Full TCP/IP for 8-Bit Architectures

Adam Dunkels
adam@sics.se

Presented by: Haining Mo
Outline

• Abstract
• Characteristics of lwIP
• Characteristics of uIP
• Implementation of uIP
• Http Server: A Practical Application of uIP
• Future Work
Abstract

• TCP/IP: A global standard for communication.
• Traditional TCP/IP: Large code size & memory usage.
• lwIP (Lightweight IP) & uIP (MicroIP).
• Applied to 8-bit & 16-bit: 8051, ARM, X86, etc.
• Running on: Network sensor node, Web server, etc.
• Implemented in C programming language.
Characteristics of lwIP

lwIP: Full scaled but simplified TCP/IP

• UDP, IP, ICMP, etc.
• TCP: Slide window, RTTM, Congestion control, Retransmission, Urgent data, etc.
• Multiple local network interfaces.
• Layered structure.
• Modular to be extended.
• More 32-bit arithmetic.
• Code size: 14588 bytes on X86; 21756 bytes on AVR.
Characteristics of uIP

- Minimal set of features for a full TCP/IP stack
- TCP, ICMP, IP, ARP.
- Nonstrict layered structure.
- Shared sending and reception data buffer among different layers.
- Code size: 5188 bytes on X86; 5164 bytes on AVR.
  < 3000 lines of code, easy to read.
Characteristics of *uIP*: TCP

Implemented In **TCP**

- Full state transition diagram.
- Simple retransmission mechanism cooperated with application.
- Simultaneous connections restricted by available RAM.
- Connection state management e.g. sequence number, ack number, etc.
- 16-bit checksum & 32-bit add on 8-bit architectures.
- Standard TCP header structure and header option support.
Characteristics of uIP: TCP

Not Implemented In TCP

- Slide window.
- Retransmission queue.
- SACK (Selective Ack) or SNACK (Selective Negative Ack)
- Urgent data transmission.
Characteristics of uIP: IP

Implemented in IP:

- Standard IP header structure.
- IP packet formation & parse.
- Fragmented IP packets reassembling.

Not Implemented in IP:

- Routing protocols: static routing & dynamic routing.
- Simultaneous fragmented IP packets reassembling.
- IP options
Characteristics of *uIP*: ARP

Implemented in *ARP*:

- ARP cache.
- Update ARP cache according to received IP packets.
- Send ARP request to retrieve destination MAC address.
- Send ARP reply to ARP request packet.
Characteristics of $uIP$: ICMP

Implemented in **ICMP**:
- ICMP Echo packet: to set local IP address.
- ICMP Echo Reply packet.

**UDP** Not Implemented in $uIP$. 
Characteristics of uIP: API

- API (Application Programming Interface) Not Implemented in uIP.
- SOCKET Unsupported.
- Users have to combine their applications and device drivers with uIP source code.
Characteristics of uIP:RFC Compliance

• RFC 1122 includes formal requirements in TCP/IP.
• Requirements divided into two categories:
  related to host to host communication.
  e.g. TCP option handling.
  related to application to protocol stack communication.
  e.g. report TCP error conditions to application.
• Compliance to first kind requirements is a must.
• Compliance to second kind requirements is flexible.
Characteristics of uIP:RFC Compliance

- Requirements in first kind implemented in uIP.
- Ensure correct communication with other hosts compliant to RFC requirements (Not only those implementing full TCP/IP).

- Some requirements in second kind ignored in uIP.
- Only affect communication within a host.
- Decrease code size and improve efficiency.
Characteristics of uIP: Memory

- Memory allocation: connection states, arp cache, incoming packets, outgoing packets, etc.
- lwIP: dynamic memory allocation.

- uIP: static memory allocation.
- Arp cache & connection states: fixed length array.
- A global data buffer:

  For incoming packets: shared by MAC layer, IP layer, TCP layer, Application layer.
  For outgoing packets: shared by MAC layer, IP layer, TCP layer.
## Characteristics of uIP: Memory

### Incoming packets

<table>
<thead>
<tr>
<th>Global buffer</th>
<th>600 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC header 14 bytes</td>
<td>→ Parsed by Ethernet device driver</td>
</tr>
<tr>
<td>IP header 20 bytes</td>
<td>→ Parsed by uIP IP layer</td>
</tr>
<tr>
<td>TCP header 20 bytes</td>
<td>→ Parsed by uIP TCP layer</td>
</tr>
<tr>
<td>Application data</td>
<td>→ Handled by Application e.g. http parser</td>
</tr>
</tbody>
</table>
## Characteristics of uIP: Memory

### Outgoing packets

<table>
<thead>
<tr>
<th>Layer</th>
<th>Bytes</th>
<th>Added by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global buffer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC header</td>
<td>14 bytes</td>
<td>uIP ARP: src &amp; dst MAC address</td>
</tr>
<tr>
<td>IP header</td>
<td>20 bytes</td>
<td>uIP IP layer</td>
</tr>
<tr>
<td>TCP header</td>
<td>20 bytes</td>
<td>uIP TCP layer</td>
</tr>
<tr>
<td>Application data</td>
<td>&lt;576</td>
<td></td>
</tr>
<tr>
<td>600 bytes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementation of uIP: Main Loop

1. Check for packets → Process ethernet pkt
2. Process ethernet pkt → Application events → Output packets
3. Check for timeout → Process timeout
4. Process timeout → Application events → Output packets
Implementation: Process Timeout

Main purpose: Retransmission

1. Timer = Timer - 1
2. Timer = 0
   - Y: Return
   - N: count of rtx > threshold
     - Y: Close connection
       - Return
     - N: SYN_SENT: tx SYN packet
8. SYN_RCVD: tx SYN | ACK packet
   - ESTABLISHED: invoke app to reproduce data packet
Implementation: Process Ethernet Pkt

1. DMA read
2. Pkt into global buf
3. IP or ARP
   - IP
     - ARP info into cache
     - Process IP packet
   - ARP
     - ARP Request or Reply
       - Reply
         - ARP info into cache
       - Request
         - Send ARP reply pkt
Implementation: Process IP Packet

1. Verify IP header info
2. Local IP Set
   - N: Error return
   - Y: TCP Packet
3. TCP Packet
   - N: Error return
   - Y: Process TCP Packet
4. ICMP Echo
   - N: Error return
   - Y: Set local IP address
Implementation: Process TCP Packet

1. One of current connections should accept the TCP packet
   - Y: Find the correct connection
     - Verify packet sequence No
       - Verify packet ack No
         - Process based on TCP state transition diagram
           - If needed, call app to process
   - N(SYN received): Find an available connection
     - Set connection fields values
       - State=SYN_RCVD
         - Send back SYN|ACK
Http Server: A Practical Application

Hardware Platform:

• 80C51: 8-bit MCU (Micro Control Unit).
  Connected to AX88796 & RAM.

• AX88796: Local Bus Fast Ethernet Controller.
  Packet Sending: DMA write from host memory to device memory;
    Send command to initiate sending process.
  Packet Reception: Interrupt notifies ethernet packet arrival;
    DMA read from device memory to host memory.
  Host memory: the global buffer shared by MAC, IP, TCP and Http.
Http Server: A Practical Application

What do we need besides uIP:

• Ethernet device driver: etherdev_read()
  etherdev_send()
  etherdev_poll()

• Http Parser: Parse http request e.g. “GET /about.html”.
• A Simple Http File System: Realized via array(each array—an http file).
• Http File Management: Put file content into sending buffer.
• Interaction With uIP: Use the global buffer.
Http Server: A Practical Application

Characteristics of This Simple Http Server:

• 10 simultaneous client connections at most.
• Implement the response to client “GET” request.
• Transmit html file to client using uIP.
• Only one file transfer within one connection.
• Fixed html file content & length.
• No database or script support.
Http Server: A Practical Application

A Real Debugging Scenario:

uip_conns[]: connection info e.g. state, seq no, ack no, rtx timer.
tcpstate flags: connection state.

0: CLOSED; 1: SYN_RCVD; 3: ESTABLISHED; 5: FIN_WAIT2; 7: TIME_WAIT.
Future Work

• Prioritized Connections.
• Security Aspects.
• Address Auto-configuration.
• Improving Throughput.
Thank you!