

## CS6811: Research Seminar on Reconfigurable Hardware

Lockwood : Spring 2002

### *Review of: Protocol Boosters: Applying Programmability to Network Infrastructures*

- Authors:

- William S. Marcus (Bellcore)
- Ilija Hadzic (University of Pennsylvania)
- Anthony McAuley (Bellcore)
- Jonathan Smith (University of Pennsylvania)

- Published in:

- IEEE Communications Magazine: Oct 1998
- On-line as: <http://www.cis.upenn.edu/~boosters/commag.ps>

- Summarized by: John Lockwood

## Protocol Boosters

- Methodology for protocol design
  - Overcomes slow evolution of low-level protocols
  - Add, delete, or delays messages
- Base Protocol
  - Packets send and received by endpoints
- Protocol Booster
  - Pair of nodes added between network endpoints

## Booster Types

- Sample Boosters
  - Compression
  - Error Correction
- Arbitrary Ordering is Possible
  - (but not always logical)
  - Example: Compression + FEC
  - Example: FEC + Compression

## Topology of Protocol Boosters

- Concatenated Boosters
  - Different boosters over different links
- Nested Boosters
  - Booster operates on boosted protocol
- Some orderings are Illogical
  - Compression followed by error correction

## Booster-to-Booster Communication

- Intra-Booster Exchange Messages
  - Originated by source booster
  - Terminated by destination booster
- Routing Assumptions
  - Static routing
    - Same Boosters originate / terminate booster
  - Dynamic routing
    - Would require identification of message

## Policies for Installing a Booster

- Observed network behavior
  - Eg: Congested Load > Threshold
- Packet source / destination fields
  - Boost packets between Host X and Host Y
  - For example: Add forward error correction for traffic traversing over satellite link
- Time of day
  - Eg: “Business Only” traffic from 9-5pm

## FZC Booster

- FEC Error Correction
  - Speculatively transmits additional data
  - Anticipates future errors
  - Low latency (No round-trip delay)
- Forward “EraZure” Correction (FZC)
  - Intended for wireless networks with bit errors
  - Transmitter
    - Adds  $h$  parity packets
  - Receiver
    - Removes  $h$  parity and regenerates packets

## Other members of the TCP/IP booster family

- ARQ Booster
  - Automatic Repeat Request
  - Allows “Local” retransmission
    - Common on wireless links
- Reorder Booster
  - Delays packets based on sequence number
- Error Detection Booster

## Booster Implementation

- Software
  - Linux 2.0.32 Kernel
    - Dynamically loadable kernel modules (lkm)
  - i386 Architecture
- Booster Instance
  - Invoked with `ioctl` system call
- Booster Sequence
  - Concatenated chain of *booster instances*
- Booster Trap
  - Directs packets to *booster sequence*

## FZC Booster Implementation

- Operation of Transmitter
  - Overwrites 16-bit “id” field w/sequence number
  - Buffer  $k$  packets
  - Zero-pads packets to size of largest packet
  - Stores original protocol number and sequence number in packet trailer
  - Performs FZC matrix multiply
  - Produces total of  $h$  overcode packets
  - Sets Proto = “Protocol Booster”
  - Sets Booster Header =  $\langle \text{FZC}, k, \text{Seq\#} \rangle$

## FZC De-Booster Implementation

- Operation of Receiver
  - Passes all other packets
  - Receives buffer of  $k$  overcode
  - Releases packets when
    - All  $k$  packets present
      - No Work needed
    - Number of received packets + parity =  $k$ 
      - Performs Matrix computation
    - Content Replacement
      - No fix possible
  - Restores original Header

## Implementation Testing

- Wireless Environment
  - Actually run on 10base-T Ethernet
  - Pentium 166 Endpoints
- Process Created
  - Random Loss
  - Bursty Loss [consecutive packets]
- UDP/IP Packets
  - Packet size = 512 bytes
  - Block size = 20 packets
  - Trial size = 1000 packets
- Ran `ttcp -u -t ; ttcp -u -r`

## Implementation Effectiveness

- No Parity
  - Loss rate is same as specified
  - Overhead of the protocol software doesn't kill end-to-end performance
- More Parity
  - Less loss
- 0% Overcode : 9.6 Mbps
- 4% Overcode (K=50, h=2) : 7.7 Mbps
- 30% Overcode (k=20, h=6) : 3.9 Mbps

## Programmable Protocol Processing Pipeline (P4)

- Field Programmable Gate Array
  - Six Altera 8000 FPGAs
  - FIFOs between each FPGA
  - Crossbar between devices
- Traffic Flows
  - OC3 ATM
  - Virtual Circuits

## FPGA Implementation of FEC in P4

- Environment
  - R = 1/2 Convolutional Encoder (One FPGA)
  - Viterbi Decoder (Four FPGAs)
  - Fore PCA200 NIC (!)
  - Bit error rates varied from  $10^{-12}$  to  $10^{-4}$
- Results
  - Throughput benefit starts with BER =  $10^{-7}$
  - 10x Performance Boost at BER =  $10^{-5}$
  - TCP/IP Otherwise stalls with BER =  $10^{-4}$

## Conclusions

- Method for Protocol Construction
  - Incremental
  - Dynamic
- Forms Families of Protocols
  - Start with optimistic protocol
  - Adds functionality on as-needed basis
  - In contrast to design for worst-case
- Implementation
  - Software : Linux
  - Hardware : FPGAs