# Delta Compression

## What is Delta Compression

Compression is **the process of encoding information using less data than its original representation**. Delta compression is concerned with data sets and their relationships to one another. While traditional compression operates on a single data set, **delta compression operates on two separate data sets: source data and target data**. With these two data sets, **a delta compression algorithm aims to encode the differences into what is called a delta**.

The fundamental idea is to **initially send a copy of source data to a remote computer and from that point only send deltas**. Ideally these **deltas encode only the differences between source and target, which sometimes makes them effective and comparatively small**.

> http://cis.poly.edu/tr/tr-cis-2002-02.pdf (http://cis.poly.edu/tr/tr-cis-2002-02.pdf)

⚠ IMPORTANT

While delta compression can be very effective it imposes strict ordering requirements. More specifically, **deltas need to be applied in the order they are produced**. If two deltas A and B are sequentially generated by client X, then client Y is required to apply delta A before delta B. This introduces dependencies and additional complexity between deltas/packets.

☞ WARNING

As with compression in general, **delta compression is subject to space-time trade-off**. This effectively means that delta compression results in decreased space/bandwidth at the cost of sometimes significant increases processing time.

## How to use Delta Compression

Assume that we have a large buffer with serialized objects that we have previously sent to a remote computer, let this be called the **source buffer**. This means that **the source buffer is already distributed and identical at both computers**.

```
// Assume that this buffer is filled, distributed and identical at both computers
PacketBuffer sourceBuffer(1024);
```

Now assume that we make small changes to our local source buffer, let this new and changed buffer be called the **target buffer**:

```
// We define a new buffer to seraialize into
PacketBuffer targetBuffer(1024);

// ...

// In a real environment, we would pack all our objects into the buffer here

// ...
```

We can use a delta compression algorithm to encode the differences between the new target buffer and our old source buffer into a delta. Ideally this **delta is much smaller than any of the previously mentioned buffers**. We may then **send our delta to the remote computer**.

```
var deltaToSend = DeltaCompressor::Encode(sourceBuffer, targetBuffer, DeltaCoder.Algorithm.BsDiff);

// In a real environment, we would send the delta to a remote client here
```

Assuming that a remote computer successfully receives a delta it can **transform its source buffer into our target buffer using its received delta**.

```
// Assuming that a remote computer received data which it constructed a PacketDelta object with
PacketDelta receivedDelta;

targetBuffer = DeltaCompressor::Decode(sourceBuffer, receivedDelta, DeltaCoder.Algorithm.BsDiff);

// ...

// In a real environment, we would unpack all our objects from the buffer here

// ...
```

**WARNING**

Not all delta compression algorithms are effective at compressing all types of data. This means that **the choice of delta compression algorithm can play a significant role**. For example, some are more efficient at dealing with volatile data than others. Either experiment with multiple algorithms or do your research before settling for a delta compression algorithm.

**WARNING**

Since delta compression algorithms exploit differences between data sets they can be very **effective for data that is highly stable**. Delta compression **may not be particularly useful in situations where differential changes between sets are large**.

**NOTE**

It is required to **use the same delta compression algorithm when encoding as when decoding**.