Write a program in C/C++ that satisfies the following requirements:

1. Creates a TCP listening socket bound to INADDR_ANY and a port number provided by the user as a command-line argument.
2. Accepts incoming connections on this socket, then reads and parses data sent from the peer.
3. The data sent from the peer is in a simple type-length-value (TLV) format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	...
TYPE		LENGTH				VALUE (variable length)																										

- TYPE (2 bytes, network byte order)
    - Valid values:
        - 0xE110 - Hello
        - 0xDA7A - Data
        - 0x0B1E - Goodbye
- LENGTH (4 bytes, network byte order)
- VALUE (variable length, number of bytes denoted by the "LENGTH" field)

4. As the program reads data from the socket, it should summarize each TLV blob to stdout in the following format:

   [SrcIP:SrcPort] [TYPE] [LENGTH] [First 4 bytes of VALUE in hex]

   For example, if the program is listening on TCP port 1234, and a peer sends data using netcat as follows:

```
echo 'E11000000000DA7A0000000501020304050B1E00000000' | \
xxd -r -p | \
nc -p 5678 localhost 1234
```

   The program would print the following to stdout:

```
[127.0.0.1:5678] [Hello] [0] []
[127.0.0.1:5678] [Data] [5] [0x01 0x02 0x03 0x04]
[127.0.0.1:5678] [Goodbye] [0] []
```

5. The program must be able to handle multiple long-lived peers concurrently sending TLV blobs at various rates.
6. The program must robustly handle connections from malicious and/or buggy peers (intentionally left vague - handle according to your experience for best practices).

You may leverage open source libraries (e.g. boost, libevent, etc.).

Be prepared to discuss your implementation in detail, including reasons behind your design decisions, any tradeoffs you made, and theoretical performance limits.