission Timer and the Minimum RTO", 21st International Conference on Computer Communications and Networks (ICCCN), 2013.

[71] A. Tsetse, A. Wijesinha, R. Karne, A. Loukili and P. Appiah-Kubi, "An Experimental Evaluation of IP4-IPv6 IVI Translation", Applied Computing Review, March 2013, Vol. 13, No. 1, pages 19-27.

[72] A. K. Tsetse, A. L. Wijesinha, R. K. Karne and A. Loukili. "Measuring the IPv4-IPv6 IVI Translation Overhead", ACM Research in Applied Computation Symposium (RACS 2012)- Oct 23-26 2012, San Antonio Texas, USA.

[73] A. K. Tsetse, A. L. Wijesinha, R. K. Karne and A. Loukili. "A Bare PC NAT Box", International Conference on Communications and Information Technology (ICCIT 2012), June 26-28 2012, Hammamet, Tunisia.

[74] A. K. Tsetse, A. L. Wijesinha, R. K. Karne and A. Loukili. "A 6to4 Gateway with Co-located NAT", IEEE International Conference on Electro Information Technology (EIT 2012), May 6-8 2012 Indianapolis, USA.

[75] R. Almajed, R. K. Karne and A.L. Wijesinha, "Binary Transformation of Applications to run on Bare PCs", The 34th ACM/SIGAPP Symposium On Applied Computing, April 8-12, 2019, Limassol, Cyprus

[76] A. Peter, R. K. Karne, A. L.Wijesinha, and P. Appiah-Kubi, "Transforming a Bare PC Application to Run on an ARM Device", IEEE SoutheastCon, April 4-7, Jacksonville, FL, 2013.

[77] A. Peter, R. K. Karne, A. L. Wijesinha, and P. Appiah-Kubi. "The Design and Implementation of Bare PC Graphics", The Seventh International Multi-Conference on Computing in the Global Information Technology (ICCGI), Venice, Italy-June 24-29, 2012. (Best Paper Award).

[78] A. Peter, R. K. Karne, and A. L.Wijesinha, "A Bare Machine Sensor Application for an ARM Processor", IEEE International Electro-Information Technology Conference (EIT), May 9-11, Rapid City, South Dakota, 2013.