# Lightweight TCP/IP Architecture model for embedded systems using SysML

AHMED E. HASSAN

Electrical engineering, faculty of engineering,

dr.hassan@ahmed-hassan.org

M. Z. Rashed

Magdi_12003@yahoo.com

Computer science dept, faculty of computer science

AHMED I. SHARAF

Computer science dept, faculty of computer science

Ahmed.sharaf.84@gmail.com

Abstract:

Embedded systems are usually suffering from limited resources which require a modified architecture of software to take the best usage of the resources. TCP/IP is the most important communication protocol for networked embedded systems which provide internet connectivity for hosts. Many of TCP/IP development for embedded systems lack system modeling, analysis and design.

This paper presents a proposed lightweight TCP/IP architecture which consists of TCP/IP kernel, memory management, multi-threading and socket support to fit limited hardware resources devices. SysML is a general purpose modeling language is used to model the lightweight TCP/IP.

The proposed SysML model discusses full functional analysis, design and requirement of the proposed architecture.

***Keywords: Communication protocol, embedded systems, TCP/IP, SysML, system engineering and modeling languages.***

## 1. Introduction:

Embedded systems are computers which are part of special purpose devices [26]. These systems vary in size, scope of use and complexity, also these systems reside nearly in most devices. Embedded systems are usually resource limited in terms of processing power, memory, and power consumption [9].

A networked embedded system is a collection of spatially and functionally distributed embedded nodes, which are interconnected by means of wired or wireless communication infrastructure and communication protocol. There have been various reasons for emergence of networked embedded systems, influenced largely by their domain applications. The benefits of using distributed systems and evolutionary need to replace point to point connection in these systems by a single bus are some of most important ones.

Applying the power of Internet and communication protocols can add many new functionally to embedded systems [15].

Implementing protocol software is based on detailed specification. These specifications are usually specified by a standards body such as the International Organization for Standardization (ISO), Institution of Electrical and Electronic Engineers (IEEE), or International Telecommunications Union (ITU-T). Example protocols include the Internet Protocol (IP) [19] and IEEE 802.2 Logical Link Control (LLC). Using the OSI [5] layering as an abstraction mechanism, the software architecture of a

complex communications protocol can be partitioned into higher and lower layers. For some functions, the higher and lower layers may be other protocols. For example, a device driver may be a lower layer function, while an application may be a higher layer implementation. The analysis and design of that communication protocol is hidden in implementation.

With the success of the Internet, the TCP/IP [18] protocol suite has become a global standard for communication. TCP/IP is the underlying protocol used for web page transfers, e-mail transmissions, file transfers, and peer to-peer networking over the Internet. TCP/IP Inherits its nature as both a sophisticated software and a communication protocol that make it difficult to fit embedded systems. There is no previous implementation of TCP/IP which follows the software life cycle.

TCP/IP modeling requires requirement analysis, design, test and verification phases. Embedded systems don't depend on operating systems in many cases, therefore TCP/IP could be modeled as separate module.

This separate could module handles internal memory management, multi-threading, socket support and time management. Modeling these components requires a general purpose modeling language that can fit embedded systems such as systems modeling language (SysML) [7] and [16]. Authors of that work propose a TCP/IP model requirement, design and analysis using SysML.
The Unified Modeling Language (UML) [27] is an object oriented modeling language, which cannot fit all phases of the proposed TCP/IP model. SysML can be used for specifying, analyzing, designing and verifying systems that may include hardware, software, information, personnel, procedures and facilities. This modeling language could be integrated with any other engineering analysis model through a graphical and semantic foundation for modeling system requirement, behavior, structure and parametric. SysML has the following advantages versus UML [17]:

1. SysML's semantics are more flexible and expressive. SysML reduces UML's software-centric restrictions and adds two new diagram types, requirement and parametric diagrams.
2. SysML is a smaller language that is easier to learn and apply. Since SysML removes many of UML's software-centric constructs, the overall language is smaller as measured both in diagram types and total constructs

Authors of that work proposed a lightweight TCP/IP architecture and its model using SysML. It has proved itself as a lead modeling language, Robert Karban et all used SysML in model based system engineering in Telescope modeling [23].

## 2. Related Work:

Communication protocol design process is a special type of software design processes which requires a fast runtime execution and a flexible modeling technique [25]. The methodologies used for the development of protocols can be grouped in three categories [13] :

(1) The procedural approach which is defined as applying the structured design methodology and using C programming language for protocol development. Even though this approach results in efficient implementation, the maintenance and reusability are rather poor.

(2) The second category is defined as using a formal description technique to create the protocol's specifications. The formal description is then translated into program code. SDL [11], Estell[10] and Lotos[14] are examples of specifications used , with SDL being the most widely adopted. There is no reusability due to the missed design phase.

(3) The last category which is continuously gaining ground is defined as using methodologies that are based on the object oriented (OO) paradigm. The object oriented approach results in implementation that exhibit increased modularity, flexibility, extensibility and reusability. Danny Patel et all [6] used this approach to represent an OO TCP/IP for RTLinux[8].

But using this technique could add a middleware layer which will reduce the memory space and slow down the system especially in limited memory software where memory is in bytes or a few KBs.

Adams Dunkels presented two proposal based on the first approach which are called UIP [2] and IwIP[1] . His proposals are based on redesigning TCP/IP as lightweight separate software that is targeting tiny 8 bit microcontrollers [3]. But there are three weak points in Dunkels's approach.

First, there is no software model. Since communication protocols are complex software, modeling is necessary process. Software modeling and analysis can improve system maintenance, flexibility, extensibility and ease of system understanding.
Second, both UIP and IwIP are targeting tiny embedded systems that make it difficult to use this approach for anther microcontroller architecture. Lastly, there is no modularity which makes it hard to customize the functionality of system.
Authors of that work using a hybrid technique which is based on the first development approach and SysML as modeling language. The proposed model inherits efficient implementation, system flexibility, extensibility and system understanding. The proposed lightweight TCP/IP model uses SysML for modeling, analysis and design.

## 3. Proposed model:

This section contains model architecture section and requirement diagram section. The model architecture represents the proposed architecture model and its components. The requirement diagram represents a visual representation of system requirement.

### 3.1. Model architecture

The proposed lightweight architecture is a layered model as shown in Figure 1. This architecture is designed to be a generic that is not depending on a specific operating system or specific platform. The proposed architecture consists of mainly 7, the focus of that paper is on TCP/IP core layer. The proposed layers are briefly explained as follows:

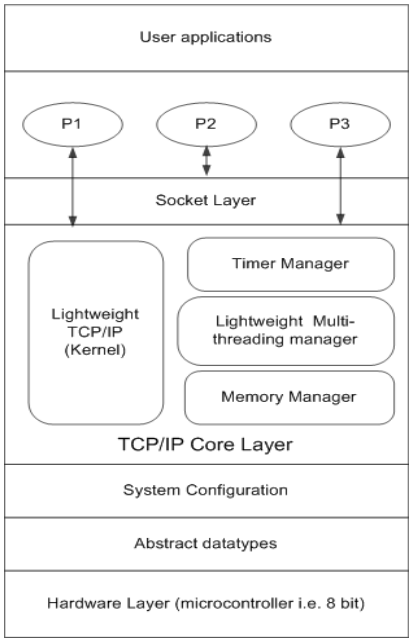


Figure 1 TCP/IP proposed architecture

- *Hardware layer :*

This layer handles the hardware details of the model, its machine dependent layer.

- *Abstract datatypes :*

This layer handle the different representation of datatypes, it is also machine dependent layer.

- *System configuration :*

This layer handles the system global variables and system constants.

- *TCP/IP core :*

This layer is core of the system, it handles the lightweight TCP/IP (kernel), time management, multi-threading and memory management. This layer will be most important layer in the model.

- *Socket Layer :*

This layer handles the socket support and how the systems deals with socket

- *Protocol process :*

This layer handles the communication protocols as software processes. This layer is high configurable layer.

- *User application:*

This layer handles the application which will depend on the communication protocol infrastructure.
The details of the proposed architecture will be discussed in more details in the next section.

The primary step of the proposed model is to determine the requirement of the architecture based on embedded systems environment and limitations.
The proposed model requirement diagrams [24] is shown in Figure 2, which describes the requirement of the model in visualized diagram instead of text based requirement.

### 3.2. Requirement diagram:

The system requirements are classified into three categories:

- System requirement
- User requirement
- Developer requirement.

The proposed system requirement is classified into two categories both hardware and software

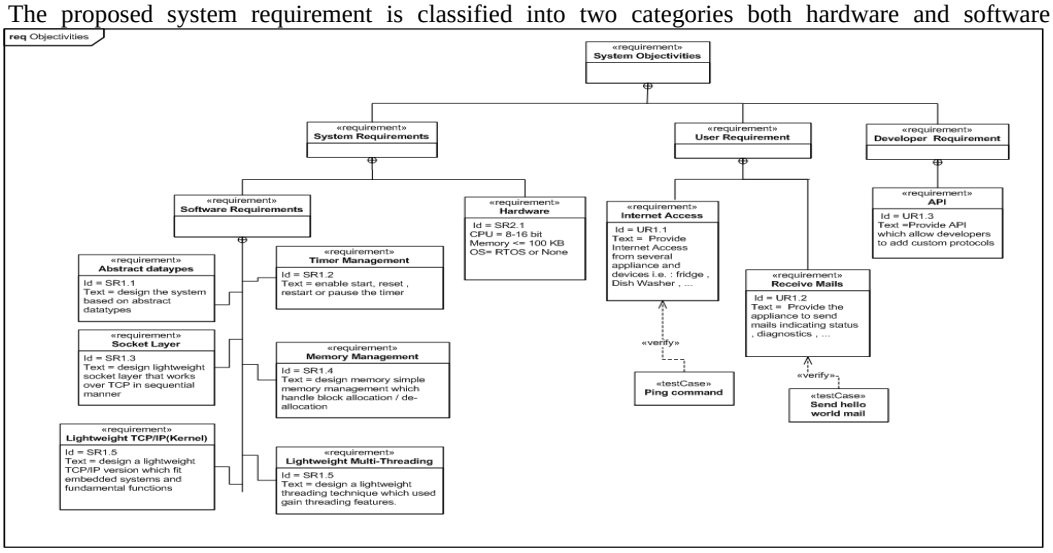



Figure 2 Requirement diagram of the proposed model

requirements. The hardware requirement specifies the embedded systems hardware features such as 8-bit microprocessor, 100 KB of memory or less and the type of the operating system whatever real time operating system (RTOS) or no operating system at all. The software requirement which the proposed model will contains such as Abstract datatypes, time management, lightweight socket support, memory management, lightweight TCP/IP and lightweight multi-threading technique.
The user requirement category specifies what applications could use the proposed system from user prospective view. Embedded systems and appliance could access the internet through a cheap and fast way, theses systems also could send e-mails regarding diagnostics or machine status.

The third category or the developer requirement specifies how it is easy and not complex for developers using the proposed system to develop any application using TCP/IP API and cheap microcontroller. The proposed system also hides the details of complex communication protocols with a good level of extensibility through encapsulating the lightweight TCP/IP component or just the kernel. At that phase it is clear for developers how to add other features such as generic memory management, timer management and threading management that does not depend on specific operating system.

## 4. Proposed Analysis and Design

In this section the detailed analysis and design model will be discussed with more details. The focus will be the lightweight TCP/IP core layer which is the kernel of the proposed architecture.

### 4.1. Hardware Layer:

This layer represents the details of microcontroller architecture and features. The proposed architecture is dedicated to 8 bit microcontrollers which support standard C compiler. The interfacing and hardware details are not included in this paper.

### 4.2. Abstract datatypes:

This layer contains generic data types which are used in the whole architecture, this layer is machine dependent layer that depends on the hardware architecture. The developer can add or edit the data types to match specific hardware.

### 4.3. System Configurations:

This layer represents system configuration and system options. The configuration layer handles global variables such as IP address, MAC address, and buffer size. It can be also coded by a developer, adding or hard coded system global variables which are allowed.

### 4.4. TCP/IP core layer:

This layer consists of lightweight TCP/IP, timer manager, multi-threading manager and memory manager. The block diagram and use case of every component and the whole layer is explained as follows:

- *Lightweight TCP/IP(kernel) :*

This layer is the most important layer in the proposed architecture and it also called the kernel layer. This layer encapsulate the core of TCP/IP, it includes both IP and TCP [20] protocols. The details of this layer are hidden from developer, which hides the complexity of the system inside the kernel layer. This layer is not editable by the user. Figure 4 illustrate the relation between the kernel layer and the other components such as datatypes, configuration and the architecture layers. Implementing any protocol process need that kernel as a root module to inherit the functionality of TCP/IP.

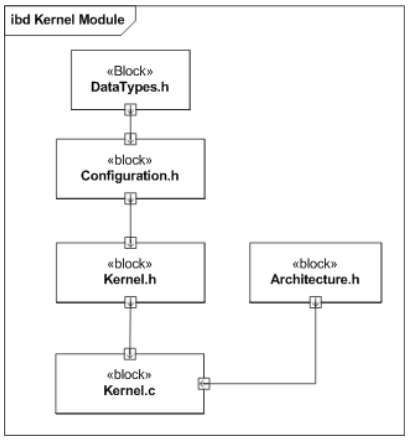


Figure 3 internal block diagram of Lightweight TCP/IP

- *Timer manager :*

One critical point in any communication protocol process is time management, most protocol process depend on time for many cases such as connection time out, resend data and stop sending. Therefore time management component in TCP/IP is important. Calculating the intervals is determined by the hardware clock which is wrapped into *"clock.h"* module. The internal block diagram is shown in Figure 4.Time manager should provide basic operations such as set time interval restart the timer and reset the timer with the last configured interval value, these operating are shown in Figure 6 which represents the use case of Time manager module.

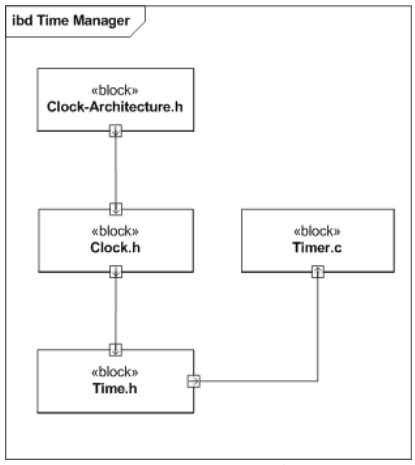


Figure 4 internal block diagram of Time manager

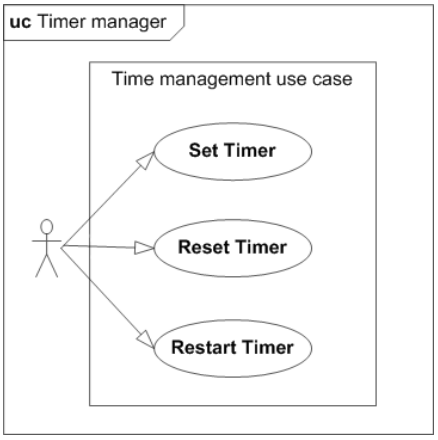
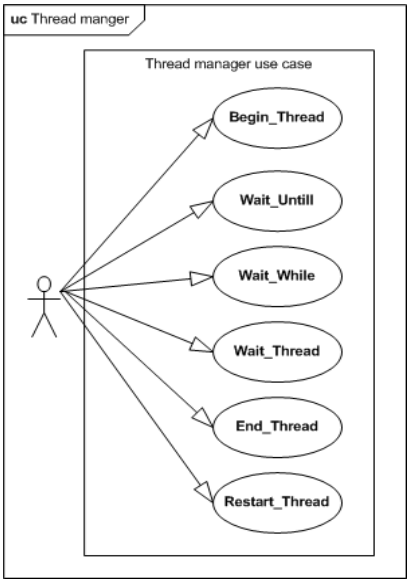


Figure5 use case diagram of Time manager

- *Multi-threading manager :*

Multi-threading techniques are very useful methods used general development, embedded systems also need the same technique but with special requirements. Embedded systems requires more lightweight software that can be managed and executed in limited processing power such as 8-16 bit microcontroller. The proposed architecture represents a lightweight multi-threading technique which based on Protothread [4].

The proposed multi-threading technique is stack less thread which provides linear code execution for event driven systems. One advantage of these threads is there is no need to implement thread per stack as ordinary thread. In memory constrained systems, the overhead of allocating multiple stacks can consume large amounts of the available memory. The multi-threading use case diagram is shown in Figure 6, which illustrate the proposed functionality.




- *Memory manager :*

In embedded systems the most scary resource is memory. Figure 7 show the use case of memory manager functions and how to deal with memory block, the most important functions are: "*Init_Block"* which represents the declaration of memory block to be handled, "*Block_alloc"* which represents the allocation of memory that already declared and "*Block_free"* which represents the memory de-allocation of declared block. The proposed behavior for the memory manager is to use single buffer for holding packets , when a packet is arrived the device driver place it in the global buffer and call the kernel modules. If the packet contains data, the kernel notifies the corresponding application. When the application receives a notify message it have to take one action from the following:

- Perform online processing on global buffer
- Copy the packet contents to secondary buffer and perform the processing on it.

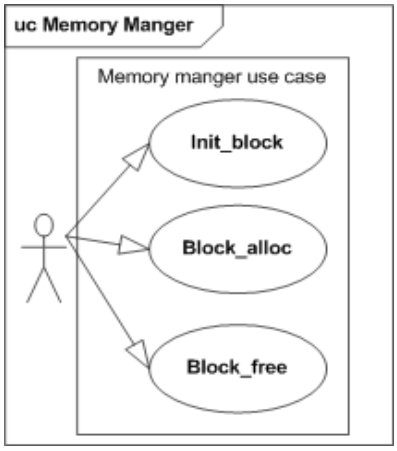



Figure 7 memory management use case

Figure 8 shows the proposed activity diagram which illustrate the workflow in memory manager .One other view is also important in memory management is the functionality of the proposed architecture which demonstrate what will the module do with memory blocks.

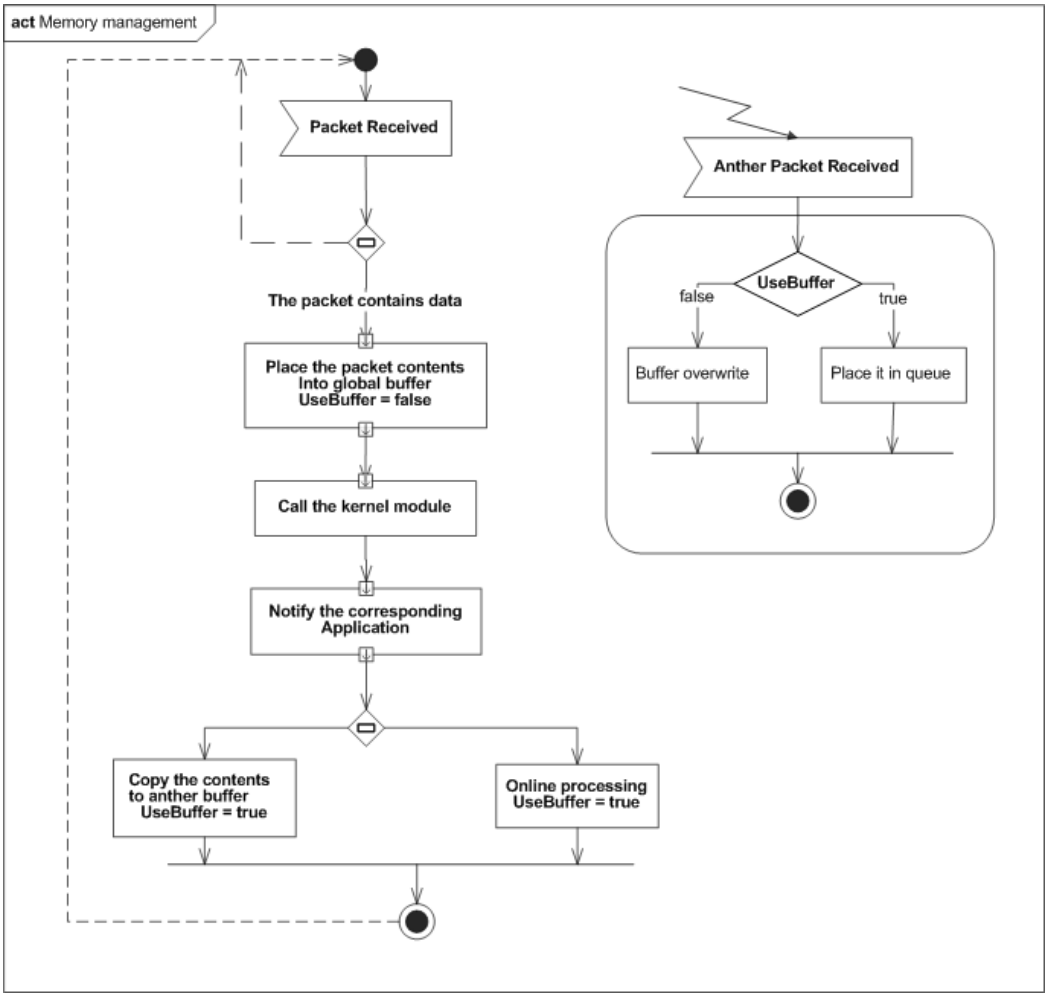


Figure 8 memory management activity diagram

### 4.5. Socket manager:

Socket [12] is an end point of a bidirectional communication link between hosts or between processes on the same host. Socket component is based heavily on the thread mechanism that make socket inherits the complexity of thread, the socket being by invoking "*Begin_thread"* and terminated by "*End_thread".* Figure 9 shows the internal block diagram of socket manager. Figure 10 shows the corresponding use case which illustrate the main functionality of socket management component.

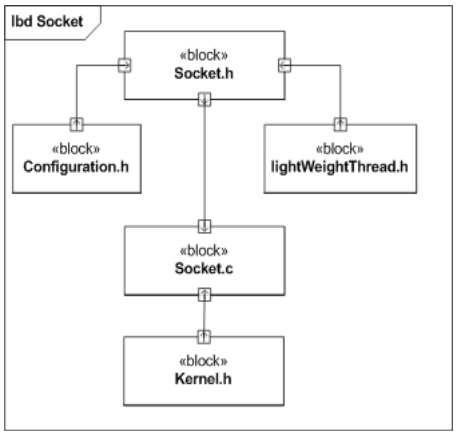



Figure 9 internal block diagram of socket manager

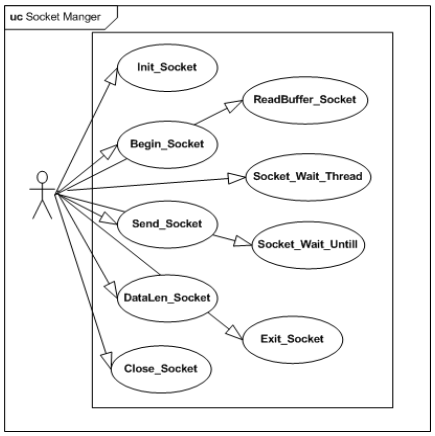



Figure 10 socket management use case

### 4.6. Protocol process:

In this section a communication protocols is illustrated as example to describe how to model the protocol processes using that proposed model. The selected protocol is OSI protocol. It's used to show how the proposed model can represents a flexible and reusable model in field of embedded communication protocols

- *Address Resolution protocol (ARP) :*

Address resolution protocol [22] is a simple protocol process that map IP address to its corresponding physical (MAC) address. This process requires accessing the kernel component to handle IP address, MAC address and messaging technique. Figure 11 shows the internal block diagram of ARP based on the proposed architecture.

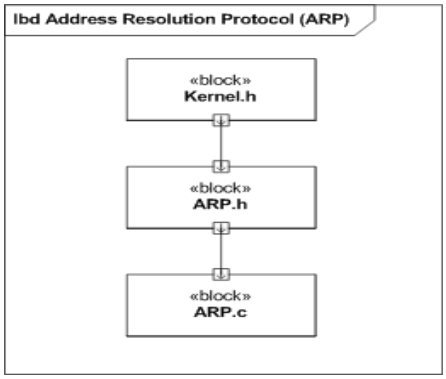



Figure 11 ARP internal block diagram

- *Simple Mail Transfer Protocol (SMTP):*

Simple mail transfer protocol [21] is more advanced protocol than ARP that requires kernel component, socket component and system configuration component. Figure 12 shows the internal block diagram of SMTP.

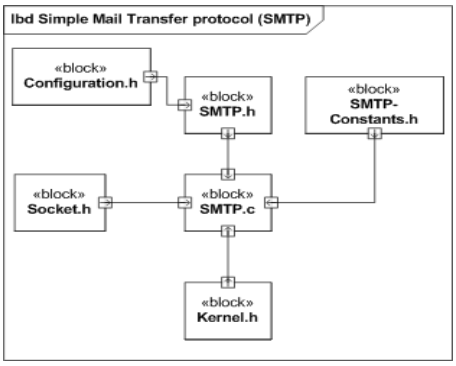



Figure 12 SMTP internal block diagram

### 4.7. User Applications:

User application layer define the developed application which based on the proposed architecture, this layer is not discussed in this paper.

### 5-Conclusion:

In this paper, a lightweight TCP/IP architecture is presented for embedded systems. The proposed architecture consists of TCP/IP core, memory management, time management and multi-threading .It also offers new methodology and an API for embedded system which enable developers to add web features to

their systems through handling TCP/IP stack. The proposed architecture is modeled using SysML which is a general purpose modeling language and relatively new.
SysML offers new techniques and diagrams to facilitate the process of software modeling in embedded systems and communication protocol design. The purpose of the proposed architecture is to handle TCP/IP stack with reusability and ease of use as standard software for special environment.

**References**

[1] A.Dunkels. lwIP-a lightweight TCP/IP stack. Webpage.2002, URL: http://www.sics.se/˜adam/lwip/

[2] A.Dunkels. uIP a TCP/IP stack for 8-and16-bit microcontrollers.Webpage.2002. URL: http://dunkels.com/adam/uip/

[3] Adam Dunkels , Full TCP/IP for 8-Bit Architectures

[4] Adam Dunkels et all , Protothreads: Simplifying Event-Driven Programming Memory-Constrained Embedded Systems.

[5] Cassel, Lillian N , Computer Networks and Open Systems : An Application Development Perspective , Jones & Bartlett Publishers, Inc. , (2000)

[6] Danny Patel , Object oriented design of an embedded communication protocol in UML.

[7] Dennis M.Buede , The Engineering Design of systems , models and methods.John Wiley & Sons 2009

[8] Doug Abbott , Linux for Embedded and Real-Time Applications , 2nd ed. , Elsevier 2006

[9] Edward Insam, TCP/IP Embedded Internet Applications , Newnes 2003.

[10] Estell , ISO international standard IS8807 , July 1989.

[11] Jan Ellsberger et al , SDL:Formal Object oriented language for communicating systems. Prentice Hall 97.

[12] Jeremy Bentham , TCP/IP Lean Web Servers for Embedded Systems, CMP books 2002.
[13] K.Thramboulidis , A.Mikroyannidis . Using UML for the design of communication protocols : the TCP case study ,2003.

[14] Lotos , ISO international standard IS8807 , Feb 1989.

[15] Miroslav Popovic , Communication protocol engineering , CRC press 2006.

[16] Omg , Systems Modeling Language (SysML) Specification , 2005

[17] Omg, http://www.sysmlforum.com/FAQ.htm , june 2010

[18] Pete Loshin , TCP/IP Clearly explained 4th ed. Elsevier 2003

[19] RFC791 Internet Protocol http://www.faqs.org/rfcs/rfc791.html

[20] RFC793 Transmission Control Protcol http://www.faqs.org/rfcs/rfc793.html

[21] RFC821 Simple Mail Transfer Protocol , http://www.faqs.org/rfcs/rfc821.html

[22] RFC826 Ethernet Address Resolution Protocol: Or Converting Ne http://www.faqs.org/rfcs/rfc826.html.

[23] Robert Karban , Rudlof Hauber and Tim Weilkiens . MBSE in Telescope modeling.

[24] Sanford Friedenthal , Alan Moore and Rick Steiner , A Practical Guide to SysML the Systems Modeling Language , Elsevier 2008.

[25] T. Sridhar , Designing Embedded Communications Software , CMP Books 2003

[26] Tammy Noergaard , Embedded Systems Architecture A Comprehensive Guide for Engineers and Programmers , Elsevier 2005.

[27] Tim Weilkiens , Systems Engineering with SysML/UML Modeling, Analysis, Design , Elsevier 2006