7.8. Packet Reassembly

7.8.1. What is it?

Network protocols often need to transport large chunks of data which are complete in themselves, e.g. when transferring a file. The underlying protocol might not be able to handle that chunk size (e.g. limitation of the network packet size), or is stream-based like TCP, which doesn't know data chunks at all.

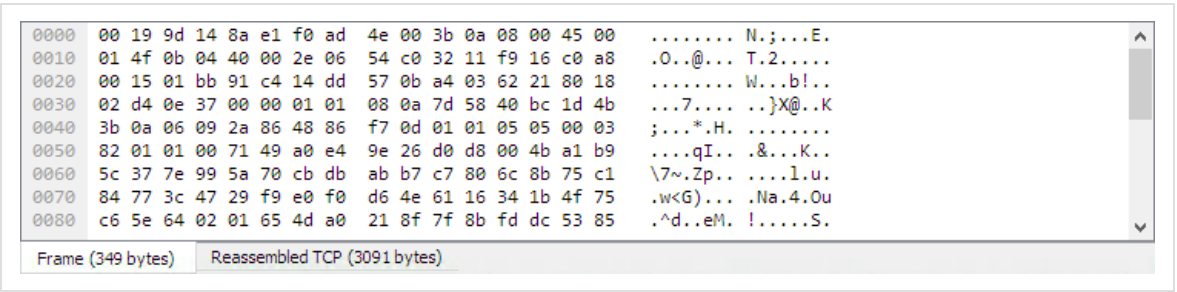
In that case the network protocol has to handle the chunk boundaries itself and (if required) spread the data over multiple packets. It obviously also needs a mechanism to determine the chunk boundaries on the receiving side.

Wireshark calls this mechanism reassembly, although a specific protocol specification might use a different term for this (e.g. desegmentation, defragmentation, etc).

7.8.2. How Wireshark handles it

For some of the network protocols Wireshark knows of, a mechanism is implemented to find, decode and display these chunks of data. Wireshark will try to find the corresponding packets of this chunk, and will show the combined data as additional pages in the "Packet Bytes" pane (for information about this pane. See Section 3.19, "The "Packet Bytes" Pane").

Figure 7.6. The "Packet Bytes" pane with a reassembled tab



Reassembly might take place at several protocol layers, so it's possible that multiple tabs in the "Packet Bytes" pane appear.

> **Note**
>
> You will find the reassembled data in the last packet of the chunk.

For example, in a *HTTP* GET response, the requested data (e.g. an HTML page) is returned. Wireshark will show the hex dump of the data in a new tab "Uncompressed entity body" in the "Packet Bytes" pane.

Reassembly is enabled in the preferences by default but can be disabled in the preferences for the protocol in question. Enabling or disabling reassembly settings for a protocol typically requires two things:

1. The lower level protocol (e.g., TCP) must support reassembly. Often this reassembly can be enabled or disabled via the protocol preferences.
2. The higher level protocol (e.g., HTTP) must use the reassembly mechanism to reassemble fragmented protocol data. This too can often be enabled or disabled via the protocol preferences.

The tooltip of the higher level protocol setting will notify you if and which lower level protocol setting also has to be considered.

7.8.3. TCP Reassembly

Protocols such as HTTP or TLS are likely to span multiple TCP segments. The TCP protocol preference “Allow subdissector to reassemble TCP streams” (enabled by default) makes it possible for Wireshark to collect a contiguous sequence of TCP segments and hand them over to the higher level protocol (for example, to reconstruct a full HTTP message). All but the final segment will be marked with “[TCP segment of a reassembled PDU]” in the packet list.

Disable this preference to reduce memory and processing overhead if you are only interested in TCP sequence number analysis (Section 7.5, “TCP Analysis”). Keep in mind, though, that higher level protocols might be wrongly dissected. For example, HTTP messages could be shown as “Continuation” and TLS records could be shown as “Ignored Unknown Record”. Such results can also be observed if you start capturing while a TCP connection was already started or when TCP segments are lost or delivered out-of-order.

To reassemble of out-of-order TCP segments, the TCP protocol preference “Reassemble out-of-order segments” (currently disabled by default) must be enabled in addition to the previous preference. If all packets are received in-order, this preference will not have any effect. Otherwise (if missing segments are encountered while sequentially processing a packet capture), it is assumes that the new and missing segments belong to the same PDU. Caveats:

- Lost packets are assumed to be received out-of-order or retransmitted later. Applications usually retransmit segments until these are acknowledged, but if the packet capture drops packets, then Wireshark will not be able to reconstruct the TCP stream. In such cases, you can try to disable this preference and hopefully have a partial dissection instead of seeing just “[TCP segment of a reassembled PDU]” for every TCP segment.
- When doing a capture in monitor mode (IEEE 802.11), packets are more likely to get lost due to signal reception issues. In that case it is recommended to disable the option.
- If the new and missing segments are in fact part of different PDUs, then processing is currently delayed until no more segments are missing, even if the begin of the missing segments completed a PDU. For example, assume six segments forming two PDUs `ABC` and `DEF`. When received as `ABECDF`, an application can start processing the first PDU after receiving `ABEC`. Wireshark however requires the missing segment `D` to be received as well. This issue will be addressed in the future.
- In the GUI and during a two-pass dissection (`tshark -2`), the previous scenario will display both PDUs in the packet with last segment (`F`) rather than displaying it in the first packet that has the final missing segment of a PDU. This issue will be addressed in the future.
- When enabled, fields such as the SMB “Time from request” (`smb.time`) might be smaller if the request follows other out-of-order segments (this reflects application behavior). If the previous scenario however occurs, then the time of the request is based on the frame where all missing segments are received.

Regardless of the setting of these two reassembly-related preferences, you can always use the “Follow TCP Stream” option (Section 7.2, “Following Protocol Streams”) which displays segments in the expected order.

Prev Up Next

7.7. Time Zones Home 7.9. Name Resolution