Report Number: WUCSE-2004-70

2004-09-02

Robust Header Compression (ROHC) in Next-Generation Network Processors

David E. Taylor, Andreas Herkersdorf, and Gero Dittmann

Robust Header Compression (ROHC) provides for more efficient use of radio links for wireless communication in a packet switched network. Due to its potential advantages in the wireless access area andthe proliferation of network processors in access infrastructure, there exists a need to understand the resource requirements and architectural implications of implementing ROHC in this environment. We presentan analysis of the primary functional blocks of ROHC and extract the architectural implications on next-generation network processor design for wireless access. The discussion focuses on memory space andbandwidth dimensioning as well as processing resource budgets. We conclude with an examination of... Read complete abstract on page 2.

Robust Header Compression (ROHC) in Next-Generation Network Processors

David E. Taylor, Andreas Herkersdorf, and Gero Dittmann

Complete Abstract:

Robust Header Compression (ROHC) provides for more efficient use of radio links for wireless communication in a packet switched network. Due to its potential advantages in the wireless access area andthe proliferation of network processors in access infrastructure, there exists a need to understand the resource requirements and architectural implications of implementing ROHC in this environment. We presentan analysis of the primary functional blocks of ROHC and extract the architectural implications on next-generation network processor design for wireless access. The discussion focuses on memory space andbandwidth dimensioning as well as processing resource budgets. We conclude with an examination of resource consumption and potential performance gains achievable by offloading computationally intensiveROHC functions to application specific hardware assists. We explore the design tradeoffs for hardware as-sists in the form of reconfigurable hardware, Application-Specific Instruction-set Processors (ASIPs), andApplication-Specific Integrated Circuits (ASICs).

2004-70

Robust Header Compression (ROHC) in Next-Generation Network Processors

Authors: Taylor, David E.; Herkersdorf, Andreas; Doring, Andreas; Dittmann, Gero

Abstract: Robust Header Compression (ROHC) provides for more efficient use of radio links for wireless communication in a packet switched network. Due to its potential advantages in the wireless access area and the proliferation of network processors in access infrastructure, there exists a need to understand the resource requirements and architectural implications of implementing ROHC in this environment. We present an analysis of the primary functional blocks of ROHC and extract the architectural implications on next-generation network processor design for wireless access. The discussion focuses on memory space and bandwidth dimensioning as well as processing resource budgets. We conclude with an examination of resource consumption and potential performance gains achievable by offloading computationally intensive ROHC functions to application specific hardware assists. We explore the design tradeoffs for hardware assists in the form of reconfigurable hardware, Application-Specific Instruction-set Processors (ASIPs), and Application-Specific Integrated Circuits (ASICs).

Type of Report: Other

Header Compression (ROHC) in Next-Generation Network Processors

David E. Taylor, Andreas Herkersdorf, Andreas Döring, Gero Dittmann

WUCSE-2004-70

September 13, 2002

Applied Research Laboratory
Department of Computer Science and Engineering
Washington University in Saint Louis
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130
david.taylor@wustl.edu

This report is a summary of an investigation performed by David E. Taylor during a summer internship at the IBM Zurich Research Laboratory under the guidance of Andreas Herkersdorf. Andreas Döring and Gero Dittmann provided invaluable insight and contributions to the analysis. Herkersdorf is currently with the Institute of Integrated Systems, Technical University of Munich, Germany (e-mail: a.herkersdorf@tum.de). Döring and Dittmann are with IBM Zurich Research Laboratory (e-mail: {ado,ged}@zurich.ibm.com).

Abstract

Robust Header Compression (ROHC) provides for more efficient use of radio links for wireless communication in a packet switched network. Due to its potential advantages in the wireless access area and the proliferation of network processors in access infrastructure, there exists a need to understand the resource requirements and architectural implications of implementing ROHC in this environment. We present an analysis of the primary functional blocks of ROHC and extract the architectural implications on next-generation network processor design for wireless access. The discussion focuses on memory space and bandwidth dimensioning as well as processing resource budgets. We conclude with an examination of resource consumption and potential performance gains achievable by offloading computationally intensive ROHC functions to application specific hardware assists. We explore the design tradeoffs for hardware assists in the form of reconfigurable hardware, Application-Specific Instruction-set Processors (ASIPs), and Application-Specific Integrated Circuits (ASICs).

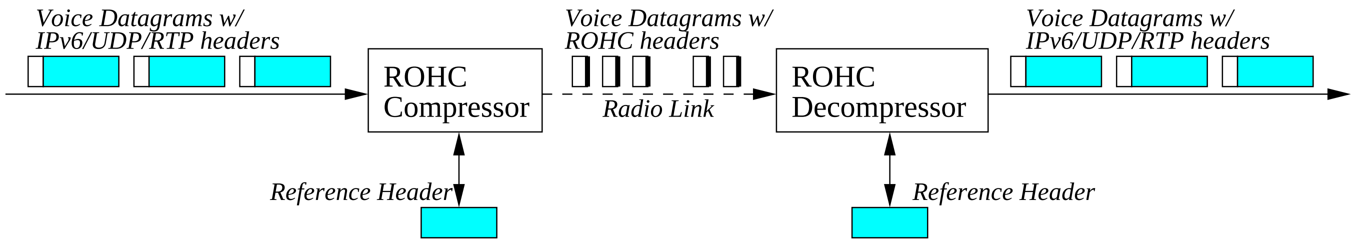
Voice Datagrams w/ IPv6/UDP/RTP headers
ROHC Compressor
Voice Datagrams w/ ROHC headers
Radio Link
ROHC Decompressor
Voice Datagrams w/ IPv6/UDP/RTP headers
Reference Header
Reference Header

Figure 1: Example scenario of ROHC compression/decompression of IPv6/UDP/RTP headers for communication over a radio link.

1 Introduction

Header compression provides for more efficient use of link bandwidth in a packet switched network by leveraging header field redundancies in packets belonging to the same flow. The key observation that many packet header fields such as source and destination addresses remain constant throughout the duration of a flow while other fields such as sequence numbers change predictably allows header compression techniques to transmit only a few bytes of header information per packet. Typically, reference copies of the full headers are stored at the compression and decompression points in order to reliably communicate and reconstruct original packet headers. Particulars of packet header field classification into static and dynamic sets depend upon the communication protocols and encoding techniques employed by the compression scheme. We provide a brief history of header compression techniques in Section 2.

Link efficiency comes at the cost of processing and memory resources in the nodes communicating over the link. While this may not be a favorable tradeoff in many environments such as communication over optical fiber, it is particularly advantageous for communication over radio links due to their high cost relative to the provided bandwidth [1]. Recently introduced as a new standard by the IETF, Robust Header Compression (ROHC) seeks to provide reliable header compression for efficient use of links with high loss rates [2]. As an example of the potential value of ROHC, consider a mobile handset transmitting voice datagrams using IPv6/UDP/RTP as shown in Figure 1. In order to achieve a high perceived quality of response, voice datagrams are typically on the order of 20 bytes while IPv6/UDP/RTP headers are on the order of 100 bytes. ROHC achieves typical header sizes of one to four bytes which could reduce the overhead in our example by a factor of 100 and the total bandwidth consumption by a factor of six. In order to facilitate our discussion, an overview of ROHC is provided in Section 3.

Clearly, ROHC deployment for wireless networking requires implementation at both ends of the radio link. While implementation in handsets, mobile terminals and “appliances” raises many interesting issues, our discussion focuses on the use of ROHC at access and aggregation nodes in wireless networks. As shown in Figure 2, Base Station Controllers (BSCs) which concentrate multiple links from Base Station Transceivers (BSTs) in cellular networks are primary examples. Industry estimates suggest that BSC link rates may soon reach 2.5 Gb/s, imposing a uni-directional throughput constraint of over 7.8 million packets per second for minimum size packets. For orthogonal ingress and egress processing of our 120 byte voice datagram example, the throughput constraint is over 5.2 million packets per second. In order to understand the required resources of implementing ROHC in this environment, we provide an analysis of the primary functional blocks employed by ROHC in Section 4.

Due to the need for programmability, flexibility, and rapid deployment of new applications, services, and protocols at network edges, access routers and aggregation nodes increasingly employ network processors. Designed for high-speed packet processing, network processor architectures seek to achieve maximum

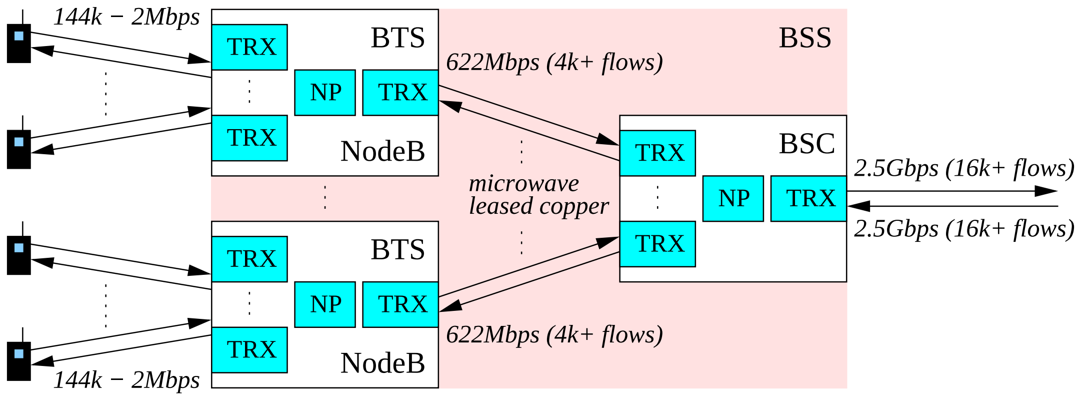

Figure 2: Block diagram of 3G wireless access network. Link bandwidths are based on future capacity estimates for 3G wireless systems (WCDMA, EDGE, GSM).

flexibility while meeting the real-time throughput constraints imposed by the supported link rates. As in most systems, flexibility is provided via software programmable processors, either General Purpose Processors (GPPs) or processors with optimized instruction sets. Based on our functional analysis, we extract architectural implications of ROHC implementation on network processor design for the wireless access environment in Section 5. A key result is that the processor memory interfaces should be dimensioned to provide an aggregate bandwidth greater than eight times the bandwidth of the supported link in order to support worst case traffic patterns. This poses a challenge for processors designed to support high speed links or many aggregated links, such as those targeted to the wireless access infrastructure. Based on results from preliminary implementation efforts, ROHC may easily saturate instruction per packet budgets in next generation network processors [3]. In order to meet throughput constraints and maximize the number of available software instructions per packet, many network processors also contain optimized datapaths and application-specific hardware assists for redundant and computationally intensive tasks. While many processor architectures include hardware assists for essential ROHC functions such as packet classification and Cyclic Redundancy Check (CRC) computations, there is no support for ROHC encoding and decoding functions which comprise approximately one third of the per-packet workload. We envision that this will change in the future; hence, we examine the resource requirements and potential performance gains of implementing ROHC encoding and decoding functions in application-specific hardware assists in the form of reconfigurable hardware, Application-Specific Instruction set Processors (ASIPs), and Application-Specific Integrated Circuits (ASICs) in Section 6.

Note that we assume a working knowledge of the IP and UDP protocols. The aforementioned Real-time Transport Protocol (RTP) provides “end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services” [4]. For the purpose of our discussion, it is only important to note that each header contains a sequence number (SN) and timestamp (TS) used for maintaining packet ordering and calculating link jitter, respectively.

2 A Brief History of Header Compression

The notion of header compression was introduced by Jacobson in 1990 for the purpose of improving interactive terminal response of low-speed serial modem links [5]. Nine years later, Degermark, Nordgren, and Pink developed techniques for compressing permutations of IPv4 headers, IPv6 base and extension headers, TCP and UDP headers, and encapsulated IPv4 and IPv6 headers for transmission over lossy links [6]. That same year Casner and Jacobson added a specification for compressing IP/UDP/RTP headers commonly referred to as CRTP [7].

Seeking to improve performance over lossy links with long round-trip times (RTT), a group of researchers from Ericsson and Lulea University proposed Robust Checksum-based Header Compression (ROCCO) as an Internet Draft in June 1999 [8]. Coupled with transport and link layer protocols capable of partitioning checksum coverage into sensitive and insensitive regions [9], ROCCO was designed to perform well with audio and video codecs which tolerate bit errors in data frames. Studies showed that ROCCO compressed headers to half of the size of those provided by CRTP and remained robust over links with BER rates an order of magnitude higher [10]. In July 2000, Ericsson and Japan Telecom successfully completed a field trial of VoIP over WCDMA using ROCCO [11].

In July 2001, the Robust Header Compression (ROHC) framework was presented as a new standard for header compression by a large working group of the IETF [2]. While designed to be a general framework extensible to new protocol stacks, much of the standard focuses on efficiency and robustness of IP/UDP/RTP compression for wireless communication over radio links. Several related standards have been proposed, including a specification for completely removing headers of IP/UDP/RTP packets, termed 0-byte compression, for use over existing cellular links based on GSM and IS-95 [12][13][14]. In May 2002, several companies participated in a successful first trial of the major parts of the ROHC standard including robustness tests over emulated WCDMA/3G links [11].

3 Header Compression with ROHC

The general approach of Robust Header Compression (ROHC) is to establish a common context at the compressor and decompressor by transmitting the full packet header, then gradually transition to higher levels of compression via the use of various encoding techniques and packet formats. ROHC is designed to be a flexible header compression framework capable of supporting several protocol stacks. General packet formats, compressor and decompressor finite state machines, modes of operation, error recovery and correction mechanisms, and encoding methods are defined at the framework level. Each supported protocol stack defines a “profile” within the ROHC framework. Profiles fully specify the detailed packet formats, state transition logic, and state actions as well as the encoding methods used for each header field in the protocol stack.

The ROHC framework and profiles for RTP, UDP, ESP, and uncompressed headers are defined in [2]. A specification for running ROHC over the Point-to-Point Protocol (PPP), commonly used for initialization and communication of control information, is defined in [14]. As mentioned in the previous section, a Link Layer Assisted (LLA) profile for IP/UDP/RTP, which is capable of completely removing packet headers, is defined in [13][12].

For the purpose of our study, we focus on supporting the general ROHC framework as well as the standard IP/UDP/RTP compression profile. RTP/UPD/IP compression operates via the following principle: establish functions from the RTP sequence number (SN) to other fields, then reliably communicate the SN. When functions change, parameters must be updated. The following sections briefly describe the major aspects of the ROHC framework and associated IP/UDP/RTP profile in order to facilitate discussion of our functional analysis and architectural implications on next-generation network processor design. Note that we delve into the details of field encoding and decoding techniques in Section 3.5 as these comprise one third of the processing workload; hence, our analysis of application-specific hardware assists in Section 6 focuses on these functions.

3.1 Assumptions

ROHC was designed under assumptions about the supporting layers and environment of use. We list several of the key assumptions here:

- Channels of a single radio link can have different BER, bandwidth, etc.
- Multi-access per channel is achievable by maintaining multiple contexts per channel identified by Context Identifiers (CIDs)
- Channels between compressor and decompressor must maintain ordering and not duplicate packets
- Packet length is provided by lower layers, most likely the link layer
- ROHC deals with residual errors in delivered headers via internal mechanisms to prevent or reduce damage propagation to higher layers
- Link layer must provide for compression parameter negotiation

3.2 Packet Formats

In order to reliably initialize static header fields and dynamic header field functions at the compressor and decompressor, ROHC defines several packet formats at the framework level and allows individual profiles to define profile-specific packet formats. For the purpose of our discussion, it only is necessary to note the major packet types. Initialization and Refresh (IR) packets carry static header field information as well as dynamic header field function parameters. Compressed header formats may be defined by the profile; however at minimum, they must carry a Context Identifier (CID). Note that the CID field may be eliminated if only one context exists on the given channel; otherwise, the CID field may either define 16 contexts per channel or 16k contexts per channel by selecting either the “small” or “large” CID size.

Feedback from decompressor to compressor may be sent as acknowledgments in individual packets or encapsulated in compressed packets, termed “piggybacked” feedback, if a compressed context exists in the reverse direction. Acknowledgments may positively acknowledge successful decompression (ACK) or negatively acknowledge decompression failures due to incorrect dynamic header field function parameters (NACK) or static header field information (STATIC-NACK).

3.3 Compressor & Decompressor Finite State Machines

ROHC compression is essentially the interaction between compressor (CFSM) and decompressor (DFSM) finite state machines. Logically, there is one state machine pair (CFSM/DFSM) per context. Both machines start in the lowest compression state and gradually transition to higher states, yet transitions need not be synchronized. Normally, only the CFSM temporarily transits back to lower states. The DFSM only transits back upon context damage detection.

The states of the Compressor Finite State Machine (CFSM) are listed below with their associated properties:

- **Initialization & Refresh (IR)**: Initialize static context at decompressor; recover from errors; transmit complete header plus additional information.

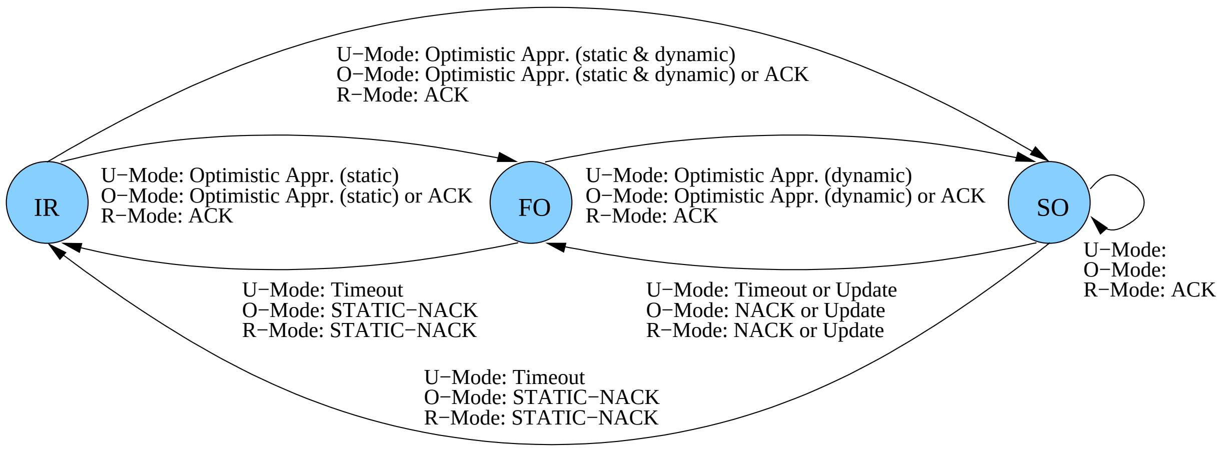
U−Mode: Optimistic Appr. (static & dynamic)
O−Mode: Optimistic Appr. (static & dynamic) or ACK
R−Mode: ACK
IR
FO
SO
U−Mode: Optimistic Appr. (static)
O−Mode: Optimistic Appr. (static) or ACK
R−Mode: ACK
U−Mode: Optimistic Appr. (dynamic)
O−Mode: Optimistic Appr. (dynamic) or ACK
R−Mode: ACK
U−Mode:
O−Mode:
R−Mode: ACK
U−Mode: Timeout
O−Mode: STATIC−NACK
R−Mode: STATIC−NACK
U−Mode: Timeout or Update
O−Mode: NACK or Update
R−Mode: NACK or Update
U−Mode: Timeout
O−Mode: STATIC−NACK
R−Mode: STATIC−NACK

Figure 3: Compressor Finite State Machine (CFSM) for Robust Header Compression (ROHC).

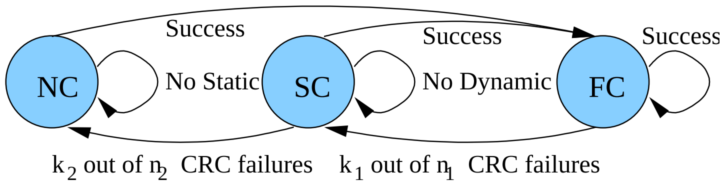

Figure 4: Decompressor Finite State Machine (DFSM) for Robust Header Compression (ROHC).

- **First Order (FO)**: Efficiently communicate packet stream irregularities; occasionally transmit dynamic packet header field information (usually partially compressed); occasionally update static packet header fields.

- **Second Order (SO)** Maintain optimal compression efficiency; compressed header is completely predictable given the sequence number; compressor is confident that the decompressor has correct SN function parameters; transmit encoded dynamic fields.

A state machine diagram of the CFSM is shown in Figure 3. The states of the Decompressor Finite State Machine (DFSM) are listed below with their associated properties:

- **No Context (NC)**: No successful decompression verified by CRC; transit to Full Context (FC) state upon receipt of static and dynamic information and successful decompression verified by CRC.

- **Static Context (SC)**: Normally, successful decompression of a header sent by compressor in First Order (FO) state warrants transition to Full Context (FC) state; transit to NC upon several decompression failures of headers sent in FO state as detected by CRC.

- **Full Context (FC)**: Valid context with static and dynamic header field information; transit to SC upon repeated decompression failures as detected by CRC.

A state machine diagram of the DFSM is shown in Figure 4. State transitions are addressed within the context of our analysis provided in Section 4. We refer the reader to [2] for more detailed explanations of states, their uses and properties.

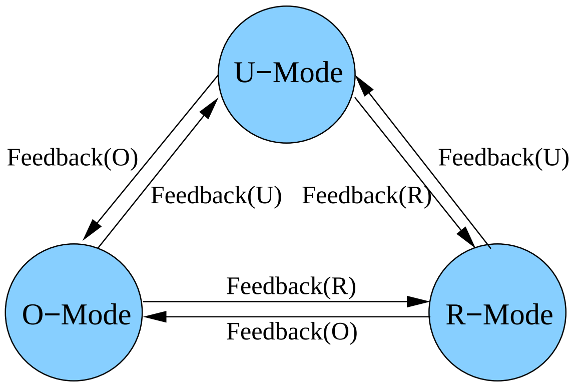

Figure 5: Diagram of compression mode transitions in ROHC; feedback arrows denote multi-way handshakes between compressor and decompressor.

3.4 Modes of Operation

ROHC modes of operation determine state machine transition logic and state actions. The CFSM and DFSM states remain the same for all modes. The optimal mode of operation depends upon environmental characteristics such as feedback abilities, error probability and distribution, etc. The ROHC specification states that all implementations must include all three modes.

Initially, all contexts must begin in *Unidirectional mode (U-Mode)* in which packets are sent from compressor to decompressor only, as the return path is assumed to be unavailable or "undesirable". CFSM transitions in U-Mode are due to periodic timeouts and irregularities in dynamic header field patterns. U-Mode is less efficient with higher probability of loss propagation than bi-directional modes. *Bidirectional Optimistic Mode (O-Mode)* utilizes a feedback channel for error recovery requests; hence, it does not employ periodic refresh as in U-Mode. O-Mode maximizes compression efficiency, minimizes feedback channel usage, and reduces error propagation; however, context invalidation may be higher than R-mode during long error or loss bursts.

Bidirectional Reliable Mode (R-Mode) intensively uses a feedback channel to acknowledge all context updates. Note that not all packets update the context. R-Mode employs "stricter" logic to prevent synchronization loss of CFSM/DFSM; therefore, it maximizes robustness against loss and damage propagation. It may have a lower probability of context invalidation than O-mode, but it also may deliver more damaged headers when the context is invalid.

As shown in Figure 5, compression mode transfers may occur from any mode to any mode. The decision to move from one mode to another is initiated by the decompressor. The ROHC specification describes in detail the variables and multi-way handshakes, denoted by the "feedback" arrows in Figure 5, to reliably transition from one mode to another. All feedback sent between compressor and decompressor during a mode transition must by protected by a CRC.

3.5 Encoding Methods

The encoding method for dynamic fields in the packet headers is selected based on the dynamic pattern of the specific field. Hence, several encoding methods may be employed for the dynamic fields in a single packet header. Note that while the encoding methods are specified for the ROHC framework, several of the methods are specific to IP/UDP/RTP headers.

3.5.1 Least Significant Bits (LSB) Encoding

LSB encoding is applied to header fields subject to small changes. Quite simply, the k least significant bits of the field are transmitted. The decompressor derives the original value using a reference value, v_{ref}, which must include the untransmitted bits. Correctness is guaranteed if the compressor and decompressor use interpretation intervals in which the original value resides and the transmitted k LSBs are unique to the original value.

LSB Interpretation Interval Function The LSB interpretation interval may be expressed as a function:

$$f(v_{ref}, k) = [v_{ref} - p, v_{ref} + (2^k - 1) - p] \tag{1}$$

For any k, the k LSBs must uniquely identify a value in $f(v_{ref}, k)$. p is a shifting parameter that may be tuned for efficiency. For example:

- For increasing values, set p to -1; the interpretation interval becomes:

$$f(v_{ref}, k) = [v_{ref} + 1, v_{ref} + 2^k] \tag{2}$$

- For non-decreasing values, set p to 0.
- For small negative changes and larger positive changes, it may be desirable to have $\frac{3}{4}$ of the interval used for positive changes; in this case, set p to $2^{k-2} - 1$; the interpretation interval becomes:

$$f(v_{ref}, k) = [v_{ref} - 2^{k-2} + 1, v_{ref} + 3 * 2^{k-2}] \tag{3}$$

The compressor uses as v_{ref_c} the last compressed value protected by a CRC. The decompressor uses as v_{ref_d} the last decompressed value verified by a CRC. k is selected as the minimum value such that v falls in the interpretation interval $f(v_{ref_c}, k)$. This selection function is referred to as $g(v_{ref_c}, v)$.

3.5.2 Window-based LSB (WLSB) Encoding

Window-based LSB (WLSB) Encoding provides for robust LSB encoding when the compressor is unable to determine the exact reference value, v_{ref_d}, in use by the decompressor. The compressor maintains a sliding window of possible v_{ref_d} values and calculates k such that all v_{ref_d} candidates produce the correct v. Letting $v_{ref_{min}}$ and $v_{ref_{max}}$ be the lower and upper bounds of the sliding window, respectively, then k is chosen as follows:

$$k = max(g(v_{ref_{min}}, v), g(v_{ref_{max}}, v)) \tag{4}$$

The window is advanced based on positive acknowledgments (ACKs) or the window width limit.

3.5.3 Scaled RTP Timestamp Encoding

Due to fixed sampling periods employed in real-time applications such as audio and video, the RTP Timestamp (TS) usually increases by an integral of a fixed time interval. When such fixed intervals exist, TS may be derived by the linear function:

$$TS = TS\_SCALED * TS\_STRIDE + TS\_OFFSET \tag{5}$$

where TS_STRIDE is the fixed time interval, TS_SCALED is the integral multiple, and TS_OFFSET is the linear offset.

TS_STRIDE is explicitly communicated from compressor to decompressor. TS_OFFSET is implicitly communicated by sending several uncompressed TS values from which the decompressor can extract its value. Note that TS_OFFSET is updated locally at TS wraparound via special interpretation interval rules. Following initialization, only TS_SCALED is communicated between compressor and decompressor. Specifically, it is sent compressed via WLSB or timer-based encoding.

3.5.4 Timer-based RTP Timestamp Encoding

When fixed sampling intervals are employed by applications and packets are transmitted in lock-step with the sampling, the RTP Timestamp (TS) not only increases by an integral of a fixed time interval, TIME_STRIDE, but it may also be approximated by a linear function of the time of day. Deviations in this approximation are due to delay jitter; therefore, the approximation is refined using k LSBs of the scaled TS.

In order to determine the number of LSBs needed to refine the linear approximation, the compressor maintains a sliding window of potential TS reference values employed by the decompressor as well as their arrival times at the compressor. Let T_j and a_j be the timestamp and arrival time for header j.

Delay jitter perceived at the decompressor, J, is due to contributions from inter-packet jitter before compression, Max_Jitter_BC, and jitter properties of the communication channel, Max_Jitter_CD. Note that Max_Jitter_CD is a parameter that is initialized and may be updated throughout the duration of a context. The compressor calculates Max_Jitter_BC for each new header n according to Equation 6 for all j in the sliding window.

$$Max\_Jitter\_BC = max\{|(T_n - T_j) - \frac{a_n - a_j}{TIME\_STRIDE}|\} \tag{6}$$

Accounting for clock quantization errors, the total delay jitter may be expressed as follows:

$$J = Max\_Jitter\_BC + Max\_Jitter\_CD + 2 \tag{7}$$

Finally, the compressor selects the k LSBs necessary for refinement according to the following equation:

$$k = \lceil \lg(2 \times J + 1) \rceil \tag{8}$$

Note that this calculation is similar to WLSB encoding with $p = 2^{k-1} - 1$. The compressor sends k bits of the scaled timestamp to the decompressor.

The decompressor maintains as a reference the scaled timestamp, T_{ref}, and arrival time, a_{ref}, of the last successfully decompressed header. The approximated timestamp, T_{approx}, of an arriving packet n at time a_n is calculated as follows:

$$T_{approx} = T_{ref} + \frac{a_n - a_{ref}}{TIME\_STRIDE} \tag{9}$$

This approximation is refined via the k LSBs transmitted by the compressor using LSB decoding with $p = 2^{k-1} - 1$.

3.5.5 IPv4 Identifier (IP-ID) Offset Encoding

Offset encoding assumes that the IP-ID increases for each outgoing packet. Therefore, it is more efficient to encode the offset relative to the RTP Sequence Number (RTPSN) where:

$$Offset = IPID - RTPSN \tag{10}$$

For transmission, the offset is compressed and decompressed using WLSB encoding with $p = 0$. Network byte order is synchronized using the NBO or NBO2 flag in the header.

3.5.6 Extension Header Compression

ROHC defines a list compression technique for IP header extension chains and CSRC lists in RTP headers. We do not anticipate a large contribution to processing and memory resource consumption due to list compression; therefore, it is excluded from our analysis.

3.6 CRC

For initialization packets, ROHC employs an 8-bit CRC covering all packet fields, excluding the payload. The polynomial is given below:

$$1 + x + x^2 + x^8 \tag{11}$$

For compressed headers, ROHC calculates the CRC over all of the header fields of the original packet. Header fields are split into two groups, namely CRC-STATIC and CRC-DYNAMIC based on their frequency of change between packets. The static portion of the CRC is calculated over a list of the CRC-STATIC fields concatenated in the order in which they appear in the original packet header, then stored with the context. The CRC calculation continues over the CRC-DYNAMIC fields concatenated in their original order. The static portion of the CRC is only recomputed when CRC-STATIC fields change. Note that the CRC is used to detect bit errors incurred during transmission as well as invalid context information caused by missed or incorrect context updates; therefore, two CRC widths are used to ensure uniqueness between CRC values for context updates. The polynomials used for compressed headers are listed below:

$$1 + x + x^3 \tag{12}$$

$$1 + x + x^2 + x^3 + x^6 + x^7 \tag{13}$$

4 Functional Analysis of ROHC

In order to understand the architectural implications of implementing ROHC in a network processor, we must analyze the fundamental operations and interactions of the functional blocks described in the ROHC specification. While there is inevitable variance depending upon implementation decisions and environments, a fundamental analysis provides valuable insight into issues such as resource requirements and scalability. It is important to note that we do not address ROHC performance or compression efficiency, and we attempt to keep our analysis as implementation and traffic independent as possible. Given the absence of open ROHC implementations and packet traces of high bandwidth links with aggregated 3G traffic, we seek to establish useful bounds for network processor architects. To facilitate our discussion without significant loss of generality, we present a logical block diagram of an ROHC Compressor/Decompressor in Figure 6. For the following discussion, we assume that a single logical entity in the network processor is responsible for ROHC compression and decompression.

As reflected in Figure 6, a network processor may support several links. Since there may be multiple channels per link and the number of contexts and compression profile may be configured on a per-channel basis in ROHC, the network processor must provide a mechanism for identifying the link and channel of all arriving and departing packets. As specified by ROHC, the network processor must also provide the length of link layer frames passed to the compressor/decompressor.

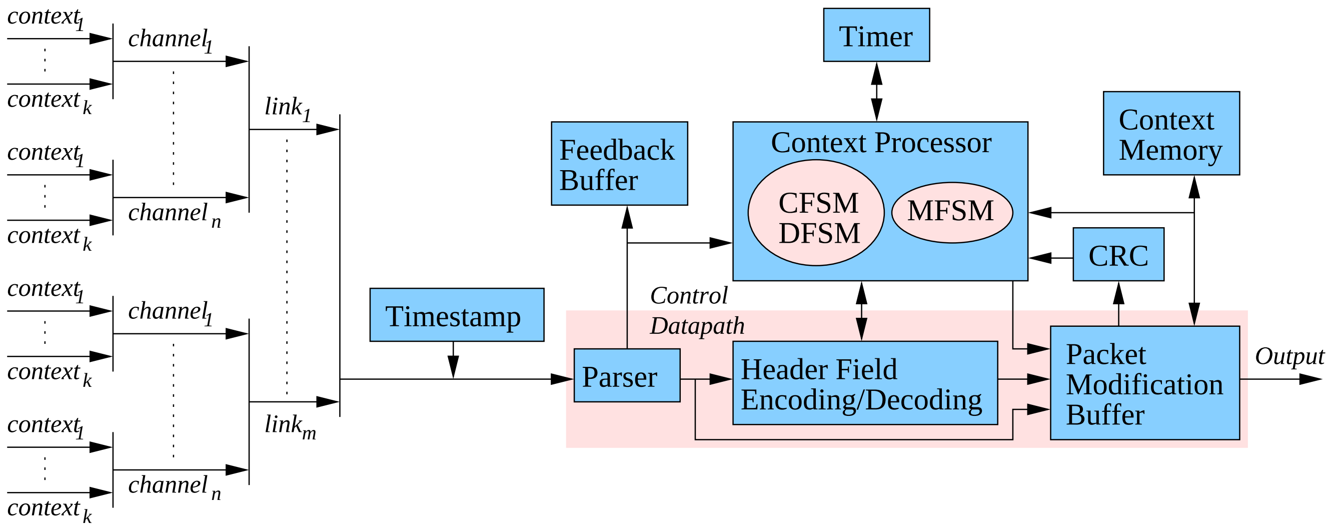

Figure 6: Logical block diagram of ROHC Compressor/Decompressor for IP/UDP/RTP compression.

The Packet Modification Buffer is assumed to be a local or dedicated memory space where packets may be temporarily stored during compression and decompression. A large, presumably external, Context Memory provides storage for all of the per-context information such as state, mode, reference values, etc. The Feedback Buffers allow "piggybacked" feedback to be stored and later retrieved when the context addressed by the feedback is made active via receipt of a packet or specific feedback processing. The remaining blocks are discussed in more detail in the following sub-sections.

4.1 Context Processor

The Context Processor may be viewed as implementing the core of the ROHC framework. In essence, the Context Processor coordinates packet parsing, context fetches and updates, state and mode transitions, field encoding, and error checking for each ROHC packet. While we assume the general structure and interfaces of the Context Processor remain constant for all ROHC profiles, all packets are processed by three profile-specific sub-blocks: a Packet Parser, a Compression or Decompression Finite State Machine (CFSM/DFSM), and a Mode Finite State Machine. Note that ROHC profiles are assigned on a per-context basis; therefore, the processing performed by each sub-block may change on a per-packet basis for a Context Processor serving many channels.

4.1.1 Packet Parser

The Packet Parser may be viewed as a finite state machine capable of decoding the many permutations of packet header formats. The minimal set of headers recognized by the parser must include the base and extension headers of the protocols supported by the ROHC profile as well as the profile-specific packet headers. Based on a link and channel identifier, the parser may determine the direction and profile of the packet; for example, ROHC compression for IP/UDP/RTP.

One of the primary functions of the Packet Parser is returning the context identifier (CID) for the packet so that the context information may be retrieved from Context Memory. For packets with compressed headers, the CID is contained in the ROHC header. The Packet Parser must simply search for the bit-pattern identifying the CID field. For uncompressed headers, the CID must be assigned by a block capable of binding the packet to an established ROHC flow or allocating a new CID for the flow.

Based on the mode and compression/decompression state of the context, the Packet Parser must also extract the header fields requiring encoding or decoding. These fields are passed by the Context Processor to the appropriate encoding or decoding block. When the parser detects "piggybacked" feedback, the information is stored in per-channel, per-context feedback buffers. Finally, the Packet Parser must extract CRC values for error detection.

4.1.2 Compression/Decompression Finite State Machines (CFSM/DFSM)

Compression and Decompression Finite State Machines (CFSM/DFSM) dictate the format of transmitted packets and feedback. We seek to determine the amount of state information required per-context by the CFSM/DFSM. Coupled with the results of packet parsing, field encoding/decoding, and CRC computations, the mode of operation stored with the context information determines state transitions and actions. In the case of decompression, all modes share the same Decompression Finite State Machine (DFSM) transition logic shown in Figure 4. "Success" is defined as a decompression of a header that passes the CRC check. Note that transitions to lower compression states occur when a certain percentage of the most recent header decompression attempts fail. The transition parameters used by the DFSM are configurable, such as the number of failures in the Full Context (FC) state, k_2, out of the last n_2 packet headers. Implementation requires a sliding window of width n which indicates decompression "success" or "failure" for the most recent n packet headers. If the sum of "failures" in this window reaches k, then a downward transition is triggered. This window along with two pairs of parameters must be stored with the context information. Decompression state actions such as field decoding and feedback transmission differ for each mode.

While state transitions and actions of the Compressor Finite State Machine (CFSM) differ for each mode of operation, we observe that the structure remains static as shown in Figure 3. Due to the lack of feedback channel in Unidirectional Mode (U-Mode), all forward transitions depend on the "confidence" of the compressor that the decompressor has the static and/or dynamic context necessary to successfully decompress the header. "Confidence" is established based on the "optimistic approach principle" which stipulates that the compressor send several packets containing the same information according to the lower compression state. ROHC uses this qualitative specification to allow for implementation flexibility. A straightforward implementation simply requires a parameter n and a counter $count$ for each transition utilizing "optimistic approach".

Downward transitions occur due to periodic timeouts and pattern changes in the packet stream. Due to the lack of a feedback channel, the CFSM must periodically transit back to lower compression states. ROHC stipulates that the CFSM should transit back to the IR state less frequently than to the FO state; therefore, timeout values should be maintained with the context information. If the packet header to be compressed does not adhere to the established pattern, the CFSM must immediately transit back to the FO state in order to establish the new dynamic information.

Transitions in Bidirectional Reliable Mode (R-Mode) utilize the "secure reference principle" where "confidence" of synchronization with the decompressor is based solely on the receipt of acknowledgments (ACKs); therefore, no parameters need be stored with the context. Downward transitions are triggered by negative acknowledgments of static (STATIC-NACK) or dynamic (NACK) context information or the need for updates. State transition logic for Bidirectional Optimistic Mode (O-Mode) is a blend of the principles employed by U-Mode and R-Mode. Forward transitions in O-Mode depend on the "optimistic approach principle" but the optimistic confidence may be confirmed via use of optional ACKs. Likewise, downward transitions occur due to the receipt of NACKs and STATIC-NACKs which eliminates the need for timeouts.

Table 1: Parameters stored with context information for implementation of a CFSM/DFSM.

FSM	Mode	Transition	Parameter(s)	Est. Size
DFSM	All Modes	FC$\rightarrow$SC	n_1, k_1, win_1	4 B
	All Modes	SC$\rightarrow$NC	n_2, k_2, win_2	4 B
CFSM	U-Mode	IR$\rightarrow$FO	n_{stat}, $count_{stat}$	2 B
	U-Mode	FO$\rightarrow$SO	n_{dyn}, $count_{dyn}$	2 B
	U-Mode	IR$\rightarrow$SO	n_{sd}, $count_{sd}$	2 B
	U/O-Mode	SO$\rightarrow$IR	$timeout_1$	2 B
	U/O-Mode	SO$\rightarrow$FO	$timeout_2$	2 B
	U/O-Mode	FO$\rightarrow$IR	$timeout_3$	2 B
Total				20 B

Based on the preceding analysis, the CFSM/DFSM may be implemented as a single profile-specific sub-block of the Context Processor. Each context must simply store the current state and mode, along with the parameters reflected in Table 1.

4.1.3 Mode Finite State Machine (MFSM)

We assume that a mode variable is maintained for each context. The value of the variable controls context attributes such as state actions and usable packet types. In many cases, the behavior of compressor or decompressor is restricted during mode transitions.

4.2 Timer

The Timer block shown in Figure 6 simply denotes the existence of a timing mechanisms available for maintaining timeout timers while in Unidirectional Mode (U-Mode). At minimum, the Context Processor must maintain per-context timeout values relative to a free-running timer. When a packet arrives on a given context, the Context Processor must detect if a timeout has occurred for the context. Several design options exist for such a mechanism and selection of the ideal approach largely depends on the implementation platform.

4.3 Field Encoding & Decoding

Header field encoding and decoding are major consumers of processing resources. We first present an analysis of the computational requirements of the encoding schemes presented in Section 3.5. Memory resources required for encoding parameters are addressed in Section 5.6.

Table 2 presents the frequency of fundamental operations employed in each encoding and decoding technique. Note that the number of operations required for LSB encoding and decoding are based upon the analyses presented in the following sub-sections. Also note that the number of operations required for timer-based TS encoding depends upon the width, j, of the sliding window of reference values. The set of fundamental operations has direct implications on the Instruction Set Architecture (ISA) of network processor for pure software implementations of ROHC.

Table 2: Core function frequency in ROHC encoding techniques for IP/UDP/RTP compression, where j is the width of the sliding window of reference values. All functions operate on integer operands.

Technique	Add/Sub.	Mult.	Div./Mod.	Compare	LSL	$lg()$	Ceiling	Abs.
LSB Encode	6	0	0	0	1	2	1	2
WLSB Encode	12	0	0	1	2	4	2	4
IP-ID Encode	13	0	0	1	2	4	2	4
SN Encode	12	0	0	2	2	4	2	4
Scaled TS Encode	13	0	1	1	2	4	2	4
Timer TS Encode	$3j+3$	1	j	$j-1$	0	1	1	0
LSB Decode	5	1	1	1	1	0	0	0
IP-ID Decode	6	1	1	1	1	0	0	0
SN Decode	5	1	1	2	1	0	0	0
Scaled TS Decode	6	2	2	1	1	0	0	0
Timer TS Decode	7	1	2	1	1	0	0	0

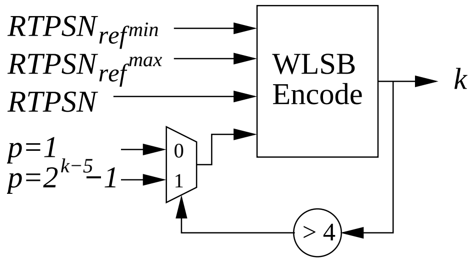

Figure 7: Block diagram of RTP Sequence Number (RTPSN) Encoder employing WLSB Encoder block.

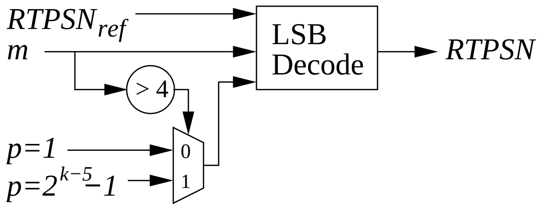

Figure 8: Block diagram of RTP Sequence Number (RTPSN) Decoder employing LSB Decoder block.

We observe that WLSB encoding is used as a building block for several encoding techniques. As reflected in Table 2, most encoding techniques contribute only a few additional computations on top of those required for WLSB encoding or LSB decoding. In order to illustrate how a WLSB Encoder may be used as a functional component, a block diagram of an RTPSN Encoder is shown in Figure 7. Note that shifting parameters are selected based on feedback from the WLSB result. This ensures that as the width of the interpretation interval grows beyond a threshold, it is proportionally shifted to provide $\frac{31}{32}$ of the interval for positive changes. Figure 8 illustrates how an LSB Decoder may be used as a functional component in a RTPSN Decoder. Complementary to the encoder, the shifting parameters are selected based on the number of bits received in the ROHC header.

We also observe that the core operations, such as logarithms, employed in LSB encoding are expensive to implement in both software and hardware. Due to the potential usefulness of WLSB Encoder and LSB Decoder blocks, the following sub-sections discuss their design and efficient implementation. We note that

$$g(v_{ref_c}, v) = min(k | v_{ref_c} - a2^{k-b} + c \leq v \leq v_{ref_c} + 2^k - 1 - a2^{k-b} + c) \tag{16}$$

$$g(v_{ref_c}, v) = min(k | \lg(\frac{v - v_{ref_c} - c + 1}{1 - a2^{-b}}) \leq k \leq \lg(\frac{v_{ref_c} - v + c}{a}) + b) \tag{17}$$

the structure of the functions allows for optimizations which eliminate the high cost of implementing the core operations independently.

4.3.1 WLSB Encoder

As previously stated, WLSB encoding chooses the minimum number of least significant bits of a value to send such that the bits will uniquely identify the original value given a set of likely reference values in use by the decompressor. This task is formalized in Equation 4. The LSB Selection Function employed by the WLSB encoder may be expressed as follows:

$$g(v_{ref_c}, v) = min(k | v_{ref_c} - p \leq v \leq v_{ref_c} + 2^k - 1 - p) \tag{14}$$

Note that the shifting parameter p may take on values that are functions of k, as specified in the previous examples. In order to find a closed-form solution for k, a parameterized function of p must be set. Based on the previous examples, the following function is proposed.

Parameterizing the LSB Selection Function Let p be represented by the function:

$$p = a \times 2^{k-b} - c \tag{15}$$

with passed parameters a, b, and c. The following examples show how p may be set:

- For $p = 0$, set $a = 0, b = 0, c = 0$
- For $p = -1$, set $a = 0, b = 0, c = 1$
- For $p = 2^{k-2} - 1$, set $a = 1, b = 2, c = 1$

Replacing p with the parameterized function, the selection function takes the form of the inequality shown in Equation 16. Solving for k yields the inequality shown in Equation 17. Since k must be an integral number of bits, the inequalities and min function may be eliminated as follows:

$$k = \lceil \lg(\frac{v - v_{ref_c} - c + 1}{1 - a2^{-b}}) \rceil \tag{18}$$

Note that the WLSB Encoder performs this computation twice, once with the minimum reference value and once with the maximum reference value, selecting the maximum result. If the value of k computed by the WLSB Encoder is smaller than the minimum value allowed by the available packet format, then the smallest available value of k is used.

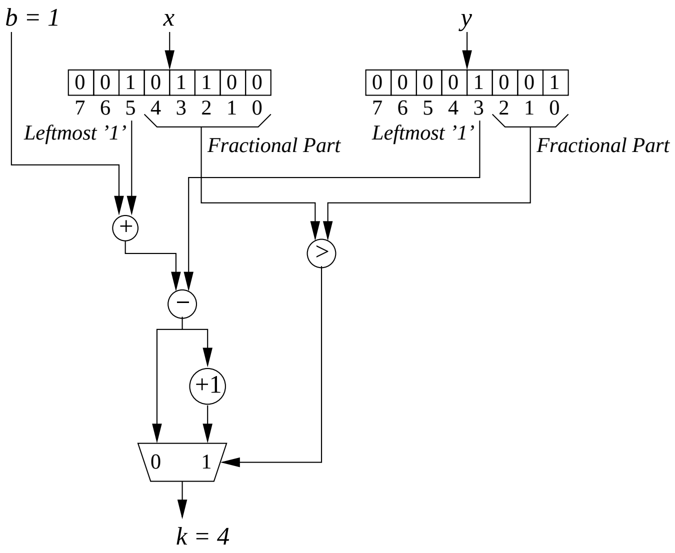

Figure 9: Example of LSB Selection Function computation via bit string searches. Equation is of the form $k = \lceil b + \lg x - \lg y \rceil$.

Efficient Implementation While Equation 18 may easily be specified in a high-level language for implementation in a General-Purpose Processor (GPP), we continue our analysis to investigate opportunities for optimized implementations. We first note that the negative exponential in the denominator may lead to the need for floating-point division if implemented literally. Logarithms are also typically expensive to implement in software due to the need for approximation arithmetic or lookup tables. In an effort to avoid these expensive computations, we re-formulate the LSB Selection Function as follows:

$$k = \lceil b + \lg\left(|v - v_{ref_c} - c + 1|\right) - \lg\left(|2^b - a|\right)\rceil \tag{19}$$

This form of the function eliminates the need for floating point division and reduces the exponential to a simple Logic Shift Left (LSL) operation. The need for an efficient mechanism for computing the binary logarithms remains. Taking advantage of the ceiling operation, the logarithms and remaining arithmetic are efficiently computed using bit string searches.

Let $x = |v - v_{ref_c} - c + 1|$ and let $y = |2^b - a|$; Equation 19 becomes:

$$k = \lceil b + \lg x - \lg y \rceil \tag{20}$$

We can easily compute the integral part of the binary logarithms by locating the bit-location of the most significant '1'. We refer to the bits to the right of this bit-location as the fractional part. A simple compare of the fractional parts of the logarithms determines whether or not to add an additional bit to the value of k found by adding b and the integral part of $\lg x$ minus the integral part of $\lg y$. To clarify, an example starting from Equation 20 is shown in Figure 9.

4.3.2 LSB Decoder

As previously stated, the decompressor derives the original value from the received LSBs using a reference value, v_{ref_d}, and an interpretation interval. Let m be the number of LSBs received by the compressor. Let $|m|$ be the value of the received LSBs. The original value, v_d, is determined to be the value in the interpretation interval whose m LSBs are equal to $|m|$. Like the WLSB Encoder, the LSB Decoder must

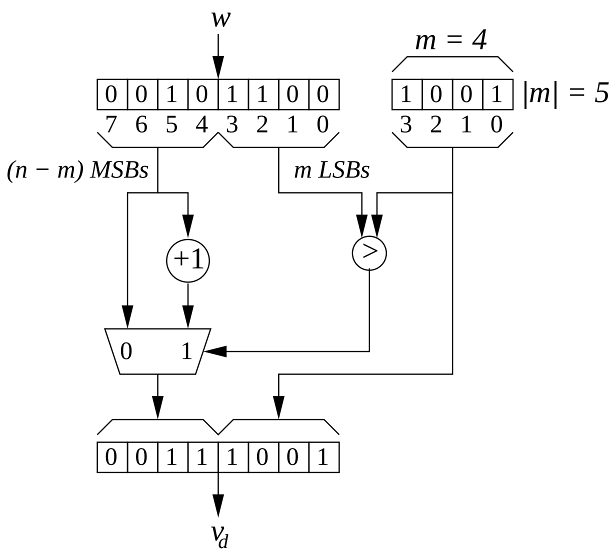

Figure 10: Example of LSB Decoder operation, where $w = v_{ref_d} - a2^{m-b} + c$ is the left end-point of the interpretation interval and m is the number of received LSBs.

employ the parameterized function of p. Hence, the interpretation interval employed by the LSB decoder may be expressed as follows:

$$f(v_{ref_d}, m) = [v_{ref_d} - a2^{m-b} + c, v_{ref_d} + 2^m(1 - a2^{-b}) - 1 + c] \tag{21}$$

Selecting the value within the interval matching the received LSBs may be done in several ways. We focus on an efficient computational implementation which avoids lookup tables or multiple iterations.

Efficient Implementation Let w be the left end-point of the interpretation interval; therefore, $w = v_{ref_d} - a2^{m-b} + c$. Let $|w|_m$ be the value of the m LSBs of w, which may be expressed as:

$$|w|_m = (w)modulo(2^m) \tag{22}$$

Let $|w|_{n-m}$ be the value of the $n - m$ MSBs of w, where n is the width in bits of w. This may be expressed as:

$$|w|_{n-m} = w - |w|_m \tag{23}$$

We note that the original value, v_d, may be selected from the interpretation interval expressed in Equation 21 as follows:

If $|m| \geq |w|_m$, **then** $v_d = (|w|_{n-m} + |m|)$; **else** $v_d = (|w|_{n-m} + 2^m + |m|)$

While this may easily be specified in high-level GPP instructions, we also provide an example of an optimized solution using bit-level optimizations as shown in Figure 10.

4.4 CRC

ROHC requires support for CRC computations based on three different polynomials in both components. In the compressor, a CRC checksum is generated from the original header and is transmitted with the header depending on the compression mode. In the decompressor, this checksum is compared to the checksum

Table 3: Area and propagation delay for CRC calculation. ASIC results based on area optimized synthesis targeting IBM Cu-11 process assuming worst-case delays. FPGA results based on synthesis with Synplicity Synplify Pro targeting a Xilinx Virtex-E (-8) device using worst-case delay estimates. FPGA area estimates assume 423 LUT/FF pairs per mm^2.

	ASIC			FPGA		
Width (bits)	Logic Depth	Area mm^2	Tp (ns)	LUTs	Area mm^2	Tp (ns)
8	3	0.0013	1.73	33	.078	2.15
16	4	0.002	1.79	57	.135	3.03
32	4	0.0037	1.87	78	.185	3.42

of the restored header. The computation of CRC checksums with generator polynomials of small degree (here 3, 7, and 8) is comparatively simple. Furthermore, the checksums can be computed on a consecutive sequence of input words. For this type of task look-up based methods are preferred in software implementations and can be used for hardware as well. For a polynomial degree d and input width w bits per lookup, table sizes of $d \times 2^w$ bits result. For software implementations utilizing standard memory widths, it is worth illustrating the performance tradeoffs that arise when deciding how the table entries for polynomial degrees of 3 or 7 bits may be efficiently packed into memory words. For example, utilizing one memory word per 3-bit entry simplifies table address generation but is space inefficient. Concatenating 10 3-bit entries and storing them in a single 32-bit word is space efficient but consumes additional processing cycles for table address generation due to the need for an integer divide and shifting operations to extract the desired result from the resulting memory word. Hence, for CRC computations with polynomials of such a small degree, direct computation in software is worth considering. It consists of a mask and parity computation for each output bit. On a processor with a population count instruction, such as the SPARC V9, only a few instructions are needed per output bit.

In hardware implementations, the choice of input width of the computation dictates the size and depth of logic. Table 3 lists the area and logic depth for computing the three polynomials in parallel. Results show the area and speed for implementations with input widths of 8, 16, and 32 bits. Results for ASIC implementation are derived from area optimized synthesis targeting the IBM CU-11 process utilizing worst-case propagation delay estimates. Results for FPGA implementation are derived from synthesis with Synplicity Synplify Pro targeting a Virtex-E device utilizing worst-case propagation delay estimates for a device with a -8 speed grade. FPGA area estimates are explained in Section 6.1.1. Note that since the output size is 3, 7, and 8 bits, respectively, the result of the CRC computation fits in one 32-bit word. Hence, all three checksums could be implemented as an instruction set extension and comfortably combined into a single instruction on a 32-bit processor. The individual results could be extracted by shift operations. The same argument holds for a strongly-tailored ASIP implementation.

5 Architectural Implications for Network Processors

Insertion of ROHC compression and decompression in the packet processing path of a network processor presents several architectural implications. Header compression logically resides between the link layer, commonly called Layer 2, and the network layer, commonly called Layer 3, in the protocol stack. Inserting header compression and decompression in the ingress and egress packet processing paths of a network processor requires additional processing, memory, and interconnection resources. It is our intention to establish bounds on the amount of additional resources to serve as "engineering bands" for network processor

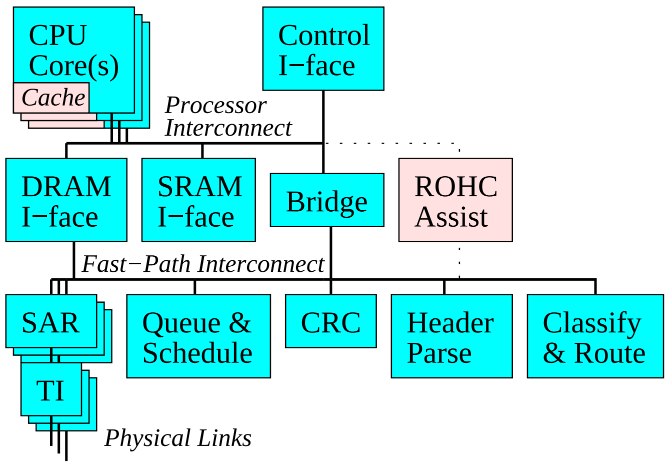

Figure 11: Block diagram of a generic network processor architecture utilizing hardware assists and hierarchical interconnection.

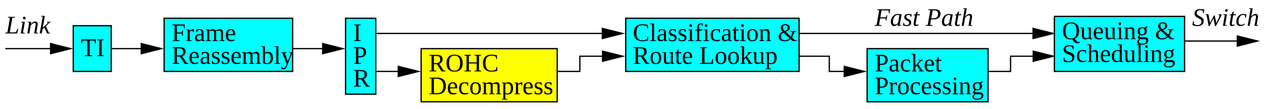

Figure 12: Logical block diagram of ingress data path including ROHC decompression.

architects. For the purpose of our analysis, we employ a generic network processor architecture utilizing hardware assists and hierarchical interconnection as shown in Figure 11. Note that Figure 11 includes an ROHC Assist hardware assist block. We motivate the need for this block in the following section and analyze the performance and resource tradeoffs in Section 6.

5.1 Functional Placement

A logic block diagram of the ingress datapath of a network processor supporting ROHC is shown in Figure 12. Note that following frame reassembly, packets containing compressed headers must be passed to the ROHC Decompressor with an identifier specifying the arrival link and channel as well as the frame length. Header decompression must precede Route Lookup & Classification, as the fields used for packet classification are contained in the original packet header. The network processor must support some form of Internal Packet Route (IPR) function capable of recognizing compressed headers and routing the packets to the ROHC Decompressor prior to classification. This could pose a problem for architectures that implement fixed datapaths between the Frame Reassembly and the Route Lookup & Classification blocks in order to maximize throughput for typical network traffic.

In order to transmit packets with compressed headers, the network processor must be able to instantiate an ROHC Compressor prior to Frame Segmentation as shown in Figure 13. As previously mentioned, this requires that an IPR have the ability to identify and pass packets requiring header compression to the ROHC Compressor along with the outgoing link, channel and Context Identifier (CID). Placement of the ROHC Compressor relative to Queuing & Scheduling depends on a number of factors such as the ROHC implementation platform and scheduling algorithms. The following sub-sections discuss the influences and requirements ROHC places on the various components of network processor architecture, beginning with the packet parsing and classification.

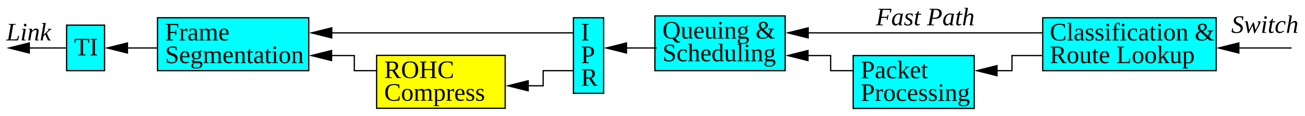

Figure 13: Logical block diagram of egress data path including ROHC compression.

5.2 Frame Reassembly & Packet Parsing

As previously mentioned, high-performance network processors often employ some form of "fast path" in which packets requiring "normal" processing are handled by optimized blocks, bypassing general purpose processing engines. One of the most fundamental and common tasks handled in a "fast path" is link layer frame segmentation and reassembly. ROHC defines a segmentation and reassembly protocol which may require modifications to fixed Frame Segmentation & Reassembly blocks. Note that the protocol is only used when a packet is larger than the largest size supported by the link layer; therefore, its use in many environments is obsolete. In order to offload the packet classification and route lookup functions, the "fast path" must include some form of header parser which extracts the required header fields. This functionality may be contained in the Frame Segmentation & Reassembly block or implemented as a separate component. In order to leverage "fast path" components for ROHC functionality such as flow classification and CRC computations, the header parser must recognize a new class of header formats defined by each ROHC profile. This suggests that a programmable header parsing block, similar to the one described in [15], would be a favorable architectural feature of a network processor supporting ROHC.

5.3 Packet Classifi cation

Within the context of ROHC, a flow refers to a sequence of packets that demonstrate the necessary redundancy in static fields as defined by the ROHC profile. In order to bind an uncompressed packet to an ROHC flow, the static fields of the packet headers must be compared against the set of established flows. This is precisely the function performed by packet classifiers which typically search filter tables used for network management and firewalls. Most packet classifiers are capable of performing filter matches on header fields such as the IP source and destination addresses, transport protocol, and transport source and destination ports. For ROHC compression of IPv6/UDP/RTP and IPv4/UDP/RTP, the fields that must be examined in order to identify a flow are reported in [2] and summarized in Table 4. The total search key sizes for IPv4 and IPv6 headers are 140 bits and 352 bits, respectively. Note that if the IPv6 Flow Label is used (non-zero) the number of bits to be examined could be as low as 20 bits.

At minimum, ROHC requires that the packet classification block support exact match lookups using search keys with configurable widths. This type of search may be efficiently implementing using hashing techniques; however, it is likely that more elaborate algorithms will be employed in packet classifiers employed by next-generation network processors. For high-performance packet classifiers, many network processor platforms make use of Ternary Content Addressable Memory (TCAM) or ASICs implementing proprietary algorithms. We note that the case of binding an IPv6/UDP/RTP header to an ROHC flow could pose a problem for such implementations, as the required 352 bit search key is wider than the maximum width provided by commercially available classifiers.

Ideally, entries in the classification table would contain the CID of the ROHC flow to be compressed. This implies that updates be generated by a block responsible for managing the CID space of each channel. Since there is no explicit flow termination signal in current packet switched networks, a suitable control block must manage the CID space of each channel. This control block could be integrated with the filter

Table 4: Packet header fields examined for classification of ROHC flows.

Header	Field	Size (bits)
IPv4	Version	4
	Protocol	8
	Src.Addr.	32
	Dest.Addr.	32
IPv6	Version	4
	Flow Label	20
	Next Hdr.	8
	Src.Addr.	128
	Dest.Addr.	128
UDP	Src. Port	16
	Dest. Port	16
RTP	SSRC	32
Total	IPv4/UDP/RTP	140
	IPv6/UDP/RTP	352
	IPv6 Flow Label	20

and route database manager of the packet classifier. At minimum, the control block must implement a CID allocation algorithm such as Least Recently Used (LRU), manage a set of connection timers, and generate appropriate feedback such as the "REJECT" feedback message used to inform a compressor that no compression context is available for a new flow.

5.4 Timestamping

A mechanism for assigning an arrival time to ingress packets is required in order to support timer-based RTP Timestamp encoding. Ideally, each packet should be stamped at the transmission interface, immediately following reassembly, and prior to any pre-decompression buffering. The arrival timestamp may be carried in an internal header, or shim, and passed to the decompressor along with the packet. Non-deterministic buffering delays prior to decompression should be kept to less than a few microseconds. RTP Timestamps are usually on the order of milliseconds; therefore, such small buffer delays should not make contributions to the jitter calculations performed during timer-based RTP Timestamp decoding.

5.5 Scheduling & Queuing

An in-depth discussion of the affects of ROHC on scheduling and queuing mechanisms in next-generation network processors in large wireless network aggregation nodes is beyond the scope of our discussion. However, we would like to enumerate a few peculiarities in the ROHC standard which imply that further study on this topic would be beneficial.

1. Non-deterministic processing time due to radio link properties and application behavior
2. Change of total packet length
3. Decompressor creates packets in reverse direction for acknowledgments in a bi-directional modes
4. Decompressor may create bursts of decompressed packets when using "negative caching"

5.6 Memory Resources

All header compression schemes achieve transmission efficiency by trading off memory resources. In essence, headers are redundantly stored instead of transmitted. We examine the amount of memory space and bandwidth required to support ROHC in large aggregation nodes.

5.6.1 Space Requirements

Since the compressor must maintain multiple reference values for sliding windows, there is a significant difference between the amount of memory space utilized by an ROHC compressor and decompressor. In IP/UDP/RTP compression, several fields require the compressor to store multiple reference values in a sliding window when WLSB encoding is used. In order to formulate an upper bound on capacity estimates, we account for per-context memory requirements of a compressor. As shown in Table 5, the memory space required for a reference header and encoding parameters for ROHC compression of IPv4/UDP/RTP and IPv6/UDP/RTP differ by only one byte. Assumptions guiding our choosing sliding windows of width 10 and encoding parameter sizes are provided in the Appendix. Using 244 bytes as the upper bound for reference header and encoding parameters and adding the 20 bytes for state machine transition parameters, 264 bytes seems to be a reasonable approximation for the per-context memory requirements for ROHC compression. We also performed this analysis for decompression and found that the per-context memory requirements for ROHC decompression of IPv4/UDP/RTP and IPv6/UDP/RTP are approximately 100 bytes less than that required for compression.

While the specific configuration of contexts per channel, channels per link, and links per port will vary depending on the application or access router configuration, we anticipate that a single network processor may need to support thousands of concurrent contexts. Based on our per-context estimates, total memory requirements for context information exceeds 1MB for four thousand flows. This implies that off-chip SDRAM should be used for context storage as on-chip memory resources are typically limited to a few kilobytes. While commodity SDRAM products provide ample space for context information storage, we examine the additional off-chip memory bandwidth required to support ROHC.

5.6.2 Bandwidth Requirements

ROHC memory bandwidth consumption will depend heavily upon implementation design decisions and target platforms. Similarly, dimensioning of memory interface bandwidth for network processors is difficult due to the heterogeneity of applications. Processor architects employ various “rules of thumb” in order to gain a first-order approximation of the required memory bandwidth. For the purpose of our analysis, we choose a conservative rule that states the memory interface should provide four times the bandwidth of the aggregate link rate. For example, a network processor supporting an aggregate link throughput of 1 Gb/s should employ a 4 Gb/s memory interface. This rule is based upon the assumption that a packet must be written to memory when received, read from memory for processing, written back to memory after processing, and read from memory for transmission for a total of four transits over the memory interface. It should be noted that some processor architectures employ register pipelines to avoid reading packets from memory for processing. Any packet modifications are applied when the packet is read from memory prior to transmission, resulting in a total of two transits over the memory interface. Given that the wireless access environment requires support for many low-speed or aggregated links, we envision that the number of processing engines in the network processor will be less than the number of links. An architecture employing register pipelines requires an additional packet memory between the ports and pipelines as packets may

Table 5: Context memory space usage for reference headers and encoding parameters for ROHC IP/UDP/RTP compression. Assumes $n = 10$ for WLSB window widths, see the Appendix for further explanation.

Header	Field	Size (bytes)	Ref.Hdr. (bytes)	Total (bytes)
IPv4	Version	0.5	0.5	0.5
	Hdr. Len.	0.5	0	0
	TOS	1	1	1
	Total Len.	2	0	0
	IP-ID	2	$n \times 2 + 5$	25
	Flags	0.5	0.5	0.5
	Frag. Offset	1.5	0	0
	TTL	1	1	1
	Protocol	1	1	1
	Hdr. Chk.	2	0	0
	Src. Addr.	4	4	4
	Dst. Addr.	4	4	4
Total		**20**		**37**
IPv6	Version	0.5	0.5	0.5
	Traffic Class	1	1	1
	Flow Label	2.5	2.5	2.5
	Payload Len.	2	0	0
	Next Hdr.	1	1	1
	Hop Limit	1	1	1
	Src. Addr.	16	16	16
	Dst. Addr.	16	16	16
Total		**40**		**38**
UDP	Length	2	0	0
	Checksum	2	0	0
	Src. Port	2	2	2
	Dst. Port	2	2	2
Total		**8**		**4**
RTP	Flags, CC, M, PT	2	2	2
	SSRC	4	4	4
	SN	2	$n \times 2 + 5$	25
	TS	4	$n \times 8 + 31$	111
	CSRC	0-60	0-60	0-60
Total		**12 - 72**		**134-194**
Total	**IPv4/UDP/RTP**	**40-100**		**183-243**
	IPv6/UDP/RTP	**60-120**		**184-244**

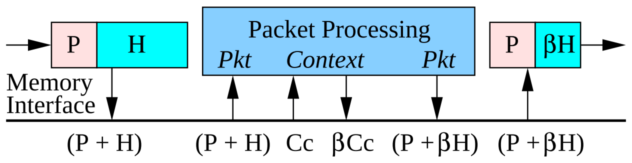

Figure 14: First-order memory bandwidth usage model for ROHC compression in the egress packet processing path of a network processor.

arrive simulataneously on each link. Since we seek to establish conservative bounds for the additional memory bandwidth required for ROHC, we utilize the single memory interface architecture for our analysis.

In order to gain a first-order approximation for the additional memory bandwidth required for ROHC, we must first establish some assumptions. Due to the large number of supported channels at an aggregation node in the network, we assume that context information is stored off-chip. We also assume that ROHC processing does not require additional reading and writing of packet data, as ROHC processing may utilize an on-chip buffer like the Packet Modification Buffer shown in Figure 6. Based on these assumptions, the only additional memory accesses generated by ROHC processing is for context fetches and updates.

As shown in Figure 14, we assume that packet headers are compressed by some factor β that ranges from zero to one. Initialization headers which contain the entire original header correspond to $\beta = 1$, while contexts with high compression efficiency corresponds to $\beta \leq 0.1$. While all of the context information must be fetched in order to decompress or compress a packet header, only a portion of the context needs to be updated and written back to memory. We make the first-order approximation that the amount of context information written back to memory is proportional to the compression factor β. In general, small headers require few updates of context information while larger headers induce more context information updates.

Based on these assumptions, we find that context accesses for ROHC compression contribute an additional $(1+\beta) \times C_c$ bytes per packet of memory bandwidth where C_c is the size of the compression context. Similarly, context accesses for ROHC decompression contribute an additional $(1+\beta) \times C_d$ bytes per packet of memory bandwidth where C_d is the size of the decompression context. As we seek to garner upper-bounds for worst-case design, we will continue our analysis for the compression case since C_c was found to be 264 bytes in the previous section, approximately 100 bytes more than C_d. Note that the analysis for decompression is symmetric with only a change in the value of the context size.

Letting H and P be the length in bytes of the packet header and packet payload, respectively, we assume that packet storage upon arrival and fetch prior to processing consumes $2 \times (P + H)$ bytes per packet of memory bandwidth. Packet storage after processing and header compression and packet fetch for transmission, requires $2 \times (P + \beta H)$. Given a fixed link rate, R, expressed in bytes per second, the number of packets per second equals $\frac{R}{P+H}$. Thus, the amount of memory bandwidth required for a system without ROHC processing is $4 \times R\frac{bytes}{sec}$. The expression for memory bandwidth requirements for ROHC compression processing relative to link speed becomes:

$$MemBW = [2 + \frac{264 \times (1+\beta)}{P+H} + \frac{2 \times (P+\beta H)}{P+H}] \times R\frac{bytes}{sec} \tag{24}$$

Note that the memory bandwidth scaling factor relative to the link rate is now a function of the header size, payload size, and β. A plot of the memory bandwidth scaling factor versus payload size over the range of β values for ROHC compression of IPv6/UDP/RTP headers is given in Figure 15. Note that we assumed a static uncompressed header size, H, of 100 bytes.

Supporting our previous example of 20 byte voice datagrams with 100 byte IPv6/UDP/RTP headers

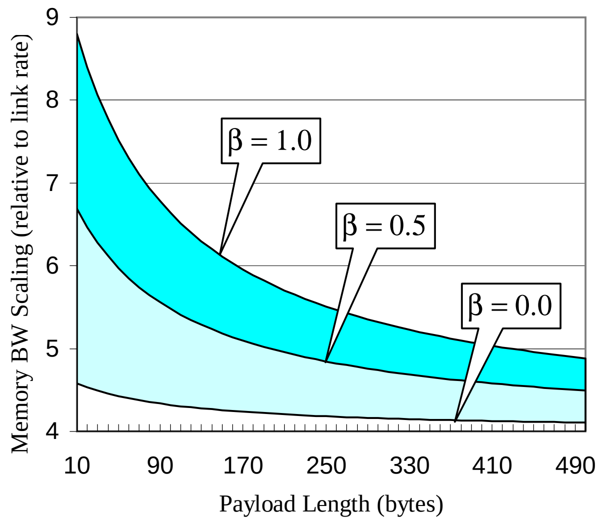

Figure 15: Memory bandwidth scaling factor relative to supported link rate versus packet payload size for ROHC compression of IPv6/UDP/RTP headers. Assumes static uncompressed header size of 100 bytes.

would require a total memory bandwidth of approximately 8.4 times the link rate, more than double the bandwidth required for normal packet processing. For a network processor supporting a bi-directional 2.5 Gb/s link or 5 Gb/s total throughput, this implies that the memory interface be dimensioned for 42 Gb/s. Using a standard datapath width of 128 bits, this implies that the interace must be operated at over 328 MHz for standard memory technologies or 164 MHz for Dual Data Rate (DDR) memory technologies [16, 17]. Note that we have assumed simplistic packet handling and application of ROHC compression to 100% of the link traffic. Our results imply that ROHC processing has a significant cost in terms of off-chip memory bandwidth consumption. As link speed increases continue to outpace memory technology improvements, optimized fast-path or header-only processing techniques may become essential to meeting throughput constraints especially when supporting additional memory-intensive applications.

5.7 Processing Resources

The following analysis seeks to assess the impact of ROHC given the capacity of current network processors, as well as provide motivation for the architectural enhancements for supporting ROHC in next-generation network processors which are discussed in Section 6. Due to the complexity of the ROHC standard and lack of open implementations at this time, we construct reasonable estimates of the required processing resources based on publicly available data from initial header compression development efforts. It must be emphasized that we only seek to establish some intuition regarding processing resource consumption and scaling in order to facilitate our architectural discussion.

We begin by establishing bounds on the number of instruction cycles required for compression and decompression in purely software-based implementations. Starting with a lower bound, we analyze the performance results reported in [5] for the comparatively simple TCP/IP header compression scheme. Based on the processing times given for the various test platforms, we estimate that compression requires approximately 460 cycles per packet while decompression requires approximately 200 cycles per packet.

In order to formulate an upper bound, we analyze results reported at a recent IETF meeting for an initial ROHC implementation by Roke Manor Research [3]. Running on a 270MHz *SUN Ultra 5* workstation, header compression required 41μs per packet while header decompression required 56μs per packet. Given that the *SUN Ultra 5* is equipped with an *UltraSPARC IIi* processor which is a 4-way SuperScalar architecture [18][19], we assume that efficiency ranges from one instruction per cycle to four instructions per cycle. Based on these assumptions we extrapolate that in this first-pass version of ROHC, compression requires between 11,070 instructions per packet and 44,280 instructions per packet while decompression requires between 15,120 instructions per packet and 60,480 instructions per packet.

We believe that these estimates must be further refined within the context of a network processor. Given the absence of an operating system, efficiencies achievable via code optimizations, and the offloading of packet classification to dedicated hardware in a network processor, we believe that the performance of ROHC software could be improved by an order of magnitude. Making this admittedly optimistic assumption, we will use the estimate of 1,500 instructions per packet for the remainder of the discussion. Given that this estimate is approximately three times higher than the requirements for TCP/IP header compression, we do not feel that it is unrealistic.

We now compare this processing resource consumption estimate to the capacity claims of current generation network processors from major producers targeting the OC-48 (2.5Gb/s) access market [20][21][22]. Due to the variety of processor architectures and varying degrees to which the processors offload tasks to on-chip hardware assists, we find that the total MIPS provided by current generation products ranges from 2,128 MIPS to 5,400 MIPS. Assuming these products must process orthogonal ingress and egress flows, we assume a total throughput constraint of 5Gb/s. Based on this assumption, we extract a budget range of 3.4 to 8.6 instructions per byte of link traffic. For our 120 byte IP/UDP/RTP voice packet example, the processing resources budgets for these products range from 408 to 1032 instructions per packet.

Comparing these capacities with our estimates of ROHC processing requirements, we find that application demands may be more than double the processor capacity. Even if we make the assumption that the next generation of these products will possess double the current processing capacity, ROHC could very well consume all of the processing resources leaving no cycles available for mandatory packet processing or additional service applications. Clearly, ROHC serves as another example of the need for more processing resources in network processors. In order to meet this demand, system architects could simply increase the number of parallel processors in the system. We now investigate the more intriguing architectural question of how to accommodate ROHC in next-generation network processors via the use of application specific hardware assists.

6 Offloading ROHC Functions to Application-Specifi c Hardware Assists

A significant portion of the implementation complexity of ROHC lies in the functionality and interactions of the Compressor and Decompressor Finite State Machines (CFSM/DFSM) and the Mode Finite State Machine (MFSM) that establish the context information, encoding parameters, and packet formats. The software-programmable processing engines in network processors are well-suited for such tasks, suggesting a hardware-software co-design to achieve a favorable balance among flexibility and performance. In this section, we consider several of the computationally intensive and redundant tasks specific to ROHC as candidates for implementation as hardware assists. In addition to absolute processing times, the results reported in [3] also provide a breakdown of the amount of time consumed by various steps in the compression process. Correlating these results to the logical block diagram presented in Figure 6, we make the following observations:

- Packet classification and context binding comprises approximately 14% of the workload
- Packet parsing comprises approximately one third of the workload

- CRC computations comprise 14% to 20% of the workload
- Header field encoding and decoding comprises approximately one third of the workload

As previously discussed, the contributions due to packet parsing, packet classification, and CRC computations may be offloaded to existing hardware assists in current-generation network processor platforms. Offloading the encoding and decoding of header fields allows a majority of the ROHC processing to be done in hardware assists. We begin our analysis of ROHC hardware assists by examining the achievable performance and die area required for Header Field Codecs in several implementation technologies with varying degrees of flexibility. In order to evaluate the implications of a hardware-software co-design, we continue with a discussion of the interconnection bandwidth consumption and hardware-software handovers in Section 6.2. Although we do not provide an explicit analysis, we do believe that reconfigurable hardware assists provide a viable high-performance option for the finite state machines of the Context Processor if additional offloading were required to achieve a specific performance target. As in the previous sections, we focus our analysis on supporting ROHC for IP/UDP/RTP headers.

6.1 Performance, Flexibility, & Size

In light of our previous analysis of ROHC header field encoding and decoding techniques, we now consider implementation options for a set of ROHC Header Field Codecs hardware assists. Note that we consider two general codec designs: *generic* codecs which require that the shifting parameters be passed with the input values and *field-specific* codecs for each header field which are optimized for the known shifting parameters and field width. Due to the nature of WLSB encoding, all of the encoders may be designed in an *iterative* fashion which seeks to maximize logic reuse and minimize area, or they may designed in a *pipelined* fashion which computes all intermediate results in parallel and allows a new set of fields to be issued every clock cycle. We implemented the generic and field-specific encoders in both paradigms in order to effectively explore the design space.

We note that given the current lack of insight into ROHC behavior, flexibility is essential for initial implementations. As more experience is garnered, less-flexible implementations providing higher performance may become favorable in the future. We consider three implementation options that represent a likely migration path for ROHC hardware assists. Regarding performance constraints, our analysis assumes a 2.5 Gb/s bi-directional link; therefore, the hardware assists must provide for 2.6 million header encoding operations per second and 2.6 million header decoding operations per second. An operation refers to the complete encoding or decoding of all the dynamic header fields of the packet. In order for the generic codecs to meet the throughput constraint, they must provide 7.8 million operations per second (Mops) as they must operate on all three of the dynamic header fields in the IP/UDP/RTP stack. For ease in comparison, performance results are listed with millions of packets per second (Mpkts).

6.1.1 Reconfigurable Hardware Assists

By augmenting a network processor with Reconfigurable Hardware Assists (RHAs), flexible implementations of computationally intensive and profile-specific functions may be realized. In order to assess the achievable performance and size of ROHC Header Field Codecs implemented as RHAs, we designed and implemented several codecs in VHDL. The designs were then synthesized using Synplicity's Synplify Pro targeting a Xilinx Virtex-E device. Achievable clock frequencies are based on worst-case delay estimates in a device with a -8 speed grade. Based on the figures claimed in the Xilinx Virtex-E datasheet [23], a single LUT/FF pair translates to 13.5 equivalent ASIC gates. A recent announcement by IBM and Xilinx claims

Table 6: Reconfigurable hardware resource usage and performance estimates for ROHC IP/UDP/RTP encoding methods. LUT/FF usage and clock period estimates according to synthesis with Synplicity Pro targeting a Xilinx Virtex-E (-8) device using worst-case delay estimates. Area estimates assume 423 LUT/FF pairs per mm^2.

Block	cycles/op	LUTs	FFs	Area mm^2	T_{clk} ns	Tput $Mops$	Tput $Mpkts$
Generic LSB Encoder (*Pipelined*)	1	1077	273	2.546	10.107	98.941	32.908
Generic LSB Encoder (*Iterative*)	2	694	297	1.641	9.749	51.287	17.096
Generic WLSB Encoder (*Pipelined*)	1	2250	603	5.319	10.057	99.433	33.144
Generic WLSB Encoder (*Iterative*)	4	714	417	1.688	9.77	25.589	8.530
IP-ID Encoder (*Pipelined*)	1	336	187	0.794	7.465	133.958	133.958
IP-ID Encoder (*Iterative*)	2	193	165	0.456	7.447	67.141	67.141
RTP SN Encoder (*Pipelined*)	1	559	221	1.322	7.547	132.503	132.503
RTP SN Encoder (*Iterative*)	4	198	151	0.468	7.379	33.880	33.880
RTP Scaled TS Encoder (*Pipelined*)	1	2279	590	5.388	10.181	98.222	98.222
RTP Scaled TS Encoder (*Iterative*)	4	727	386	1.719	9.749	25.644	25.644
Generic LSB Decoder	1	647	210	1.530	19.036	52.532	17.511
IP-ID Decoder	1	188	166	0.444	12.203	81.947	81.947
RTP SN Decoder	1	248	155	0.586	12.082	82.768	82.768
RTP Scaled TS Decoder	1	621	200	1.468	19.407	51.528	51.528

that forthcoming embedded Xilinx FPGA cores of 40k equivalent gates will require 7 mm^2 in the IBM Cu-08 process [24]. We therefore use the estimate of 423 LUT/FF pairs per mm^2 for our area estimate. Our implementation results are shown in Table 6.

A fairly large degree of size and speed optimization is achievable in FPGA technology via low-level description and hand-placement of designs; however, our analysis focuses not on determination of absolute performance, but extraction of relative trends between codec designs. Figure 16 illustrates the throughput and area tradeoffs of employing different types of codecs. In general, *iterative* and *generic* codec designs are more area efficient while a set of *field-specific* and *pipelined* codec designs provided better throughput. For our example case of supporting 2.5 Gb/s links, we note that a combination of an *iterative, generic* WLSB encoder and *generic* LSB Decoder exceeds the necessary throughput of 2.6 million packets per second with better area efficiency than use of field-specific encoders. Specifically, a set of generic codecs provides a worst-case throughput of 8.5 million packets per second while consuming approximately 3.218 mm^2 of die space. A set of *pipelined, field-specific* encoders provide the highest worst-case throughput at 98.2 million packets per second, but require 7.5 mm^2 of die area. Likewise, the collection of *field-specific* decoders which collectively would 2.5mm^2 of die space provide the highest worst-case throughput of 51.5 million packets per second.

For a network processor employing generic codecs or supporting a single ROHC profile, reconfiguration may be relatively infrequent implying that manually triggered reconfiguration of RHAs would be sufficient. However for network processors supporting multiple ROHC profiles or higher link rates requiring the use of profile-specific codec designs, run-time reconfiguration mechanisms would be required incurring an additional hardware cost. We defer discussion of such mechanisms at this time.

Based on our preliminary implementation results, we also predict that implementation of a full ROHC Compressor/Decompressor as an RHA would likely require more die area than available on-chip in the announced embedded FPGA cores. As additional service applications such as content filtering and transcoding

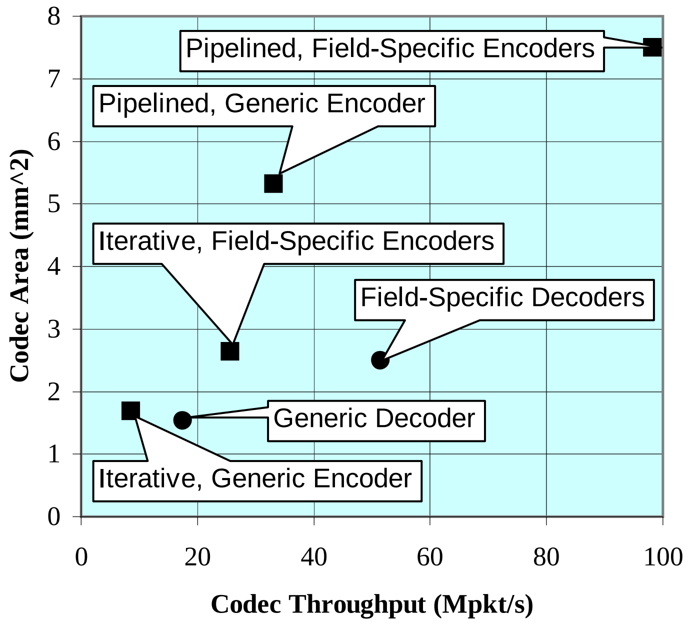

Figure 16: Area versus throughput tradeoff for various ROHC header field codec designs implemented as Reconfigurable Hardware Assists (RHAs) in embedded FPGA technology.

are pushed to the network edge, fully offloading ROHC processing may become desirable. In such a case, the network processor platform could be augmented with an additional reconfigurable hardware processing element utilizing the Dynamic Hardware Plugins (DHP) architecture [25]. DHP was developed in order to exploit the high-performance and flexibility provided by reconfigurable hardware technology for high-speed network processing. By providing reconfigurable logic regions with static interfaces to on-chip I/O ports and arbitrated off-chip memory resources, DHPs allow computationally intensive functions and applications to be completely offloaded from the network processor without loss of flexibility.

6.1.2 Application Specific Instruction set Processors (ASIPs)

An ASIP is an alternative between GPPs and ASICs that combines the required flexibility with efficiency by designing a specialized processor core for a class of applications—in our case header compression—such that it is software programmable but its structure is optimized to speed up the common case. Interfaces to the environment of the ASIP, including memories, are tailored to the application class, and processor infrastructure that does not considerably contribute to processing performance is pruned off. This avoids logic that is only rarely used but consumes area and power.

A significant improvement over a GPP stems from a specialized instruction set that speeds up frequently used code sections and reduces program-memory requirements by implementing common combinations of operations in hardware as extended instructions. An example for an ASIP specialized in header parsing for 10 Gb/s is given in [15]. The size of the parser, including a small instruction memory, is in the order of $0.45mm^2$ in a $0.18\mu m$-process which demonstrates the area efficiency of the ASIP approach. Examples of methods for the derivation of ASIP instruction sets from application representations can be found in [26, 27].

An ASIP for header compression can employ instructions of different complexity. Single instructions can implement complete coding blocks, such as a WLSB encoder or an LSB decoder, or they can be more basic functions such as a parameterized version of Figure 9, that can be used to implement a variety of

Table 7: ASIC resource usage and performance estimates for ROHC IP/UDP/RTP encoding methods. Area and clock period estimates according to synthesis targeting IBM Cu-11 process with worst-case delay estimates.

Block	cycles/op	Area mm^2	T_{clk} ns	Tput $Mops$	Tput $Mpkts$
Generic LSB Encoder (*Pipelined, Area Optimized*)	1	0.036	7.9	126.582	42.194
Generic LSB Encoder (*Pipelined, Speed Optimized*)	1	0.106	2.03	492.611	164.204
Generic LSB Encoder (*Iterative, Area Optimized*)	2	0.033	6.6	75.758	25.253
Generic LSB Encoder (*Iterative, Speed Optimized*)	2	0.079	2.04	245.098	81.699
Generic WLSB Encoder (*Pipelined, Area Optimized*)	1	0.072	10.5	95.238	31.746
Generic WLSB Encoder (*Iterative, Area Optimized*)	4	0.035	9.1	27.473	9.158
Generic LSB Decoder (*Area Optimized*)	1	0.042	13.5	74.074	24.691
Generic LSB Decoder (*Speed Optimized*)	1	0.121	2.74	364.964	121.655

codecs. This is a trade-off between efficiency, defined as performance per area, and flexibility.

In the ROHC case, the instructions should represent the basic functions that make up a compression profile. When new compression profiles are specified they can be implemented with those basic ROHC functions and be added to the ROHC ASIP's functionality. If the ASIP is supposed to only run ROHC then the framework around the profiles, which does not change, can even be implemented in a hard-wired fashion. This solution combines the flexibility for new profiles with the most efficient implementation of the framework.

If the goal is to also be open for future header compression schemes beyond ROHC then more flexibility can be provided by adding instruction-set support for header-compression frameworks in general and implementing the ROHC framework in ASIP software, thus shifting the trade-off between flexibility and efficiency towards a less efficient but more generic implementation.

6.1.3 Application-Specific Integrated Circuits (ASICs)

As the ROHC specification matures and becomes more stable, flexibility may no longer be a necessity and performance may become a higher priority. In such a case, Application-Specific Integrated Circuits (ASICs) may be the prefered implementation technology for ROHC hardware assists as they provide high-performance at low area cost for fixed functions. Results from synthesis of select ROHC codecs targeting the IBM Cu-11 process with worst-case delay estimates are shown in Table 7. Note that design tools for ASIC synthesis provide for a wide range of optimization parameters. We provide results for both area and speed optimized synthesis runs in order to illustrate the spectrum of achievable results. Based on these results, a set of generic codecs capable of performing 75 million encodes and decodes per second would require less than 0.1 mm^2 of die area.

6.2 Interconnection Architecture

Hardware assists reduce processor cycle consumption at the cost of additional overhead for communication between processor and hardware assists. It is our aim to derive a first-order approximation of the additional interconnection bandwidth required for software/hardware handovers for ROHC hardware assists. This approximation will aid our discussion of interconnection architectures suitable for ROHC hardware assists, including placement and coupling to the processor.

For the purpose of our analysis, we assume a generic processor architecture as shown in Figure 11 and that the following steps contribute to interconnection bandwidth consumption:

1. Packet receive (TI/SAR to SDRAM interface): $P + H$
2. Packet load (SDRAM Interface to processor cache): $P + H$
3. Context load (SDRAM Interface to processor cache): C_c
4. Vector to hardware assists: HWA_i
5. Return from hardware assists: HWA_j
6. Context store (Processor cache to SDRAM interface): βC_c
7. Packet store (Processor cache to SDRAM interface): $P + \beta H$
8. Packet transmit (SDRAM Interface to TI/SAR): $P + \beta H$

The quantities following each step refer to the amount of data per packet which must transit the interconnect. Note that under these assumptions, interconnection bandwidth usage is equal to memory bandwidth usage when no hardware assists are employed. Clearly, these assumptions do not take into consideration the overhead incurred for interconnect transactions. For example, in high-performance bus-based interconnects several cycles are consumed for arbitration and addressing. We assume that such overheads may be accounted for by a general additive constant.

The values of HWA_i, the size of arguments passed to the hardware assists, and HWA_j, the size of results returned to the processor, depend on the type of ROHC hardware assists employed in the system. We examine the case of ROHC encoding employing a hardware assists for generic WLSB encoding, CRC calculations, MFSM and CFSM state transitions. Based on our previous assumptions, the total amount of information transferred, $HWA_i + HWA_j$, is approximately 85 bytes. Given that the amount of data per packet passed between the processor and hardware assists is a constant, the amount of additional interconnection bandwidth required for these transactions relative to the supported link rate decreases with the packet size. A plot of the additional interconnection bandwidth requirement relative to the supported link rate versus packet payload size for ROHC compression of IPv6/UDP/RTP headers with hardware assists is shown in Figure 17. Additional interconnection bandwidth for decompression is shown in Figure 18.

Relative to the interconnection bandwidth consumed by packet and context load and stores, the additional amount required for ROHC hardware assists is small. For our 20 byte voice datagram example an additional $0.7 \times R\frac{bytes}{sec}$ of interconnection bandwidth is required. These results imply that ROHC hardware assists may be loosely coupled to the packet processor and utilize standard interfaces to on-chip interconnect. If addressing and arbitration overheads are sufficiently large, hierarchical interconnect may be used to eliminate the contributions of hardware assist communication from the $FastPathInterconnect$ as shown in Figure 11.

7 Conclusions

We have provided an overview of Robust Header Compression (ROHC) and extracted the primary functional blocks required for an ROHC Compressor/Decompressor. From this analysis we examined the architectural implications imposed by ROHC on network processors designed for use in wireless access infrastructure

Table 8: Accounting of bytes per packet required for communication between processor and ROHC hardware assists.

Block	Input Description	HWA_i (bytes)	Output Description	HWA_j (bytes)	Total (bytes)
CFSM/DFSM	transition parameters	20	win update	1	
	type/mode/state	1	$count$ update	1	
	SN, ACK/update flags	3	$timeout$ update	2	28
MFSM	mode/trans state	1	mode/trans update	1	
	ACK/update flags	1	ACK type	1	4
Generic WLSB Encoder	$v, v_{ref_{min}}, v_{ref_{max}}$	3×12	$k, flags$	3×2	
	a, b, c	3×3			51
Generic LSB Decoder	v_{ref}	3×4	$v, flags$	3×5	
	m, a, b, c	3×2			
	m_{val}	3×4			45
IP-ID Encoder	$v, v_{ref_{min}}, v_{ref_{max}}$	6	$k, flags$	2	
RTPSN Encoder	$v, v_{ref_{min}}, v_{ref_{max}}$	6	$k, flags$	2	
RTP TS Encoder	$v, v_{ref_{min}}, v_{ref_{max}}$	12	$k, flags$	2	30
IP-ID Decoder	v_{ref}, m_{val}, m	5	$val, flags$	3	
RTPSN Decoder	v_{ref}, m_{val}, m	5	$val, flags$	3	
RTP TS Decoder	v_{ref}, m_{val}, m	10	$val, flags$	5	31
CRC	received	1	result	1	2

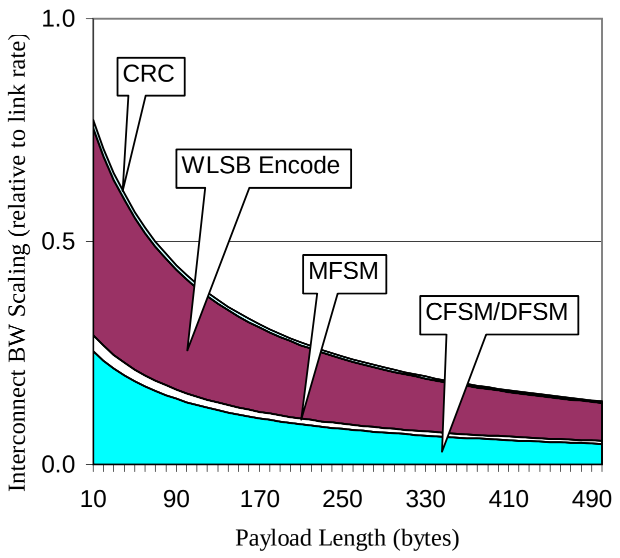

Figure 17: Additional interconnect bandwidth requirement relative to supported link rate versus packet payload size for ROHC compression of IPv6/UDP/RTP headers with hardware assists. Assumes hardware assists implementing CFSM/DFSM, MFSM, *Generic* WLSB Encoder, and CRC. Assumes static uncompressed header size of 100 bytes and excludes contributions for packet classification, frame CRC, and interconnect overheads such as arbitration and addressing.

such as Base Station Subsystems (BSS). Based on reasonable assumptions regarding the probable environment of use we provided an estimate for the amount of memory required to store context information.

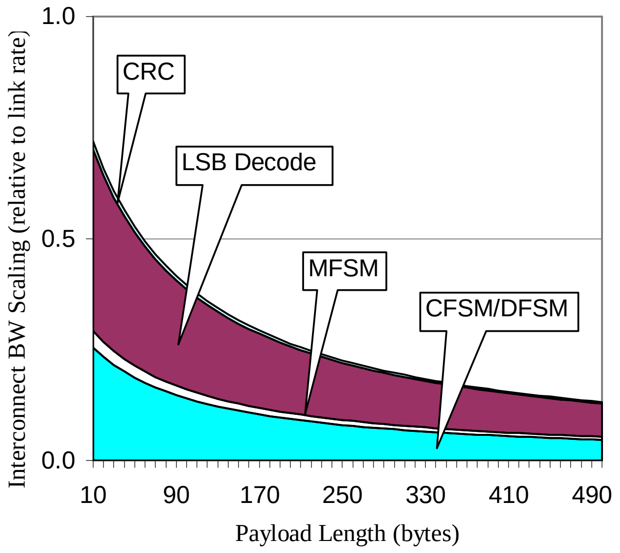

Figure 18: Interconnect bandwidth scaling factor relative to supported link rate versus packet payload size for ROHC decompression of IPv6/UDP/RTP headers. Assumes hardware assists implementing CFSM/DFSM, MFSM, *Generic* LSB Decoder, and CRC. Assumes static uncompressed header size of 100 bytes and excludes contributions for packet classification, frame CRC, and interconnect overheads such as arbitration and addressing.

Assuming a context size of 264 bytes per context and a network processor supporting thousands of flows implies that off-chip memory must be used for context storage. We then analyzed the required memory bandwidth relative to the supported link rate and found that for small payload sizes ROHC requires a memory interface providing a bandwidth of up to nine times the link rate. This is a significant result given standard industry practice of dimensioning network processor aggregate memory bandwidths equal to the link rate or at most four times the link rate.

Based on initial implementation results and capacities of existing network processors, we argue that ROHC imposes a significant strain on processing resources. We then explored the design space for application-specific hardware assists for ROHC in the form of reconfigurable hardware, Application-Specific Instruction-set Processors (ASIPs), and Application-Specific Integrated Circuits (ASICs). We showed that the additional interconnection bandwidth required for software/hardware handovers is relatively small implying that hardware assists could be loosely coupled to the packet processor and employ standard interfaces to on-chip interconnect. We assert that supporting ROHC in next-generation network processors targeted to large aggregation nodes in wireless access networks requires consideration at the architectural level. We also provide evidence for continued incorporation of application-specific hardware assists in high-performance network processor architectures.

Acknowledgments

We would like to thank Mark West for his prompt replies to our questions regarding his slides presented at the ROHC Working Group meeting of the IETF. We also would like to thank Peter Buchmann for his assistance with CAD tool flows for the ASIC hardware assists evaluation.

References

[1] Effnet, "An Introduction to EffnetEdge Header Compression Technology," tech. rep., Effnet Inc., 888 Villa Street, Mountain View, CA 94041, USA, 2002.

[2] C. Bormann, et. al., "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed." RFC 3095, July 2001. IETF Network Working Group.

[3] M. West, "ROHC Implementation Experience," in *ROHC Working Group Meeting*, 50th IETF Meeting, March 2001.

[4] H. Schulzrinne, et. al., "RTP: A Transport Protocol for Real-Time Applications." RFC 1889, January 1996. IETF Network Working Group.

[5] V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links." RFC 1144, February 1990. IETF Network Working Group.

[6] M. Degermark, B. Nordgren, and S. Pink, "IP Header Compression." RFC 2507, February 1999. IETF Network Working Group.

[7] S. Casner and V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links." RFC 2508, February 1999. IETF Network Working Group.

[8] L.-E. Jonsson, M. Degermark, H. Hannu, and K. Svanbro, "RObust Checksum-based header COmpression (ROCCO)." Internet Draft, June 1999. IETF Network Working Group.

[9] L.-A. Larzon, M. Degermark, and S. Pink, "Efficient Use of Wireless Bandwidth for Multimedia Applications," tech. rep., Lulea University of Technology, 2000.

[10] L.-A. Larzon, H. Hannu, L.-E. Jonsson, and K. Svanbro, "Efficient Transport of Voice over IP over Cellular links," in *Globecom*, 2000.

[11] M. Thoren, "Ericsson compression technology boosts 3G network capacity." Ericsson Press Release, May 2002.

[12] L.-E. Jonsson, "RObust Header Compression (ROHC): Requirements and Assumptions for 0-byte IP/UDP/RTP Compression." RFC 3243, April 2002. IETF Network Working Group.

[13] L.-E. Jonsson and G. Pelletier, "RObust Header Compression (ROHC): A Link-Layer Assisted Profile for IP/UDP/RTP." RFC 3242, April 2002. IETF Network Working Group.

[14] C. Bormann, "RObust Header Compression (ROHC) over PPP." RFC 3241, April 2002. IETF Network Working Group.

[15] G. Dittmann, "Programmable finite state machines for high-speed communication components," Master's thesis, Darmstadt University of Technology, `http://www.zurich.ibm.com/~ged/HeaderParser_Dittmann.pdf`, 2000.

[16] Micron Technology Inc., "36Mb DDR SIO SRAM 2-Word Burst." Datasheet, December 2002.

[17] Micron Technology Inc., "256Mb Double Data Rate (DDR) SDRAM." Datasheet, October 2002.

[18] SUN, "Ultra 5 Workstation Hardware Specifications." `http://sunsolve.sun.com/handbook_pub/Systems/U5/spec.html`. SUN Microsystems.

[19] SUN, "UltraSPARC-IIi Processor." `http://www.sun.com/processors/UltraSPARC-IIi/specs.html`. SUN Microsystems.

[20] Intel, "IXP2400 Network Processor," tech. rep., Intel Corporation, 2002. Product Brief.

[21] Motorola, "C-5e Network Processor Silicon Revision A0," Tech. Rep. C5ENPDATA-DS/D, Motorola, Inc., 2002. Revision 1.

[22] IBM, "PowerNP NP4GS3 Network Processor," Tech. Rep. np3_ds_TOC.fm.10, Internationl Business Machines Corporation, 2002.

[23] Xilinx Inc., "Virtex-E 1.8V Field Programmable Gate Arrays." Xilinx Datasheet (DS022), 2001.

[24] IBM and Xilinx, "IBM, Xilinx shake up art of chip design with new custom product." Press Release, June 2002. http://www-3.ibm.com/chips/products/asics/products/cores/efpga.html.

[25] D. E. Taylor, J. S. Turner, J. W. Lockwood, and E. L. Horta, "Dynamic Hardware Plugins (DHP): Exploiting Reconfigurable Hardware for High-Performance Programmable Routers," *Computer Networks*, vol. 38, pp. 295–310, February 2002. Elsevier Science.

[26] M. Arnold and H. Corporaal, "Designing domain-specific processors," in *Proceedings of the Ninth International Symposium on Hardware/Software Codesign (CODES'01)*, pp. 61–66, April 2001.

[27] I.-J. Huang and A. M. Despain, "Generating instruction sets and microarchitectures from applications," in *Proceedings of the International Conference on Computer Aided Design (ICCAD-94)*, pp. 391–396, November 1994.

Appendix

For the purpose of our analysis, we must make several assumptions in order to estimate the amount of memory required per ROHC context. Note that many header fields may either be inferred or reconstructed; therefore, they need not be stored on a per-context basis. Specifically,

- Payload length fields are inferred from link layer frame lengths
- IPv4 options or padding are not allowed
- UDP checksum is either disabled or transmitted explicitly when enabled

For the following analysis we do not consider the requirements for the "negative caching" technique described in [2] for rapid error recovery.

In order to estimate the size of sliding windows maintained by the compressor for WLSB encoded fields, we assume an RTP flow for audio frames of 20ms duration transmitted over cellular links with an RTT of 200ms. Assuming normal operating conditions, there can be as many as 10 outstanding packets from the perspective of the compressor; therefore, the compressor will have to store 10 references values in its sliding window. WLSB encoding p parameters a, b, and c are assumed to be one byte each. Local arrival timestamps and variables TS_STRIDE, TIME_STRIDE, TS_OFFSET, Max_Jitter_BC, and Max_Jitter_CD for RTP Timestamp encoding are assumed to be four bytes each. Note that we are assuming a worst-case circumstance with one context per channel; therefore, channel jitter variables must be stored with each context.