ABSTRACT
Upgrading the network infrastructure to increase the available bandwidth can be expensive and time consuming. In this paper we investigated whether applying compression to inter-process communication (IPC) messages, in order to increase the effective throughput of an existing network, can be beneficial or not. A literature study on lossless compression was used to select a few algorithms to be integrated into LINX, an IPC technology, as an extra layer to support lossless message compression. The testing involved sending messages, with real telecom data, between two nodes on a dedicated network with different network configurations and message sizes. To calculate the effective throughput for each algorithm, the round-trip time was measured. The conclusion was that the algorithms with the highest rate of compression yielded the best results.

Keywords
Compression, Message compression, Lossless compression, LINX, IPC, LZO, LZFX, LZW, LZMA, bzip2, LZ4, Effective throughput, Goodput

1. INTRODUCTION
The purpose of compression is to decrease the size of the data by representing it in a form that require less space. This will in turn decrease the time to transmit the data over a network or increase the available disk space [1].

The increased usage of mainframe computers lead to the development of new coding techniques, such as Shannon-Fano coding [2, 3] in 1949 and Huffman coding [4] in 1952. More complex compression algorithms that did not only use coding were later developed, for example the dictionary based Lempel-Ziv algorithms in 1977 [5] and 1978 [6]. The Lempel-Ziv algorithms have been the foundation for many other algorithms.

Compressing data requires resources, such as CPU time and memory. The expanding usage of multi-core CPUs in today’s communication systems increases the computation capacity quicker compared to the available bandwidth. One solution to increase the communication capacity is to compress the messages before transmitting them over the network. In theory this should improve the effective throughput and also reduce the network load, assuming that the data is compressible.

The purpose of this work, which was done for Ericsson, was to increase the available communication capacity in existing systems by simple means, without upgrading the network hardware or modifying the code of existing applications. The idea was to take advantage of compression by integrating it into the supplied communication framework used by the Ericsson Connectivity Packet Platform (CPP). The communication was done with LINX, which is an IPC technology developed by Enea [7]. LINX scales from microcontrollers to 64-bit CPUs, and supports any distributed system topology, from a single processor to large networks. By adding compression as an extra layer in LINX the communication capacity could hopefully be increased.

This paper is based on our thesis in computer science [8]. It starts with an introduction to compression and the contributions in Section 1. Section 2 contains some background information to compression and Section 3 contains work related to this paper. Section 4 contains the conclusions from the literature study as well as the compression algorithms chosen. The method, which includes the application, test cases and the data working set, is described in Section 5. This continues with the implementation of the algorithms and the test environment in Section 6. The results of the work resides in Section 7 and is followed by conclusions in Section 8. The paper ends with suggestions for future work in Section 9.

1.1 Contribution
We have tested how the performance of LINX IPC messages of sizes between 500 and 1400 bytes sent over a simple network are affected by implementing an extra layer of compression into LINX, and at which network bandwidths and message sizes it is beneficial to use such an implementation. Our results suggest that high rate of compression is more important than other factors when sending small messages over a simple network, however for more complex networks the importance of high compression ratio grows.

2. BACKGROUND INFORMATION
This section contains some background information to compression.

2.1 Compression classifications
One way to classify a compression algorithm is as lossy or lossless. Lossy compression reduces the data size by discarding information that requires large amounts of storage but is not necessary for presentation. It has proven to be
effective when applied to media formats such as video and audio [9]. The drawback of using lossy compression is that it is impossible to restore the original file, due to removal of essential data. Because of this, it is not suited for text-based data [10]. Lossless data compression can be achieved by various techniques that reduces the size without permanently removing any data. Compared to lossy compression, the original data can be restored. Lossless data compression can be found everywhere in computing, for example to save disk space, transmitting data over a network, communication over a secure shell etc. It is also used in embedded systems to improve the communication bandwidth and memory usage [11].

2.2 Compression factors

Compressing data requires execution time and memory to decrease the size of the data. The factors used to describe the performance of an algorithm are rate of compression, compression ratio and memory usage.

The rate of compression, also known as compression speed, is the time it takes to compress and decompress data. The time it will take for an algorithm to compress and decompress the data is highly dependent on how the algorithm works.

Compression ratio is how much a set of data can be compressed with a specific algorithm, i.e. how much smaller in size the compressed data is compared to the noncompressed data. The definition of compression ratio is:

$$R = \frac{\text{Size before compression}}{\text{Size after compression}}$$

This definition is sometimes referred to as compression factor in some of the papers used in the literature study. We chose to use this definition instead due to the cognitive perception of the human mind where larger is better [12].

Memory usage is how much memory a specific algorithm uses while compressing and decompressing. Like rate of compression, the memory usage is highly dependant on how the algorithm works.

2.3 Formulas for effective throughput

In our experiment, the throughput $T$ is defined as

$$T = \frac{D_c}{t}$$

where $D_c$ is the size of the compressed data and $t$ is the time required to send data between two nodes, including the time to compress and uncompress the data. While running the tests for a specific bandwidth, $T$ should be constant and $D_c$ and $t$ should vary depending on the compression algorithm used.

The effective throughput $T_e$, also called goodput, is defined as

$$T_e = \frac{D}{t}$$

where $D$ is the size of the noncompressed data and $t$ is the same as in Equation (2). In our tests $D$ will be constant in each test case. If we use no compression, the compressed size of the data $D_c$ will be equal to the size of the noncompressed data $D$, which means that by using Equation (2) and Equation (3), we get $T = T_e$ in the case of sending noncompressed data.

3. RELATED WORK

In Adaptive end-to-end compression for variable-bandwidth communication [13] by Knutsson and Björkman from 1999, messages were compressed using an adaptive algorithm which changed the compression level depending on the length of the network queue. They showed that the effective bandwidth could be increased on networks with 1-3 Mbit/s throughput by compressing the data using a computer with a 130 MHz CPU. The paper shows that it is possible to increase the effective bandwidth under certain circumstances, which is one of the things we aimed to achieve.

S. Dorward and S. Quinnlan experimented with different compression algorithms to improve the performance of packet networks in Robust Data Compression of Network Packets [14]. The experiments were conducted on 125 and 1600 byte packet sizes on a network with 10 kbit/s, 100 kbit/s, 1 Mbit/s and 10 Mbit/s link speed. They concluded that high rate of compression is important when compressing network packets, especially if the bandwidth is large when compared to the available computational capacity. This should apply to our work as well, since our message sizes are in the same range.

N. Motgi and A. Mukherjee proposed a Network conscious text compression system (NCTCSys) [15] in 2001, which compressed text based data and could be integrated on application level into different text based network transfer protocols like HTTP, FTP and SMTP. The compression applications tested were bzip2 and gzip together with LIPT (A lossless text transform to improve compression [16]). With this method they were able to reduce the data transmission time by 60-85%. This is similar to our work, but with the main difference being the protocol used and the size of the messages.

On the Distributed Computing Systems conference in 2005, C. Pu and L. Singaravelu presented Fine-Grain Adaptive Compression in Dynamically Variable Networks [17] where they applied adaptive compression to data packets using a fine-grain mixing strategy which compressed and sent as much data as possible and used any remaining bandwidth to send uncompressed data packets. The compression algorithms they tried were gzip, bzip2 and LZO which all have different properties regarding rate of compression and compression ratio. Their experiments were conducted on a network with up to 1 Gbit/s bandwidth and concluded that improvement gained when using the different compression algorithms was reduced as the physical bandwidth increased to the point where it was worse than noncompressed transmission. The maximum bandwidth where compression was beneficial was different for each of the algorithms. It was highest with LZO, which has the highest rate of compression of the three algorithms. This suggests that high rate of compression is an important property when compressing streaming data.

B. Welton, D. Kimpe, J. Cope et al. investigated if the effective bandwidth could be increased by using data compression, in the paper Improving I/O Forwarding Throughput
with Data Compression [18]. They created a set of compression services within the I/O Forwarding Scalability Layer and tested a variety of data sets on high-performance computing clusters. LZO, bzip2 and zlib were used when conducting their experiments. For certain scientific data they observed significant bandwidth improvements, which shows that the benefit of compressing the data prior to transfer is highly dependent on the data being transferred. Their results suggest avoiding computationally expensive algorithms due to the time consumption.

S. Shanmugasundaram and R. Lourdusamy published A Comparative Study Of Text Compression Algorithms in 2011, where they presented an overview of different statistical compression algorithms as well as benchmarks comparing the algorithms [19]. The benchmarks were focused on algorithms based on LZ77 and LZ78 but also included some other statistical compression techniques. The factors that were compared in the benchmarks were compression ratio and rate of compression. These benchmarks were important to our selection of compression algorithms.

In The hitchhiker’s guide to choosing the compression algorithm for your smart meter data [20], M. Ringwelski, C. Renner, A. Reinhardt et al. performed tests on compression algorithms to be implemented in Smart Meters1. They compared compression ratio, rate of compression and memory consumption for different algorithms. They suggest that the processing time of the algorithms are of high importance if the system runs on battery. This paper was relevant for the selection of compression algorithms.

In November 2012 M. Ghosh posted the results of a compression benchmark [21] done with a self-made parallel compression program called Pcompress [22] which contains C implementations of LZFX, LZMA, LzmaMt2, LZ4, libbsc, zlib and bzip2. The results are fairly new, which made them very relevant to our work when deciding which algorithms to integrate.

4. CHOICE OF ALGORITHMS

When comparing compression algorithms the most important factors are rate of compression, compression ratio and memory usage as described in Section 2.2. Most of the time there is a trade-off between these factors. A fast algorithm that uses minimal amounts of memory might have poor compression ratio, while an algorithm with high compression ratio could be very slow and use a lot of memory. This can be seen in the paper by K. Barr and K. Asanović [23].

Compressing the data before transferring it over a network can be profitable under the right circumstances. If the available bandwidth is low and the processing power of the system can compress the data in a short amount of time or significantly reduce the size of the data, the actual throughput can be increased. An algorithm that has high compression ratio is desirable but these algorithms take more time than those with lower compression ratio, which means that it will take longer time before the data can begin its journey on the network. On the other hand, if the transfer rate:

1System that sends wireless sensor signals to a target device
2Optimized and multi-threaded LZMA implementation

of the network is high, compressing the data can negatively affect the effective throughput. Even a fast algorithm with low compression ratio might be a bottleneck in some situations. By increasing the amount of allocated memory an algorithm has access to, an increase in compression ratio might be achieved. The size and the content of the noncompressed data may also affect the compression ratio.

In our type of application, i.e. compressing the data, transferring it from one node to another followed by decompressing it, fast algorithms are usually the best. However there are some circumstances that can increase the transmission time and therefore change this. Adaptive compression is a technique used to adaptively change the compression algorithm used when these kind of circumstances changes, which in turn can increase the effective throughput.

To investigate the effect of all these factors, we tried to select one algorithm that performs well for each factor. A list of the most common algorithms was made, and we then looked at the properties of each of these algorithms. We also made sure that we could find the source code for all the algorithms chosen.

Based on the literature study we selected the algorithms seen in Table 1. One of our theories was that since we were sending LINX IPC messages, the speed of the algorithm would be important because the messages would probably be quite small. This was also one of the conclusions we made from the literature study, but since we did our tests on different network bandwidths with specific message data and size we needed a verification of the conclusion. Due to this we selected some slow algorithms even though it contradicted the conclusion.

<table>
<thead>
<tr>
<th>LZO</th>
<th>bzip2</th>
<th>LZW</th>
<th>LZFX</th>
<th>LZMA</th>
<th>LZ4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Fast</td>
<td>Medium</td>
<td>Medium</td>
<td>Fast</td>
<td>Slow</td>
</tr>
<tr>
<td>R</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>M</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
<td>-</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 1: Comparison of compression speed (S), compression ratio (R) and memory usage (M) for algorithms. No data was found on the memory usage of LZFX.

5. METHOD

To determine the effect of compression when applied to LINX IPC messages, we first needed a suitable environment to conduct the tests. The idea was to construct a dedicated network where no other network traffic was present to maximize the available bandwidth and to make sure that each test was performed under the same conditions regardless of when it was conducted. To test the effect of different network bandwidths we configured the link speed of the servers interface connected to the data network to 10, 100 and 1000 Mbit/s and performed the tests for each bandwidth.

We were provided with an early prototype of the communication framework that already had LZFX integrated, which gave us a general idea on how to integrate the other algorithms. The source code for the selected algorithms was analysed and edited to fit the provided framework. The compression and decompression was integrated directly into
the LINX library as an extra layer to extend the functions that are handling the inter-process communication, as seen in Figure 1. The test application then used the LINX library to send IPC messages between the sender and receiver. The sender compresses a message (if compression is selected) and sends the message to the receiver, which decompresses the message followed by immediately compressing the message again and sending it back. Before sending the message, the sender adds the current time to the message header. When the transfer is complete the application uses this timestamp to calculate the round-trip time (RTT) of the message. The average compression ratio was calculated and printed out to the user. The message flow can be seen in Figure 1.

Figure 1: Message flow with compression as an extra layer in LINX

5.1 Test cases

The real telecom message data was exported from a Wireshark log. While analysing the log, we noticed that the messages were sent with a throughput of 14 Mbit/s. This could be because the bandwidth was low or the network load was high. With this information and the factors in Section 4, we constructed the test cases that can be seen below.

For 10, 100, and 1000 Mbit/s network bandwidth

- Send 5000, 10000 and 20000 messages, respectively, with telecom signal data of size 500, 700, 1000 and 1400 byte
- Compress and decompress the messages with LZO, LZFX, LZMA, bzip2, LZW, LZ4 and without compression
- Acquire the average round-trip time and compression ratio
- Calculate the effective throughput using the average round-trip time

Note that there was only one concurrent message on the network at any given time, i.e. we did not send a stream of messages. This means that one message was sent and the next message was not sent until the previous message returned.

As a last test we determined the maximum throughput for our application by sending many large messages over a long period of time while at the same time monitoring the throughput using iftop. This was done to see if we were utilizing the maximum available bandwidth.

To verify that the available bandwidth was correct we used iperf in TCP mode. We used TCP mode since we used LINX with TCP connections.

5.2 Data working set

The data set used in the experiments was taken from a dump of traffic data that was acquired by capturing the traffic of a node in a real telecom network. From this dump, 100 packets were exported as static arrays to a C header file using Wireshark. The exported packets were not filtered for any specific senders or receivers. To fill the payload of the messages in our experiment, we looped through the arrays in the header file and copied the data. Initial test runs showed that the protocol header of each packet could be compressed to a higher degree than the payload of the packet. Because of this, we excluded the headers of the packets to give the worst-case scenario. The total amount of data, excluding protocol headers, was approximately 20 kB.

6. IMPLEMENTATION

In this section we describe how we set up the test environment and implemented the chosen compression algorithms.

6.1 Test environment

The test environment is shown in Figure 2, where the Control Network is the network to access the servers. The connection between the servers, called Data Network in Figure 2, is where the tests were done. The Data Network is where we configured the bandwidth by setting the link speed on the servers, while the link speed of the switch remained the same in all tests. We also tested how the performance is affected when removing the switch between the servers and connecting them point-to-point. Both networks have 1000 Mbit/s network capacity. The servers have all the software needed to perform the tests, including a test program that sends noncompressed and compressed LINX IPC messages using the selected compression algorithms as seen in Figure 1. The servers run Ubuntu Linux 12.10 with kernel version 3.5.0-28-generic x86_64. The specification of the switch and server hardware can be seen in Table 2.

<table>
<thead>
<tr>
<th>CPU</th>
<th>2 x AMD Opteron(®) 6128 2.00 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>Samsung DDR3 32 GB</td>
</tr>
<tr>
<td></td>
<td>- Clock Speed: 1333 MHz</td>
</tr>
<tr>
<td>NIC</td>
<td>Intel 82576 Gigabit Network Connection</td>
</tr>
<tr>
<td></td>
<td>- Data rate: 10/100/1000 Mbps</td>
</tr>
<tr>
<td></td>
<td>- Number of ports: 2</td>
</tr>
<tr>
<td>Switch</td>
<td>Netgear GS105E</td>
</tr>
<tr>
<td></td>
<td>- Data rate: 10/100/1000 Mbps</td>
</tr>
</tbody>
</table>

Table 2: Server and switch hardware specification
6.2 Compression algorithms
The algorithms were integrated directly into the LINX library, more specifically into ../liblinx/linx.c. The method linx_send_w_s was modified to compress the messages with the selected algorithm while the linx_receive method handles the decompression. In these methods the RTT and compression ratio were also calculated. The test application, that uses the LINX library to send and receive messages, also selects which compression algorithm to use, how many messages that should be sent, the content and size of each message and where to send the messages. Here we also calculated the average RTT and compression ratio.

7. RESULTS
The results from the tests when using a switched connection can be seen in Figure 3, 4 and 5, while the measurements from the point-to-point connection can be seen in Figure 6, 7 and 8. All the measurements were calculated according to Equation (3). The lines represents the average effective throughput for one concurrent message when using the different compression algorithms and no compression. The available bandwidth can be seen in the text at the top of each figure, which is also the theoretical maximum throughput. The average compression ratio for each algorithm, with varying message sizes, can be seen in Figure 9. As a final test we measured the maximum throughput for the test application as described in Section 5.1. The result of this test can be seen in Table 3.

LZ4 and LZO performed almost equally and were the most effective of all the algorithms we tested. They are both designed to be very fast, just as LZFX. However, LZFX was always a few steps behind both LZ4 and LZO, and this gap seemed to increase with higher link speed. This could be the effect of lower compression ratio, as seen in Figure 9, or possibly lower rate of compression. LZ4 was slightly better than LZO, most of the time achieving a few percent higher effective throughput, and in worst case performing equal to LZO.

The performance of LZW was low, possibly because the compression ratio did not increase much when the size of the message increased, compared to other algorithms. This was likely because we chose to use a low number of bits for the code-words, due to the small size of the messages.

LZMA achieved the highest compression ratio of all the tested algorithms, but because of the low rate of compression, we did not see any improvement in effective throughput.

The worst of the algorithms tested was bzip2. This was because of the low rate of compression and the relatively low compression ratio. In Figure 9, we can see that the compression ratio for bzip2 was very low even when compared to the fastest algorithms, which should have the lowest compression ratio. This could be because the data was not suitable for this algorithm, but according to Burrows and Wheeler [24], the Burrows-Wheeler transform should work well for both text and binary data. It might instead depend on the fact that each message was relatively small. Because bzip2 uses blocks of size 100 kB to 900 kB when compressing, it would probably be more efficient for larger amounts of data.

Looking back at Table 1, the predictions were correct for LZO, LZFX, LZMA and LZ4. However, bzip2 seemed to have lower rate of compression, compared to LZMA, and the compression ratio was very low, while we expected it to be high. The achieved compression ratio for LZW was also lower than we had expected.

As discussed in Section 4, compression might become a bottleneck when transferring messages. When the link speed was set to 1000 Mbit/s, regardless of whether we used point-to-point or switched connection, we got negative effects when using any of the compression algorithms. In Table 3 we can see that the maximum throughput for the test application was around 260 Mbit/s, which means that we never used the maximum available bandwidth when the link speed was set to 1000 Mbit/s. However, we saw that the throughput was approximately 14 Mbit/s by analysing the logs for real telecom traffic. This means that it should be beneficial to use compression in the real environment since we saw improvement on bandwidths up to 100 Mbit/s.

We can see in Figure 3 to 8 that the effective throughput was consistently higher when using switched connections, compared to point-to-point connections. This suggests that the complexity of the network, e.g. more hops, has an impact on the effective throughput when using compression. The time to compress the message is constant, i.e. the number of hops has no impact on the compression time. However the time to transfer the message will increase, especially if the switches in the network use store-and-forward. Store-and-forward means that the switch must receive the whole frame before it begins to forward it to the recipient. We used only one switch, while in a real environment there might be several switches between the sender and receiver, thus increasing the transfer time even more.
Figure 3: Average effective throughput per message using various algorithms with 10 Mbit/s link speed and a switched network.

Figure 4: Average effective throughput per message using various algorithms with 100 Mbit/s link speed and a switched network.

Figure 5: Average effective throughput per message using various algorithms with 1000 Mbit/s link speed and a switched network.

Figure 6: Average effective throughput per message using various algorithms with 10 Mbit/s link speed.

Figure 7: Average effective throughput per message using various algorithms with 100 Mbit/s link speed.

Figure 8: Average effective throughput per message using various algorithms with 1000 Mbit/s link speed.
### Table 3: Maximum throughput for the test application with point-to-point and switched connection.

<table>
<thead>
<tr>
<th>Connection type</th>
<th>Maximum Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switched</td>
<td>261 Mbit/s</td>
</tr>
<tr>
<td>Point-to-point</td>
<td>265 Mbit/s</td>
</tr>
</tbody>
</table>

The maximum throughput for the test application was determined to be 261 Mbit/s with switched connection and 265 Mbit/s with point-to-point connection. The maximum throughput can be increased for the test application by removing unnecessary instructions and optimizing the code. This was a prototype application, which means that the final implementation will not necessarily need to calculate the compression ratio, RTT and other diagnostic information. The performance for compressed messages will increase more than for noncompressed messages in the final implementation.

### 8. CONCLUSION

We could see that it was profitable to use compression if the transfer time for a message was high compared to the time spent compressing and decompressing the messages, regardless of if this depends on low network bandwidth or many hops. The best choice of algorithm depends on the size of the messages, as well as the properties of the network, such as the complexity and available bandwidth. In our case the fastest algorithms, that still achieved a relatively high compression ratio, were best suited. This was also the conclusion we made from the literature study.

When sending messages with LINX IPC, the compression and decompression is only performed at the end-nodes, i.e. the rate of compression is independent of the network topology. This means that there might be an advantage to use algorithms with high compression ratio, for example LZMA, when considering very complex networks. With higher compression ratio, the amount of data that needs to be forwarded at each hop will be smaller, which also means that the amount of time spent forwarding packages decreases.

Another advantage of compression, given that the data can be compressed, is that the load on the network would be lower even if the effective throughput is not increased. If the goal is to decrease network load, while the latency is not as important, it would be more suitable to use algorithms with higher compression ratio.

A prediction for the future is that slow algorithms, with high compression ratio, will gain more from the increase in CPU power over time, compared to fast algorithms with low compression ratio. Both network speeds and the number of transistors in a CPU increase at a very high rate, but upgrading CPUs to a newer generation is probably easier and less expensive than upgrading the entire network infrastructure.

### 9. FUTURE WORK

There are some topics that we did not have time to investigate in this paper, but which may be interesting for future work. These include implementing message compression on the target architecture, parallel versions of the algorithms as well as adaptive compression with several algorithms. Integration of other algorithms with high rate of compression, like zlib and QuickLZ, can also be done to determine the best suited algorithm. An analysis of memory usage for the algorithms might also be of interest when considering implementation on the target architecture.

### 10. ACKNOWLEDGEMENTS

We would like to thank Ericsson, especially Marcus Jägemar and Magnus Schlyter, for excellent support and supervision. We would also like to thank Mats Björkman at Mälardalen University for his work as supervisor and examiner, and for assisting with theoretical and practical issues. Finally, we would like to thank Daniel Flemström, the manager at the Industrial Research & Innovation Lab, for helping us with the lab environment and other technical questions.

### REFERENCES


http://www.ieee-ghn.org/wiki/index.php/-
History_of_Lossless_Data_Compression_Algorithms


http://moinakg.wordpress.com/2012/11/01/compression-

