

# Olsonet eCOG1 Experimental Platform for Ad-hoc Wireless Networking

**Document type:** Technical  
Documentation

**Version:** 1.0 (Issued)

## ABSTRACT

In this document, we introduce the Olsonet platform for developing low-footprint, customizable, ad-hoc, wireless networks. It consists of a small operating system (PicOS), a layer-less networking module for PicOS (TCV), and a generic implementation of the Olsonet proprietary routing protocol TARP. Together, the three components provide an infrastructure that enables rapid development of wireless applications running on eCOG-based cards.

## 1. Introduction

Our project addresses a subclass of wireless devices with the following characteristics:

- Small physical dimensions (typically credit-card size)
- Small footprint in terms of memory and peripheral equipment
- Extremely low cost
- Long durability (low battery consumption)

Intentionally, such devices will be deployed in selected domains of Personal Wireless Communication – those that warrant cheap specialized “cards” rather than general-purpose more expensive devices. See [4] for a discussion of such applications.

Some of the targeted application domains will assume that the multiple cards present in the neighborhood form an ad-hoc network, whereby a session involving a subset of cards is forwarded via intermediate cards, which do not directly participate in the session, but are nonetheless needed to maintain connectivity. Some other applications may involve proximity-triggered sessions, which do not require forwarding. Our generic platform strives to cater to all those application types (and all conceivable transaction types) by providing an open-ended set of tools facilitating rapid development of customized embedded systems, with the networking component (communication protocols) considered their important and integral part.

## 2. PicOS

The programming paradigm of PicOS [5] comes from SMURPH [1] (also called SIDE [3] in its most recent

version), which was devised as a specification tool for expressing the behavior of reactive systems. PicOS offers a flavor of multitasking with finely-grained processes implemented as semi-preemptible coroutines with multiple entry points. This novel and unorthodox approach has a number of advantages, especially from the viewpoint of our targeted class of applications. First, by eliminating the traditional requirement that every process be equipped with private stack space, PicOS is able to implement multiprogramming within trivially small memory. Second, by constraining process preemption to natural and consistent checkpoints, PicOS renders most synchronization problems trivial. This simplifies the organization of the system itself and greatly reduces the complexity of multithreaded applications. Finally, by enforcing a reactive view of its applications (as *finite state machines* responding to events), PicOS makes them self-documenting and easily amenable to formal ways of arguing about their semantics and behavior. In essence, applications in PicOS are structured in a manner similar to StateCharts [2].

### 2.1 Processes

By a popular tradition, tasks in PicOS are called processes. A process is described by its code (resembling a function with multiple entry points) and data (representing the object on which the process is supposed to operate). The multiple entry points of a process function are called states, and the function itself can be viewed as a specification of a Finite State Machine (FSM).

The transition function of the process’s FSM is defined by associating states with events to which the process would like to respond. A typical sequence of statements executed by a process in a given entry/state (see Figure 1), includes at least one wait operation indicating which future events the process wants to perceive and what the process wants to do when the corresponding event occurs. Some wait operations are implicit, i.e., invoked internally by other functions (e.g., I/O and IPC tools).

A process can only be preempted at a state boundary when it relinquishes the CPU. While this approach may seem somewhat unfriendly from the viewpoint of real-time operation, one should remember that it only concerns

processes (i.e., application threads), not interrupts within the kernel. PicOS is able to respond to “physical” events in real time, if this kind of response is truly needed.<sup>1</sup>

```

process (name, datatype)
  entry statename:
    ...
  entry statename:
    ...
    wait (event, state);
    ...
    release;
    ...
endprocess (ncopies);

```

**Figure 1. The structure of a process declaration in PicOS**

A process may exist in a number of copies, which can be limited (by the argument of `endprocess`). Processes can be dynamically created and killed. PicOS offers an elaborate collection of IPC tools, which facilitate the development of complex multithreaded applications.

## 2.2 Device drivers

PicOS provides interface to most of the hardware features available on the eCOG evaluation board. Specifically, the following components are visible as devices with their associated drivers:

- `DUART_A` and `DUART_B`. Each DUART can operate in one of two modes with independently selectable baud rate.
- The LCD display.
- The four LEDs (a trivial output device).
- The SMSC 91C111 Ethernet chip<sup>2</sup>. Two modes of operation are available, with the “cooked” mode implementing simple structured frames for the purpose of emulating radio interfaces over local wired networks. The “raw” mode offers the full (bare) functionality of Ethernet.

Moreover, PicOS offers functions for accessing the A/D converter and the on-board speaker (as a selectable-tone buzzer). In addition to the drivers built into the kernel, PicOS library offers many useful functions for I/O formatting and presentation.

<sup>1</sup> Many people incorrectly assume that all embedded applications are inherently real-time constrained, whereas in fact very few of them really are.

<sup>2</sup> RF interface from RFMI, Xemics, and RFWaves to radio modules is not a part of the GPL release.

## 2.3 Memory

PicOS can be configured to run exclusively within the on-chip 2K words of RAM, or to take advantage of the SDRAM available on the evaluation board. In the latter case, the first 32K words of SDRAM are mapped into the data space and made directly available to the application, while the remaining SDRAM area is accessible as a backup storage (that can be referenced via two system functions). In both cases, the system provides operations for dynamic memory allocation (`malloc/free`), with the possibility of setting up independent non-interfering partitions.

## 2.4 System configuration

The present prototype version of PicOS comes with a few sample applications whose purpose is to test the system and illustrate its principles. These applications include:

**Toy** illustrating the operation of the DUARTs, LCD, and LEDs, as well as providing tests for many system functions.

**Prater** illustrating the operation of the two UARTs, also when PicOS is run under the simulator.

**MacTest**, which collaborates with a server on a UNIX machine to demonstrate the operation of the Ethernet driver.

**Sniffer**, which receives raw frames on the Ethernet interface, buffers them, and dumps their contents to the selected DUART. This application illustrates the cooperation of the Ethernet driver with basic TCV (see Section 3).

**RemoteDisplay**, which listens for a message from a server (running on UNIX machine connected to the same Ethernet) and displays its contents on the LCD.

**SerTest**, which (in collaboration with a server on a UNIX machine) demonstrates emulation of a wireless channel over a serial port.

Each application resides in a separate directory that, in addition to the source program, includes a configuration file selecting the options of PicOS needed to support the application. The system is built by executing `make` in the application directory.

## 3. TCV

The traditional layered approach to implementing networking software has been recently subjected to serious criticism for its inherent inefficiency and lack of flexibility, which are especially visible in the wireless environment. To facilitate rapid development of diverse and efficient wireless protocols, our platform advocates a layer-less paradigm of organizing protocol software. To this end, TCV is a plugin-extensible, open-ended, generic transceiver module for PicOS. The purpose of this module is to provide a simple collection of API, independent of the underlying implementation of networking, which, in

addition to enabling a rapid deployment of networked applications for eCOG-driven cards would make it easy to develop testbeds using emulated radio interfaces.

From the viewpoint of the application, TCV offers a semi-complete generic functionality, which can be redefined by plugins. Also, the actual implementation of the physical interface to the network can be specified as a relatively simple and easily exchangeable module. To facilitate development, testing, and experiments, multiple plugins and physical interfaces can coexist within the same system configuration.

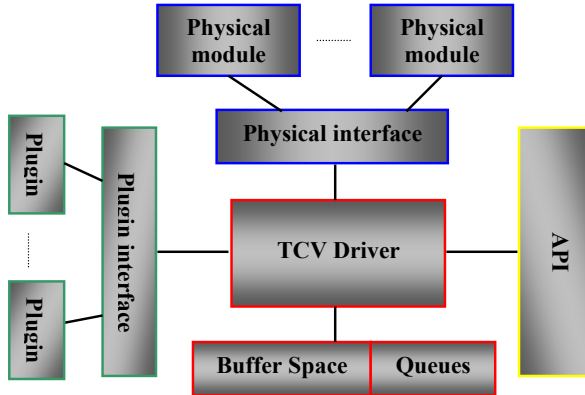


Figure 2. TCV structure

For example, within the framework of the evaluation board, the transceiver module can be associated with the Ethernet interface in a way that emulates a radio channel possibly shared by multiple boards interconnected via a local network. Such a farm of evaluation boards, additionally equipped with an emulation server (e.g., running on a Linux workstation), can be used as a powerful testbed for developing wireless applications immediately deployable on eCOG-controlled hardware. It is also possible to use the Ethernet device as a straightforward network connection. In combination with TCV, which turns this connection into a plugin-controlled flexible networking device, this option makes it possible to build embedded networked applications operating directly on the evaluation board. In particular, the TCP/IP stack or TARP (Section 4) can be implemented as plugins to the TCV module.

The structure of TCV and its relationship with other system components is shown in Figure 2. In essence, the module implements transparent management of buffer (packet) storage organized into a dynamic number of queues, timeouts definable on a per-packet basis, multiple application access points (roughly equivalent to connections or sessions), and provides a unified set of functions for interfacing plugins and physical modules.

The API of TCV can be viewed as a dynamic collection of abstract devices representing network connections or sessions. A workable TCV setup involves at least one

physical module and at least one plugin. The application can open a connection, receive and transmit packets over a connection, close a connection, without having to worry about details, like allocation/deallocation of buffer space, header/trailer processing, and possibly error recovery, which operations are performed transparently by the plugin(s). A single plugin can cooperate with multiple physical modules, multiple plugins can also “claim” frames delivered by different physical modules for their specific processing.

### 3.1 Physical interface

A physical module registers itself with TCV by providing its Id and a control function for affecting its operational parameters. A registered physical module is assigned a queue of outgoing packets. This queue is handled automatically by TCV and uniquely identified by the Id of the physical module. The physical module acquires packets for transmission by calling a function provided by TCV and, possibly, by responding to some events, e.g., triggered when the output queue becomes nonempty.

The reception end is handled in a similar manner. When the physical module receives a packet that should be submitted to the system, it calls another function provided by TCV identifying itself and passing it the packet.

### 3.2 Application interface

Before the application can use the services offered by TCV (in collaboration with its registered plugins and physical modules), it must open a session. The exact semantics of what it means to “open a session” depends on the plugin. In the simplest case, there can be a single session per physical interface representing a more or less direct connection to the network. In a more sophisticated scenario, opening a session may involve setting up a TCP connection across the network, with multiple connections coexisting in time. A session is identified by a descriptor, which has the appearance of a logical input/output device.

The operations of reading from and writing to the session descriptor deal with packets, while making it possible to extract/send information in smaller chunks. This simple way, without introducing too many options and modes, the structure of packets is made visible enough to account for those circumstances when the application prefers to see them explicitly, and transparent enough, to view a session descriptor as a stream-like device handling (almost) unpackaged sequences of bytes.

### 3.3 Plugin interface

A TCV plugin is completely described by a small collection of functions that are called automatically by TCV, in prescribed circumstances, mostly to solicit the plugin’s decision regarding the fate of a packet. For example, when an incoming packet is passed to TCV by a physical module, each plugin is given a chance to claim the

packet. To this end, every plugin must define a *receive* function whose primary responsibility is to tell TCV whether the plugin cares about the packet. The simple idea is that packets unclaimed by any plugin are dropped, while a claimed packet is associated with the first claimant and stored for further processing.

Generally, a plugin function is expected to return a value telling TCV what should happen to the packet next. The most standard options are dropping the packet, queuing the packet for transmission, and queuing it for reception at the assigned session.

#### 4. TARP

The Tiny Ad-hoc Routing Protocol (TARP) is a simple and robust forwarding scheme [7] suitable for building ad-hoc networks catering to many diverse applications. The protocol is based on the general idea of “controlled flooding,” requires minimal memory space, and assures delivery of packets, even when topology changes are frequent. TARP also provides application designers with adjustable modes of operation allowing them to select the desired level of protocol performance and complexity.

Using some minimal information passed in packet headers, TARP analyzes the route of a packet to be forwarded and determines whether the packet should be in fact retransmitted or dropped as superfluous. While the accuracy of this decision in the wireless ad-hoc environment is bound to be, TARP bases it on intelligent feedback and allows for a certain (parameterized) degree of fuzziness trading accuracy for redundancy. In particular, in stable configurations, the protocol is able to quickly and automatically uncover optimal routes while retaining some token redundancy, to be able to promptly respond to a changed network topology. Based on the anticipated properties of the target deployment (mobility, density), the application designer can pre-configure some of those parameters, suggesting to TARP the preferred mode of operation.

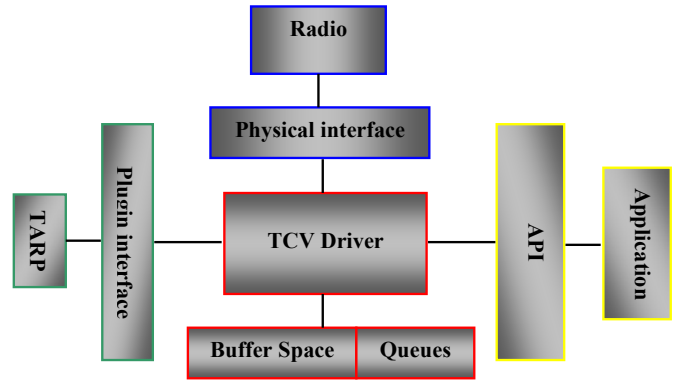


Figure 3. TARP within PicOS

Figure 3 shows the place of TARP in PicOS, in a configuration as a plugin, corresponding to a target deployment on an eCOG-based card.

#### 5. Summary

We have outlined parts of the Olsonet framework suitable for a variety of applications in diverse areas. Olsonet extends this framework to provide a platform for ad hoc multihop networks, and to build commercial applications with this paradigm.

#### 6. References

- [1] Pawel Gburzynski, “Protocol Design for Local and Metropolitan Area Networks”, Prentice Hall, 1996
- [2] Pawel Gburzynski and Jacek Maitan. “Specifying Control Programs for reactive Systems”, Proceedings of PDPTA’98, Las Vegas, July 13-16, 1998, pp. 1702-1709
- [3] Pawel Gburzynski, <http://sheerness.cs.ualberta.ca/~pawel/SIDE/product.html>
- [4] Olsonet Communications, “Applications for Ad-hoc Networks”, Business Notes, 2002.
- [5] Olsonet Communications, “PicOS”, Technical Documentation, 2003.
- [6] Olsonet Communications, “PicOS Virtual Transceiver Interface”, Technical Documentation, 2003.
- [7] Olsonet Communications, “Tiny Ad-hoc Routing Protocol (TARP)”, Technical Documentation, 2002.