1. **Compression**:
   * The worst-case time complexity of the compression operation primarily depends on the size of the input message.
   * In the provided solution, the compression algorithm iterates over the input message bytes and performs compression on each chunk.
   * The compression algorithm complexity is typically linear or close to linear in the size of the input data.
   * Therefore, the worst-case time complexity of the compression operation can be considered as O(n), where n is the size of the input message.
2. **Decompression**:
   * Similar to compression, the worst-case time complexity of the decompression operation depends on the size of the compressed data.
   * In the provided solution, the decompression algorithm iterates over the compressed data bytes and performs decompression on each chunk.
   * The decompression algorithm complexity is also typically linear or close to linear in the size of the compressed data.
   * Therefore, the worst-case time complexity of the decompression operation can also be considered as O(m), where m is the size of the compressed data.

Since both compression and decompression operations have linear time complexity relative to their respective input sizes, the overall worst-case time complexity for compression and decompression can be considered as O(n) and O(m), respectively.

In practical scenarios, the actual performance may vary based on factors such as the implementation details of the compression algorithm, the characteristics of the input data, and hardware considerations. However, the provided analysis gives a general understanding of the computational complexity of the compression and decompression operations.

For compressing messages sent between clients in a local area network, several compression algorithms can be considered, depending on factors such as message content, required compression ratio, computational resources available, and acceptable decompression overhead. Here are some suitable compression algorithms along with justifications:

1. **Lempel-Ziv-Welch (LZW) Compression**: LZW is a dictionary-based compression algorithm that works well for compressing text data, which is often the case for messages sent between clients. It builds a dictionary of frequently occurring substrings and replaces them with shorter codes during compression. LZW is efficient for compressing repetitive data, such as text messages, and it provides good compression ratios.
2. **Deflate Compression (used in ZIP, PNG, and HTTP compression)**: Deflate combines the LZ77 algorithm for string matching and Huffman coding for entropy encoding. It is widely used and supported, making it a practical choice. Deflate can achieve good compression ratios and is relatively fast to decompress.
3. **Brotli Compression**: Brotli is a newer compression algorithm developed by Google, which offers better compression ratios compared to Deflate while maintaining similar decompression speeds. It can be a good choice for scenarios where high compression is desired with acceptable computational overhead.

Considering these options, let's choose the Deflate compression algorithm for its balance between compression ratio, decompression speed, and widespread support.