

# Porting a Network Cryptographic Service to the RMC2000: A Case Study in Embedded Software Development

Stephen Jan

Paolo de Dios

Stephen A. Edwards

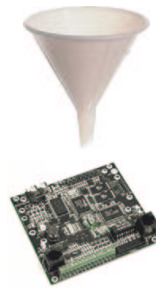
•

Dept. of Computer Science

Columbia University, New York, NY

## Porting iSSL to an Embedded Board

# iSSL



## Reengineering iSSL for Embedded

# iSSL



## iSSL

Transport layer security protocol

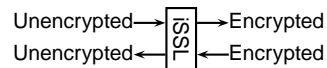
Layers on top of TCP/IP to secure communications

Computationally intensive

Perfect for a coprocessor: really just a filter

Written for Unix workstations

- Berkeley sockets
- fork
- bignum library



## The RMC2000



8-bit Z80-derived Rabbit 2000  
microcontroller @ 30 MHz  
512K flash  
128K SRAM

64K address space; accesses 1M using bank-switching

10Base-T network interface

Dynamic C development environment

Source for TCP/IP, UDP, and ICMP included

Cheap development kit

## Issues

### Different APIs

- Rabbit's TCP/IP vs. sockets
- Process model (coroutines vs. fork)
- Interrupts (manual vs. signals)

### Missing APIs

- Filesystem
- Arbitrary-precision integer package

### Efficiency

- Assembly vs. C implementation of AES cipher

## Dynamic C

Development environment supplied by Rabbit

C-like language (ANSI minus many features plus others)

## Dynamic C's Inline Assembly

```
#asm nodebug
InitValues::
    ld hl,0xa0;
c start_time = 0; // Inline C
c counter = 256; // Inline C
    ret
#endasm
```

Used in error-handling routines that catch hardware and software exceptions.

## Concurrency through Coroutines

```
for (;;) {
    costate {
        waitfor( tcp_packet_port_21() );
        // handle FTP connection
        yield(); // Force context switch
    }
    costate {
        waitfor( tcp_packet_port_23() );
        // handle telnet connection
    }
}
```

We used this to handle multiple network connections.

## New storage classes

```
shared float a, b, c; // Interrupts disabled during access
main() {
    protected int state1; // Battery-backed
    ...
    _sysIfSoftReset() // restore protected variables
}

root int func1() { ... } // in root memory
#memmap root // in root memory
#asm root
...
#endasm

xmem int func2() { ... } // in extended memory
```

## Missing APIs

- iSSL used a filesystem for optional logging  
We removed support for logging
- RSA cipher uses arbitrary-precision “bignum” library  
We omitted RSA and only ported AES (Rijndael)
- No `malloc()` or `free()`  
Replacements work very differently; changed program to use statically-allocated memory. Changed to fixed key and block sizes.
- No `random()` in Dynamic C libraries  
Wrote a new implementation

## API Differences: Networking

```
int echo_server() {
    int sock, newsock, len;
    struct sockaddr_in addr;
    char buf[1024];

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        return -1;

    memset(addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    addr.sin_port = htons(MYPORT);
    if (bind(sock, (struct sockaddr *) &addr,
            sizeof(struct sockaddr_in)) < 0) return -1;
    if (listen(sock, LISTENQ) < 0) return -1;
    for (;;) {
        if ((newsock = accept(sock, NULL, NULL)) < 0)
            return -1;
        if ((len = recv(newsock, buf, LEN, 0)) < 0)
            return -1;
        if (send(newsock, buf, len, 0) < 0) return -1;
        close(newsock);
    }
}

int echo_server()
{
    tcp_socket sock;
    int status;
    char buf[1024];

    sock_init();
    for (;;) {
        tcp_listen(&sock, PORT, 0, 0, NULL, 0);
        sock_wait_established(&sock, 0, NULL, &status);
        sock_mode(&sock, TCP_MODE_ASCII);
        while (tcp_tick(&sock)) {
            sock_wait_input(&sock, 0, NULL, &status);
            if (sock_gets(&sock, buf, LEN))
                sock_puts(&sock, buf);
        }
    }
}
```

Berkeley Sockets (Original)

Dynamic C API

## API Differences: Concurrency

```
listen(listen_fd)
for (;;) {
    accept_fd = accept(listen_fd);
    if ((childpid = fork()) == 0) {
        // process request on accept_fd
        exit(0); // terminate process
    }
}

for (;;) {
    costate {
        tcp_listen(socket1, TLS_PORT, ...);
        while (sock_established(socket1) == 0) yield;
        // handle request
    }
    costate {
        tcp_listen(socket2, TLS_PORT, ...);
        while (0 == sock_established(socket2)) yield;
        // handle request
    }
    costate {
        tcp_listen(socket2, TLS_PORT, ...);
        while (0 == sock_established(socket2)) yield;
        // handle request
    }
    costate {
        // drive TCP stack
        tcp_tick(NULL);
    }
}
```

Sockets + Fork

Custom + costate

## Performance Experiments

Compared ported implementation of AES cipher with Rabbit-supplied hand-optimized assembly.

Found assembly was 15–20× faster

Tried hand-optimizing C (moved data to “root” memory, unrolled loops, etc.); only 20% improvement

Code size fairly good: assembly only 9% smaller

## Conclusions

Ported iSSL to an RMC2000 development board

Biggest challenges from different APIs

Missing APIs also a problem

Performance disappointing: assembly implementation of AES cipher 15–20× faster

Bottom line: look carefully at what assumptions a program makes about its environment before porting it.

Future work: Dealing with different APIs.

Synthesizing wrappers? Code reengineering tool?

