# An FPGA-based Network Processor for IP Packet Compression

**Dan Munteanu** **Carey Williamson**
**Department of Computer Science, University of Calgary**
**Calgary, AB, Canada T2N 1N4**
**Email:** {munteanu,carey}@cpsc.ucalgary.ca

**Abstract**

This paper describes the design, implementation, and experimental evaluation of a reconfigurable network processor that can do on-the-fly content adaptation of IP packets for wired or wireless networks. In our demonstration application, FPGA technology is used in conjunction with traditional RISC microprocessors to perform IP packet compression in hardware, using a CAM-based hardware implementation of the Lempel-Ziv (LZ) compression algorithm. The experimental evaluation considers HTTP Web browsing traffic using the TCP/IP protocols. The measurement results show that the proposed LZ compression architecture can reduce network byte traffic volume by 5-38%. Furthermore, the hardware-based approach provides consistent throughput performance as the compression buffer size is increased.

**Keywords**: Network processor, FPGA, Lempel-Ziv compression, Measurement, Web performance

## 1 Introduction

Computer networking is one of the most dynamic fields in computer science. Recent technological advances such as Digital Subscriber Line (DSL) and Gigabit Ethernet have dramatically increased the data rates for wired computer networks, which in turn have enabled rich media content delivery to the users.

There is also growing user demand for mobile, small form-factor computing devices that are connected to the Internet using wireless technology. This technology takes several forms, including cellular data networks, infrastructure-based wireless LANs, mobile ad hoc networks, and hybrid wireless mesh networks. Despite wireless bandwidth limitations, the users of such technology want Internet content delivery similar in quality to that they experience on wired computer networks.

An emerging solution for satisfying these user demands is Web content adaptation. Nodes placed at the edges of the network, or at boundary points between heterogenous networking technologies, are responsible for transforming Web content suitably to meet constraints imposed by end-user devices regarding transmission bandwidth, storage capacity, battery power, or display capabilities. In this paper, we focus primarily on the transmission bandwidth issue, in the context of wireless LAN or wireless mesh networks.

Most methods for throughput enhancement on wireless networks [8, 9, 10] focus on spectral bandwidth and channel coding issues. Less attention is paid to reducing the amount of data to be transferred over the network. However, reducing the amount of data to be transferred over the network can make more efficient use of the limited bandwidth provided by the wireless physical layer.

This paper presents the design, implementation, and experimental evaluation of an IP packet compression engine that uses a hardware-programmable network processor. The engine uses a hardware implementation of the Lempel-Ziv (LZ) compression/decompression algorithm [22]. The LZ algorithm is implemented in the hardware-programmable fabric as a coprocessor attached to a network processor system created using FPGA technology. Such a network processor could be deployed at the router nodes in a wireless mesh network, or at an access point in a wireless LAN.

The rest of the paper is organized as follows. Section 2 briefly describes related work on packet compression. Section 3 reviews the Lempel-Ziv compression algorithm. Section 4 presents the architecture of our compression engine. Section 5 describes the experimental setup for our tests, while Section 6 presents the experimental results. Finally, Section 7 concludes the paper.

## 2 Related Work

Our work is certainly not the first to consider packet compression. Most of the prior work falls into one of two main categories: header compression strategies for TCP/IP, and efficient hardware designs for LZ compression.

Early work by Jacobson [11] and others [1, 13] proposed TCP/IP header compression algorithms. Header compression is intended for low-bandwidth links, such as dialup serial lines and wireless links. In the common case, the predictability of a TCP/IP header can be exploited to reduce the header size from 40 bytes to 3-5 bytes. For interactive telnet traffic, which typically generates 41 byte packets, this is an important optimization. For bulk data transfer, which typically generates 1500 byte packets, this optimization is less useful, offering only 3% overall reduction. Torkelsson [19] reports that hardware header compression for RTP/UDP/IPv6 produces up to 65% bandwidth savings for half-rate voice packets. In these papers, the term compression refers only to packet headers: only the header values that change from the previous packet are sent.

Several architectures for the hardware implementation of the Lempel-Ziv compression algorithm [22] have been devised. Ranganathan [18] proposes a systolic architecture that performs the string matching function in linear time with respect to the number of symbols contained in the search window. More recently, Hwang [6] proposes a similar architecture, with differences in the implementation of processing elements and their interconnection. The proposed architecture improves the hardware costs and the ease of testing the proposed circuit.

Work by Lee and Yang [14, 21] proposes using a Content Addressable Memory (CAM) for the string-matching function. This approach performs string matching by full parallel search. It is faster than the implementation using systolic arrays, since only in the worst case does the execution take $n$ steps.

Saha [20] proposes a CAM-based approach for the LZW compression algorithm. The proposed implementation offloads the main CPU by using a CAM-based compression co-processor to compress data at the application layer prior to sending it over the network.

Work by Jung [12] explicitly considered on-the-fly packet compression for wireless networks. Our approach differs by using FPGAs to implement LZ compression on a network processor. A general discussion of design issues for network processors is provided in [2].

## 3 Lempel-Ziv Compression

This section provides a brief introduction to the Lempel-Ziv compression algorithm [22]. The concept behind LZ compression is the temporal locality of information. The compression process consists of keeping track of repeating patterns, and replacing each such occurrence with pointers to where they occurred earlier in the input stream of data. At the time of the decompression, each pointer is replaced with the already decoded data stream to which it points. The input stream is reconstructed exactly, since LZ provides lossless compression.

In the compression process, a buffer is used to store the $2n$ most recently received data elements. The buffer is separated into two sections, as shown in Figure 1. The *Window* part on the left has the already processed data elements, while the *Match* part on the right contains incoming data that needs to be compressed.

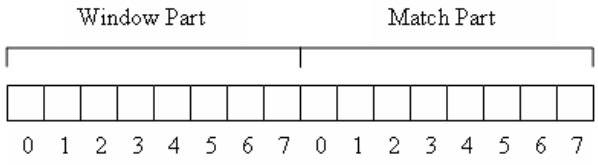


Figure 1. LZ Compression Buffer

The Window part of the buffer is initially empty (e.g., all nulls) at the beginning of the compression phase. The Match part is filled with the first $n$ characters to be compressed. The matching process follows the algorithm presented in Figure 2.

The variables *pointer* and *max_length* contain information about the start and the length of the repeating substring from the Window part of the buffer. The two variables along with the M[$max\_length + 1$] symbols from the Match part of the buffer form the complete description of the matched substring.

For example, assume a compression buffer length of 16 ($n = 8$), and an incoming string 'aabbaabbabbbb'. After the algorithm initialization, the compression buffer is '00000000 aabbaabb'. The algorithm in Figure 2 finds the longest match in the Window part of the buffer for the string starting at position 0 in the Match part of the buffer. In our specific example, the algorithm executes the `for` loops for $i$ and $j$ without finding any match between W[$index$] and M[$j$]. Therefore the values of *pointer* and *max_length* remain unchanged (zero). The last symbol (M[$max\_length + 1$] = 'a') is shifted into po-

```
max_length = 0; pointer = 0;
for ( i = 0 to n - 1 ) {
        curr_length = 0;
        index = i ;
        for ( j = 0 to  n – 2 ) {
                if (W[index] == M[j]) then curr_length ++;
                else break ;
                index ++ ;
                if (index >= n) break ;
        }
        if (curr_length > max_length) {
                max_length = curr_length;
                pointer = i;
        }
}
```

Figure 2. LZ Compression Algorithm

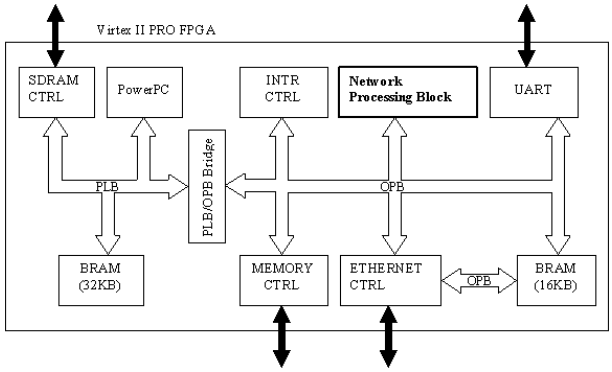


Figure 3. Network Processor Architecture

sition W[7], and all the other symbols in the match part of the buffer are shifted one position to the left. The location M[7] receives the value 'a' from the data section, so the compression buffer becomes '0000000a abbaabba'. The next iteration finds a match for $i = 7$, $index = 7$ and $j = 0$, so *pointer* is 7 and $max\_length$ is 1. The symbol that ends the matching (called the $last\_char$) is 'b'. All symbols in the processing buffer are shifted two positions to the left. The compression process continues step by step in this fashion until the entire input string has been processed.

## 4 IP Packet Compression Engine

### 4.1 Architectural Overview

The implemented network processor is built around a Memec 2pv7 development board. The network processor was implemented on a XC2PV7 FPGA device. The structure implemented in the FPGA circuit is presented in Figure 3.

The PowerPC processor and the SDRAM memory are used to execute software and the FPGA fabric is used to implement hardware blocks. Several IP cores were used in building the hardware structure in the FPGA. The Ethernet controller was an evaluation version of the 10 Mbps Memec Ethernet Controller. The CAM memories were generated using the Xilinx Coregen Utility. Also, the UART-lite serial controller from Xilinx was used. Depending on the compression buffer sizes used in the experiments, the FPGA usage ranged from 3,970 to 4,774 logic slices. The operating frequency of the system was 100 MHz.

A Linux operating system runs on the PowerPC processor [15]. The Ethernet frames are intercepted after the encapsulation of the IP packets and are sent to the compression engine. This mechanism ensures that only one IP packet at a time is compressed and sent in an Ethernet frame. Batch processing of packets could be considered in future work.

A byte stuffing method was devised to identify compressed IP packets on the network. The discussion here assumes Ethernet framing at the data link layer, though the concept applies more generally to other network technologies. The structure of the Ethernet frame before and after compression is presented in Figure 4.

The first two bytes in the Ethernet data section are both 0x00, and the compressed IP packet follows afterwards. The first two bytes in a normal IP packet represent the version of the IP protocol used and the length of the IP packet header. For the version byte, the value 0x00 is reserved [7], and the length of an IP header, represented in increments of 32-bit words, has a minimum value of 5. Thus, a machine receiving an Ethernet frame for which the first two bytes in the data section are 0x00 can determine that a compressed IP packet follows, decompressing it to restore the original packet. While we used this compression method only for IP packets, it can be used for other types of packets as well, as long as the compressed packets can be uniquely identified.

The compression engine works as a coprocessor for the PowerPC. It is accessed by the processor through registers mapped in the memory space of the system. The architecture of the Lempel-Ziv coder is presented in Figure 5.

The architecture used is CAM-based, similar to the one used by Lee and Yang [14, 21]. This architecture was chosen because of its high throughput per MHz, and low latency.

The processing steps implemented in the architec-

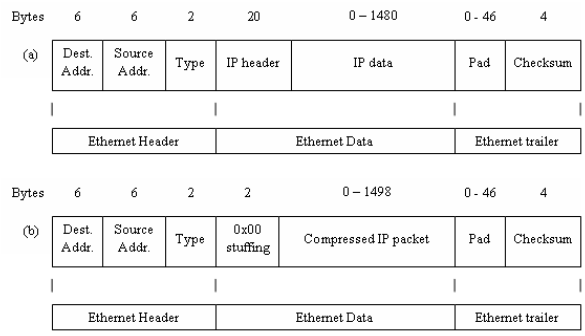


Figure 4. Byte Stuffing Method

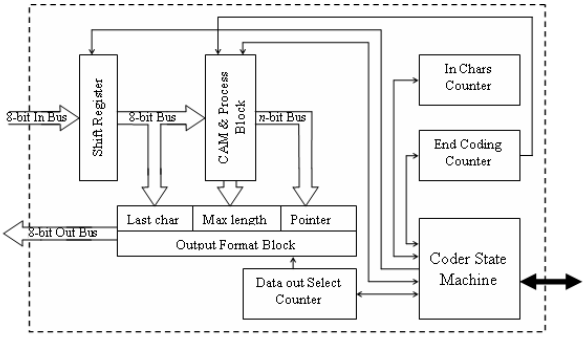


Figure 5. Lempel-Ziv Coder Architecture

ture are as follows. Data to be compressed comes into the shift register, which plays a role similar to the match part of the compression buffer in the software implementation. The CAM implements the window part of the compression buffer. As characters enter the shift register, they are compared with characters contained in the CAM. Additional pipelined processing verifies if consecutive matches occur in adjacent positions in the CAM. If the matches occur at consecutive positions, then *max_length* is incremented and a new shift operation occurs. If the matches do not occur at consecutive positions, or a match has not been found, the process stops and a triple (*pointer*, *max_length*, *last_char*) is generated and sent to the output format block for encoding.

## 4.2 Design Issues

There are two important design issues in our IP packet compression engine. The first is the compression buffer size, and the second is the encoding method used to record triples in the compressed format. The effectiveness of compression is affected by these design parameters, so our study considers them in some detail.

The compression buffer size determines how large a window of data the LZ algorithm has to work with for string matching. An additional constraint in our prototype implementation is that the LZ compression engine handles only one IP packet at a time. Given the limited length[1] of the data to be compressed, several values were investigated for the compression buffer length. Experiments were run with $n$ values of 64, 256, 512, and 1024 characters.

Three methods were explored for encoding the triple (*pointer*, *max_length*, *last_char*):

- **Fixed-Length Triples**. The first method uses a fixed-length representation for any triple. For a binary encoding of the parameters *pointer* and *max_length*, at least $log_2 n$ bits are needed to store each of the two parameters. The number of bytes needed to encode the triple is given by:

$$num\_bytes = 1 + \lceil (2(log_2 n)/8) \rceil$$

- **Mixed-Length Triples**. The second method treats the "no match" condition as a special case. That is, it checks if the *max_length* parameter is zero in the triple. If $max\_length > 0$, then the whole triple is encoded using the same approach as the first method. If $max\_length = 0$, then rather than encoding the entire triple, only the *last_char* value is written to the output. To distinguish between regular triples and "short triples", a special symbol is needed. We arbitrarily use the value 0x00 to precede the fully encoded triple. If a short triple ever has 0x00 as the *last_char*, it is written to the output preceded by another 0x00 value. This mechanism ensures proper recognition at decompression, since 0x00 cannot occur in two consecutive positions in a fully encoded triple.

- **Optimized Triples**. The third method improves upon the second method by observing that even for non-zero *max_length* values, the encoded triple may consume more space than the original data string that was matched. The third method outputs fully encoded triples only if the length of the full triple is smaller than the length of the string it encodes. If $max\_length < num\_bytes + 1$, then an array of *max_length* triples (0, 0, *last_char*) is encoded, with one entry for each *last_char* value in the matched substring.

All three methods have been implemented, tested, and evaluated in our experiments. In terms of hardware costs the first encoding method is the easiest and

[1]The default MTU in our implementation is 1500 bytes.

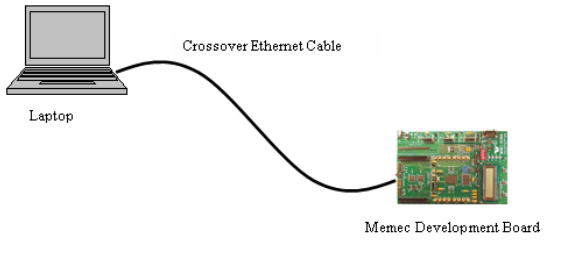


Figure 6. Experimental Setup

the most economic implementation. The second and third methods only slightly increase the logic utilization in the Output Format Block and add one more state to the Coder State Machine.

# 5 Experimental Methodology

## 5.1 Experimental Setup

The experimental evaluation of our prototype IP packet compression engine was carried out in a wired network environment, using a dedicated 10 Mbps Ethernet LAN. This network speed was considered appropriate for the testing of our prototype, given its clock rate (100 MHz).

A Dell Latitude laptop was used for running experiments. A crossover Ethernet cable was used to connect the laptop to the Memec development board implementing the network processor.

Figure 6 shows a diagram of the experimental setup. The laptop was used to download files from the network processor and to log network traffic information for analysis.

The laptop was running Red-Hat Linux operating system version 2.4.18. The Dell laptop had a 3Com 3c590 Ethernet card. The network driver on the laptop was modified to recognize compressed IP packets received from the network processor board. The network driver can also compress IP packets (in software) to send them to the network processor. The Maximum Transfer Unit (MTU) size used on both the laptop and the network processor was 1500 bytes.

## 5.2 Experimental Design

The two main system factors in the experiments are the compression buffer length and the encoding format used for triples. These factors were discussed in Section 4.2. Table 1 provides a summary of the experimental factors and levels.

Table 1. Experimental Factors and Levels

| Factor | Levels |
|---|---|
| Buffer Size | 64, 128, 256, 512, 1024 |
| Encoding | Fixed, Mixed, Optimized |

The performance metrics for the experiments are the compression ratio achieved (per-packet and average) and the effective throughput.

## 5.3 Web Workload

We use a Web workload in our experiments because HTTP traffic accounts for a large proportion of the traffic in today's wireless networks [17]. To generate different usage transfer scenarios, a Web server was installed on the network processor board used in the experiments. The Web server is based on the `fnord` Web server version 1.8 [3]. The Web server supports the HTTP/1.0 and HTTP/1.1 protocols, and it has a small memory footprint. The Web server accepts connections on TCP port 80.

Five Web pages[2] were used for the experiments with HTTP transfers. All of the chosen Web pages have embedded images, and some of them have Java scripts in their structure. The file types, sizes, and number of embedded objects are summarized in Table 2. Web pages 1, 2, and 5 are the opening pages for Web sites. These pages contain many embedded images. The HTML files for Web pages 3 and 4 are dominated by English prose.

# 6 Experimental Results

This section presents the results from our study. We first present the compression profile results for selected Web pages, followed by compression ratio results for different compression buffer lengths and encoding methods. The section concludes with a discussion of throughput results for hardware and software versions of LZ compression.

## 6.1 Compression Profile Results

The first experiment illustrates the per-packet compression results achieved for each Web page in the workload. We call these graphs "compression profile" graphs because they provide a time series representation of the compression achieved for each TCP/IP

[2] Additional experiments with other Web pages, FTP transfers, and a corpus of standard data compression tests are described in [16].

Table 2. Summary of Web Page Workload Used in Experiments

| Web Page ID | Web Site URL | HTML File Size (bytes) | Images | | Java Scripts | |
|---|---|---|---|---|---|---|
| | | | Total Num | Total Bytes | Total Num | Total Bytes |
| 1 | `cpsc.ucalgary.ca` | 25,859 | 42 | 83,174 | 2 | 117,951 |
| 2 | `www.yahoo.com` | 139,264 | 18 | 37,771 | 0 | 0 |
| 3 | `eetimes.com` | 142,672 | 93 | 169,315 | 0 | 0 |
| 4 | `ibm_linux_debug.html` | 60,709 | 8 | 2,421 | 7 | 7,751 |
| 5 | `pbs_stonehenge.html` | 71,503 | 30 | 28,127 | 2 | 10,225 |

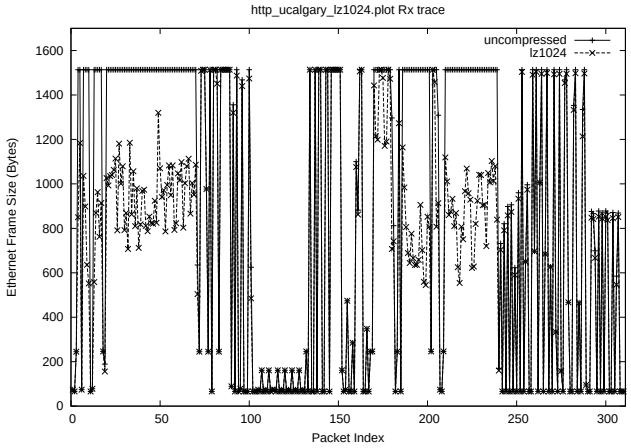


Figure 7. Compression Profile Results for `cpsc.ucalgary.ca` Web Page

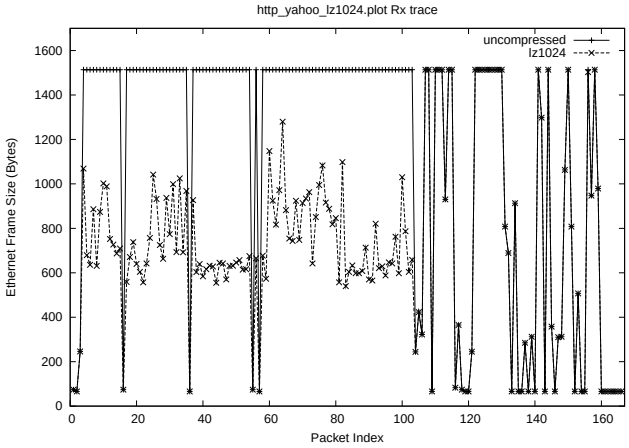


Figure 8. Compression Profile Results for `yahoo.com` Web Page

packet sent by the Web server over the Ethernet LAN. These graphs provide insight regarding which parts of the Web page are compressed effectively, and which are not.

Figure 7 and Figure 8 present the compression profile graphs for the `cpsc.ucalgary.ca` and `yahoo.com` Web pages. The horizontal axis of each graph shows the packet index in the (persistent) TCP connection for the Web page transfer from the server to the client. The vertical axis shows the size in bytes of the Ethernet frames observed. Lower values represent better compression performance. These plots show only the frames received by the laptop, since the frames sent by the laptop are typically small TCP acknowledgements. The top line (with '+' symbols) shows the sizes when no compression is used. The lower line (with 'X' symbols) shows the sizes when compression is used. The results presented here are for a compression buffer length of 1024 bytes and the Optimized Triples encoding format.

Figure 7 shows that good compression occurs in the first part of the transfer (e.g., packets 10-70), which represents the base HTML file and a few embedded files. The remaining objects in the Web page are mostly images, which do not compress well. Some of the images are very small (e.g., packets 100 to 130), and most are in a space-efficient compressed format already. When the compression engine tries to compress an IP packet containing an image (or part of an image), the length of the resulting packet often exceeds that of the original packet; in this case, the original (uncompressed) packet is sent instead. For some images, the lengths of the compressed packets exceed the network MTU size, making the compression impractical. Two objects that do compress well later in the Web page are the Java scripts (i.e., packets 180-200 and packets 210-240).

Similar observations apply for the `yahoo.com` Web page in Figure 8. The HTML page compresses well at the start of the transfer (e.g., packets 5-100). For the other Ethernet frames containing the GIF and JPEG embedded images, no compression is achieved since

they are already encoded in a space-efficient format.

Similar trends are observed for the other Web pages studied. Complete results appear in [16].

## 6.2 Compression Ratio Results

The second set of results compares the overall compression ratios for different compression buffer lengths and different encoding methods. These results are presented in Figure 9 through Figure 13. The horizontal axis of the graph shows the compression buffer length. The vertical axis shows the average compression achieved. The compression ratio is calculated relative to the cumulative byte count for Ethernet frames. The ratio expresses the relative reduction in byte traffic volume for Ethernet frames when compression is used. Higher values represent better compression performance.

Figure 9 shows the overall compression ratio results for the `cpsc.ucalgary.ca` Web page. A general trend observed is that the compression ratio tends to improve as the compression buffer size is increased. This result makes sense since the LZ algorithm has more data to work with at a time, and is more likely to find matches (and longer matches) in the strings processed. However, this trend is not universally true: there are some non-monotonic behaviours for the Fixed-Length triple encoding method. The exact compression ratio achieved can be highly sensitive to the size of a Web object, the data content, the HTTP response header, and the compression buffer length.

The overall compression ratio graphs in Figure 9 to Figure 13 show that the Fixed-Length encoding method for the triple (*pointer*, *max_length*, *last_char*) does not offer much advantage for Web page compression. The best compression ratio obtained with this method is 14% for the `cpsc.ucalgary.ca` Web page, using a compression buffer length of 256 characters. Typical results average about 5% improvement. However, there is no compression for the `yahoo.com` Web page (Figure 10) for any of the compression buffer lengths considered.

In general, the Mixed-Length encoding method provides better compression ratios. However, for the `cpsc.ucalgary.ca` Web page (Figure 9) and the `pbs_stonehenge` Web page (Figure 13), Mixed-Length encoding is worse than Fixed-Length encoding. This is due to the overhead introduced by the 0x00 stuffing on the fully encoded triples. For these Web pages, there are few triples with $max\_length = 0$, so adding one byte to each regular triple is costly.

The third encoding method (Optimized Triples) provides the best results in terms of compression ratio.

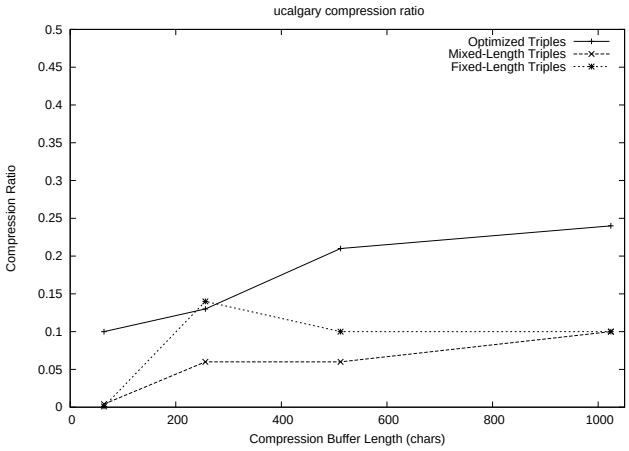


Figure 9. Relative Compression Results for `cpsc.ucalgary.ca` Web Page

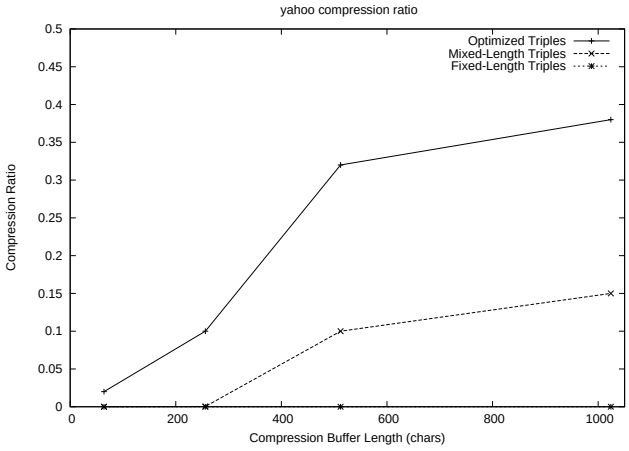


Figure 10. Relative Compression Results for `yahoo.com` Web Page

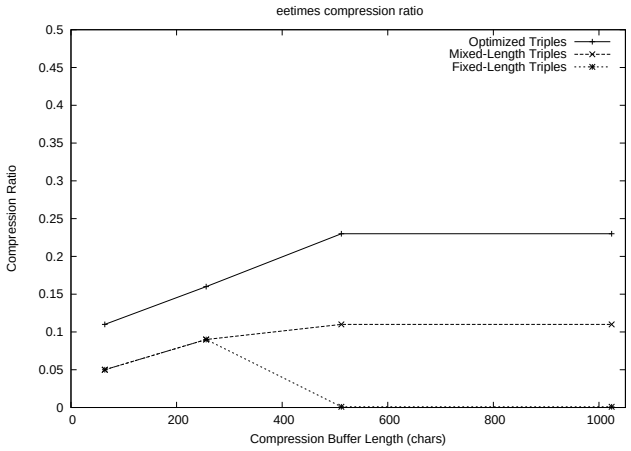


Figure 11. Relative Compression Results for `eetimes.com` Web Page

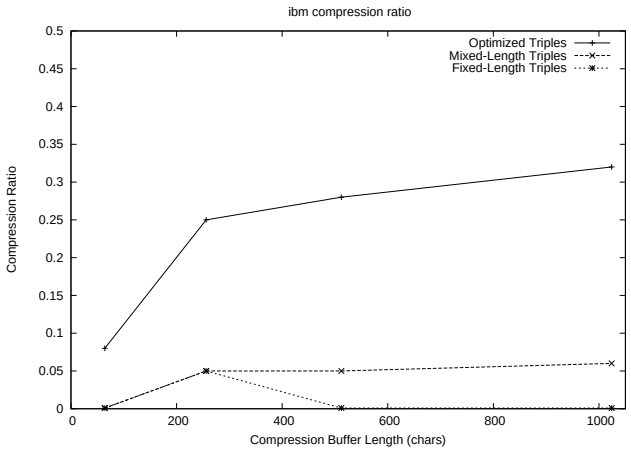


Figure 12. Relative Compression Results for `ibm_linux_debug` Web Page

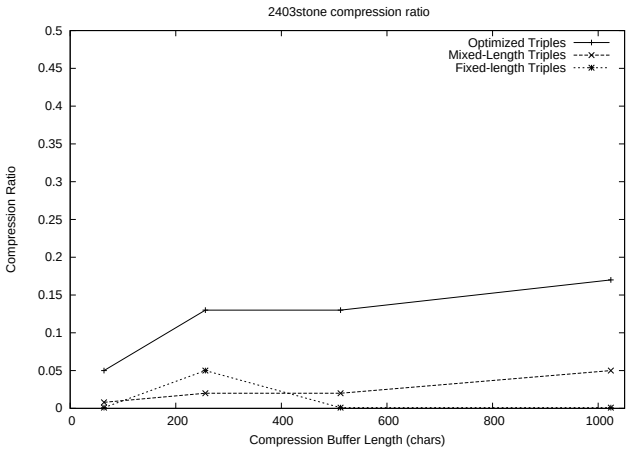


Figure 13. Relative Compression Results for `pbs_stonehenge` Web Page

The best results observed are 38% compression for the `yahoo.com` Web page. The typical compression ratio achieved ranges from 15-35%.

For the last two encoding methods, the best compression results are observed for a compression buffer length of 1024 characters. This buffer size exploits most (but not all) of the TCP/IP data carried in full Ethernet frames.

## 6.3 Latency and Throughput Results

Our final set of measurements quantifies the performance benefits of hardware-based compression compared to software-based compression.

The software implementation of the Lempel-Ziv compression algorithm has a computational complexity of $O(n^2)$, where $n$ is the compression buffer length in characters. The proposed hardware implementation, based on a CAM architecture [14, 21], has a complexity of $O(n)$.

The efficiency of both approaches was measured using FTP transfers of a large executable file (e.g., the Solaris loader `ld`, with a size of 2,730,604 bytes) with different compression buffer lengths. The performance metrics used are the packet latency and the user-level throughput reported by the FTP program. For each experiment, the uncompressed `ld` file was transferred using FTP.

The packet latency for the hardware compression process is reasonable. For a 1500-byte packet, and 100 MHz clock frequency for the compression block, the compression process adds at most 1.5 ms of delay, This delay is comparable to the packet transmission

time on an IEEE 802.11b wireless LAN, but low relative to typical round-trip latencies on the Internet. The observed delay reflects the design decisions made (i.e., interception of packets at the kernel level rather than building the compression engine as a pipeline block inside the Ethernet controller, and the use of FPGA technology with slower system speeds). An optimized system architecture and a faster implementation technology would reduce this delay, making the reported results relevant for other link-layer protocols such as Fast Ethernet or Gigabit Ethernet.

Figure 14 summarizes the throughput results. The graph shows throughput on the vertical axis versus the compression buffer length on the horizontal axis. Three lines appear on the graph. These represent the throughput for the hardware and software implementations of compression, as well as for performance of the system without compression. Higher values of throughput represent better system performance.

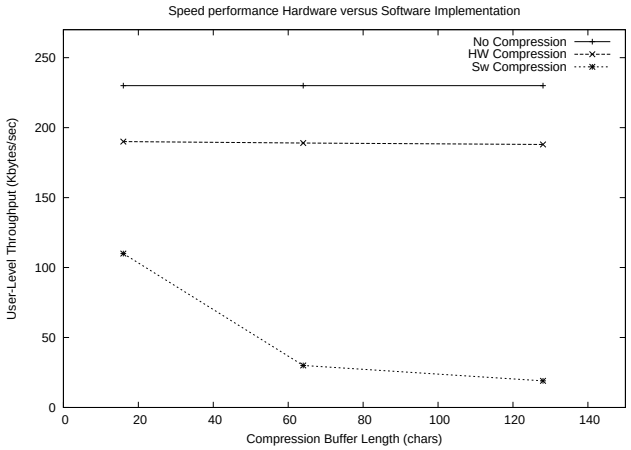


Figure 14. Throughput Performance Results

The generally low throughput results observed are attributable to the volume of (unoptimized) kernel instrumentation that was used in the experiments (i.e., recording information into the kernel logs on both the laptop and the development board). Statistical data are printed directly on the serial console.

Despite the instrumentation overhead, Figure 14 shows qualitative and quantitative differences between the hardware and software implementations of LZ compression. The throughput using the hardware implementation is 190 Kbytes/sec for a buffer length of 16 characters. The throughput for the hardware compression is 2-4 times higher than that for software compression, though the throughput for hardware compression is about 40 Kbytes/sec lower than that without compression.

Changing the compression buffer length has negligible impact on the throughput achieved by the hardware implementation. However, the software implementation suffers performance degradation when the compression buffer length is increased. This graph demonstrates the advantage of hardware-based compression versus software-based compression.

## 7 Conclusions

This paper presented the design, implementation, and evaluation of a compression mechanism for IP packets based on the hardware implementation of the Lempel-Ziv compression algorithm. Experiments with Web traffic were performed to understand the impacts of system parameters such as compression buffer length and encoding method on the overall compression performance.

The experimental results show that compression of IP packets for Web page transfers can reduce the volume of data sent over the physical medium. Compression ratios of up to 38% were observed for a compression buffer length of 1024 characters. The best results were obtained using the Optimized Triples encoding method for ($pointer$, $max\_length$, $last\_char$).

Our results suggest that the hardware compression mechanism is attractive for wireless network applications, where power consumption for transmission/reception of packets is a limiting factor. Although our experiments were performed on a wired network, the results are applicable to wireless networks as well since the proposed mechanism is not restricted by a particular choice of datalink layer protocol. Extending our prototype to wireless LAN operation requires additional hardware for a USB-based wireless interface [5].

Another conclusion of the experiments is that the hardware implementation of the Lempel-Ziv compression algorithm is more scalable than the software implementation. In particular, the throughput of the hardware-based implementation does not degrade when increasing the compression buffer length.

In addition to the wireless LAN extension, future work will explore other application domains for our network processor board. We believe that the network processor architecture is easily reconfigurable for other packet-level services such as Web content transcoding, packet monitoring, or network intrusion detection [4].

## Acknowledgements

Financial support for this research was provided by iCORE (Informatics Circle of Research Excellence) in

the Province of Alberta, as well as NSERC (Natural Sciences and Engineering Research Council) and CFI (Canada Foundation for Innovation). The authors are grateful to Nayden Markatchev for his technical support related to this work, and to Dr. Laurence Turner for his insightful comments about FPGA-based compression for wireless networks.

## References

[1] G. Boggia, P. Camarda, and V. Squeo, "ROHC+: A New Header Compression Scheme for TCP Streams in 3G Wireless Systems", *Proceedings of the IEEE International Conference on Communications*, Vol. 5, pp. 3271-3278, 2002.

[2] P. Crowley, M. Franklin, H. Hadimioglu, and P. Onufryk, "Network Processors: An Introduction to Design Issues", *Proceedings of the 8th International Symposium on High Performance Computing, Workshop on Network Processors*, Vol. 1, pp. 1-8, 2003.

[3] `http://www.fefe.de/fnord/`

[4] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. Hogsett, "Granidt: Towards Gigabit Rate Network Intrusion Detection Technology", *Proceedings of the 12th International Conference on Field-Programmable Logic and Applications*, September 2002.

[5] M. Gruteser, A. Jain, J. Deng, F. Zhao, and D. Grunwald, "Exploiting Physical Layer Power Control Mechanisms in IEEE 802.11b Network Interfaces", *Technical Report CU-CS-924-01*, Department of Computer Science, University of Colorado at Boulder, December 2001.

[6] S. Hwang and C. Wu, "Unified VLSI Systolic Array Design for LZ Data Compression", *IEEE Transactions on VLSI Systems*, Vol. 9, No. 4, pp. 489-499, August 2001.

[7] `www.iana.org/assignments/version-numbers`

[8] IEEE Standard 802.11a, IEEE 1999.

[9] IEEE Standard 802.11b, IEEE 1999.

[10] IEEE Standard 802.11g, IEEE 2003.

[11] V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links", *RFC1144*, 1990.

[12] B. Jung and W. Burleson, "Performance Optimization of Wireless Local Area Networks through VLSI Data Compression", *Wireless Networks*, Vol. 4, No. 1, pp. 27-39, 1998.

[13] K. Le, C. Clanton, Z. Liu, and H. Zheng, "Efficient and Robust Header Compression for Real-Time Services", *Proceedings of the Wireless Communications and Networking Conference*, Vol. 2, pp. 924-928, 2000.

[14] C. Lee and R. Yang, "High-throughput Data Compressor Designs Using Content Addressable Memory", *Proceedings of IEEE Circuits, Devices, and Systems*, Vol. 142, No. 1, pp. 69-73, February 1995.

[15] `http://www.mind.be`

[16] D. Munteanu, *A Hardware Programmable Network Processor*, M.Sc. Thesis, Department of Computer Science, University of Calgary, August 2004.

[17] C. Na, J. Chen, and T. Rappaport, "Hotspot Traffic Statistics and Throughput Models for Several Applications", *Proceedings of IEEE GLOBECOM*, pp. 3257-3263, December 2004.

[18] N. Ranganathan and S. Henriques, "High-Speed VLSI Designs for Lempel-Ziv-Based Data Compression", *IEEE Transactions on Circuits and Systems*, Vol. 4, No. 2, pp. 96-106, February 1993.

[19] K. Torkelsson and J. Ditmar, "Header Compression in Handel-C: an Internet Application and a New Design Language", *Proceedings of the Euromicro Symposium on Digital Systems Design*, pp. 2-7, 2001.

[20] R. Saha, "Content-Addressable Memory Speeds Up Lossless Compression", *Electronic Design* (online version), www.elecdesign.com, September 2003.

[21] R. Yang and C. Lee, "High-Throughput Data Compressor Techniques Using Content Addressable Memory", *Proceedings of the IEEE International Symposium for Circuits and Systems*, pp. 147-150, May 1994.

[22] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression", *IEEE Transactions on Information Theory*, Vol. 23, pp. 337-343, May 1977.