Meraki

Wireshark, Where Did the Time Go?

At Cisco Meraki, we depend heavily on open source software to help us solve today's networking problems. This white paper focuses on a contribution that we made to a powerful and popular open source tool: Wireshark.

WHITEPAPER
FEBRUARY 2019

Introduction

Taking over-the-air packet captures is one of the fundamental tasks in designing, maintaining and troubleshooting wireless networks. Typically, a packet capture is viewed as a list of the captured frames. Wireshark makes it easy to dive in and inspect each of these frames and all of the fields that they are comprised of. Understanding the content of wireless frames in this way is critical to understanding what is going on, but it doesn't tell the whole story. At Meraki, we developed a new way to visualize captured packet data that makes the timing of these frames much clearer, making it much easier to diagnose certain types of issues. Read on to learn about our approach.

802.11 packets can be sent at a wide range of bitrates — from 1 Mbps to 1.73 Gbps these days. With 3 orders of magnitude in bitrate variation, the amount of time a packet takes to transmit varies hugely. Furthermore, several parts of the core operation of the MAC protocol itself are time-sensitive. The Enhanced Distributed Coordination Function (EDCA) and the Network Allocation Vector (NAV) are two key functions that are time-based.

Understanding and observing any of these time-based features was difficult when looking at packet captures in Wireshark, where only a textual list of packets is shown. The lack of any clear reference to the timing of packets, other than a numerical timestamp shown by the packet, makes debugging issues really hard. In order to gain a better understanding of what was going on in packet captures and to support ongoing innovations in packet scheduling, we needed a better way to visualize capture data.

We enhanced the visualization of 802.11 packet captures in Wireshark, the de facto tool of choice for looking at packet captures. Wireshark decodes packet captures using a series of "dissectors" — one for each type of protocol being decoded — and implements dissectors for hundreds of protocols.

About 802.11:

802.11 is the standard used for Wireless Local Area Networks (WLANs) which has been around for more than two decades

About Wireshark:

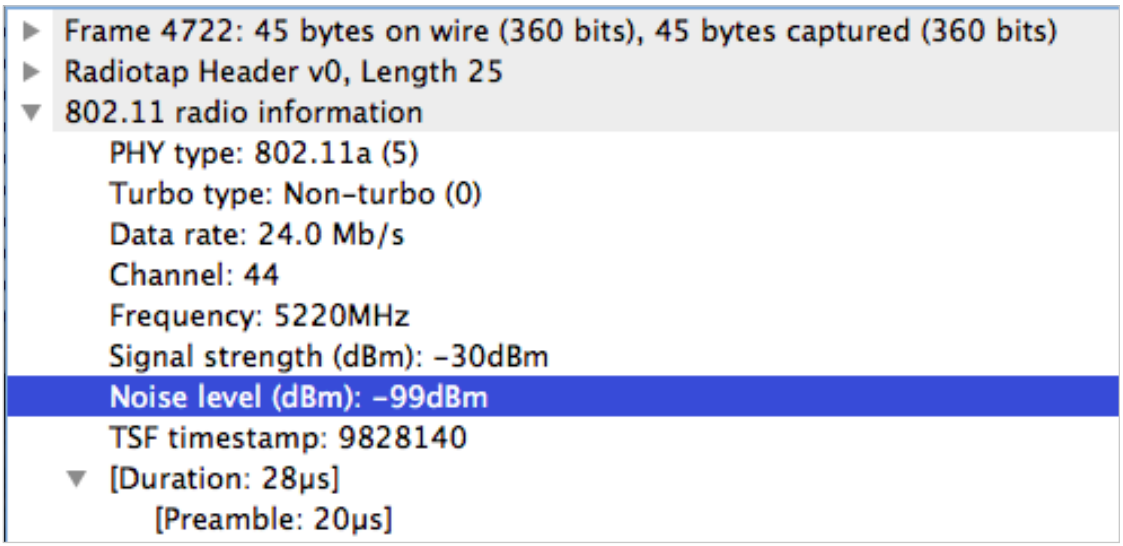
Wireshark is one of the most popular software tools used by network administrators and engineers to examine what goes on inside wired and wireless networks.

Packet Capture Formats

There are several different formats used by packet capture generators to record 802.11 packet metadata (data about the modulation, hardware timestamp, etc.). These formats appear as separate protocols to Wireshark. The most common formats are Radiotap and PRISM.

In version 2.0 Wireshark added a pseudo “meta” dissector, called the “wlan_radio” dissector, which is called after the various 802.11 metadata dissectors have run on the data. It unifies the wireless information under a single set of searchable, displayable named parameters — such as bitrate, timestamp, phy type, channel, etc. This unified wlan_radio dissector was the obvious place to implement the first step of this work.

For a first release, we submitted a patch to this wlan_radio meta dissector that calculates the duration of every 802.11 packet. It takes all the various physical layer information about the packet, and together with the length of the headers and data in the packet, calculates both the duration of the preamble itself as well as the duration of the whole packet transmission (preamble and data). The calculated total duration is made available in the wlan_radio dissector as the field wlan_radio.duration, and the preamble duration alone as wlan_radio.preamble. In Wireshark, as long as sufficient physical layer information about the packet is available, you can see this calculated duration by expanding the decoded tree of a captured 802.11 packet.



Packet dissector tree showing wlan_radio dissector with new calculated duration fields

Aggregates

The first release of this patch was not able to handle aggregates, also known as A-MPDUs. These are used in recent releases of 802.11 (802.11n and 802.11ac) in order to achieve good efficiency at high bitrates. The problem they solve is that the overhead in transmitting a packet — the preamble, the ACK response, the backoff time, and possible Request to Send / Clear to Send (RTS/CTS) exchange — is significant. At 1 Gbps rates, the time taken for this overhead dwarfs the time taken to send the actual data. The solution is to send many 802.11 data packets (MPDUs) in the data field of a single physical layer packet (PPDU). Thus, all this overhead is incurred only once while sending many data packets.

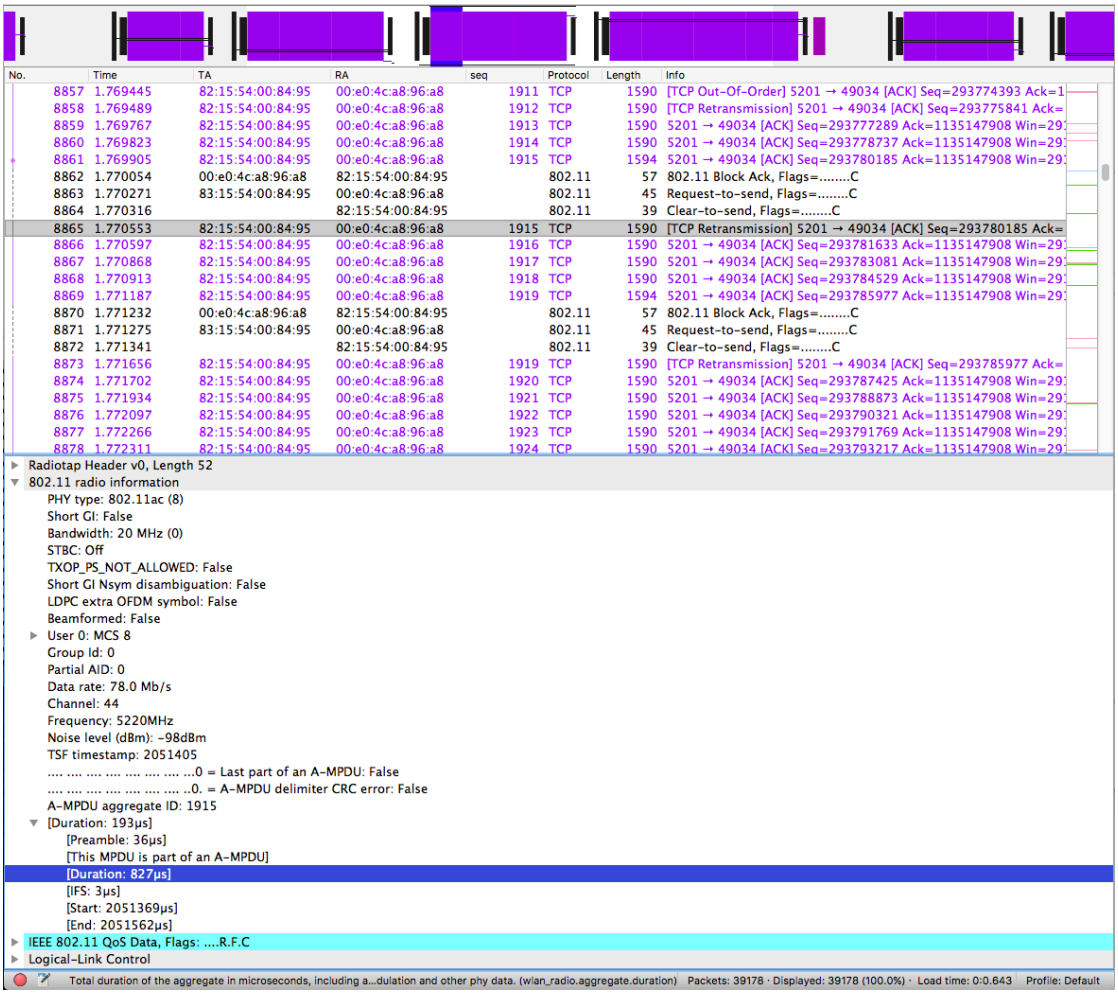
Splitting the Aggregates

It is common in Wave 2 equipment to send up to 192 data packets in a single aggregate. This reduces the MAC and PHY layer overhead by a factor of up to 192!
The 802.11 hardware receives the single physical aggregate and splits it out into multiple 802.11 frames (MPDUs), so at the software level and in packet captures, the 802.11 frames appear as many separate packets.

Timestamps

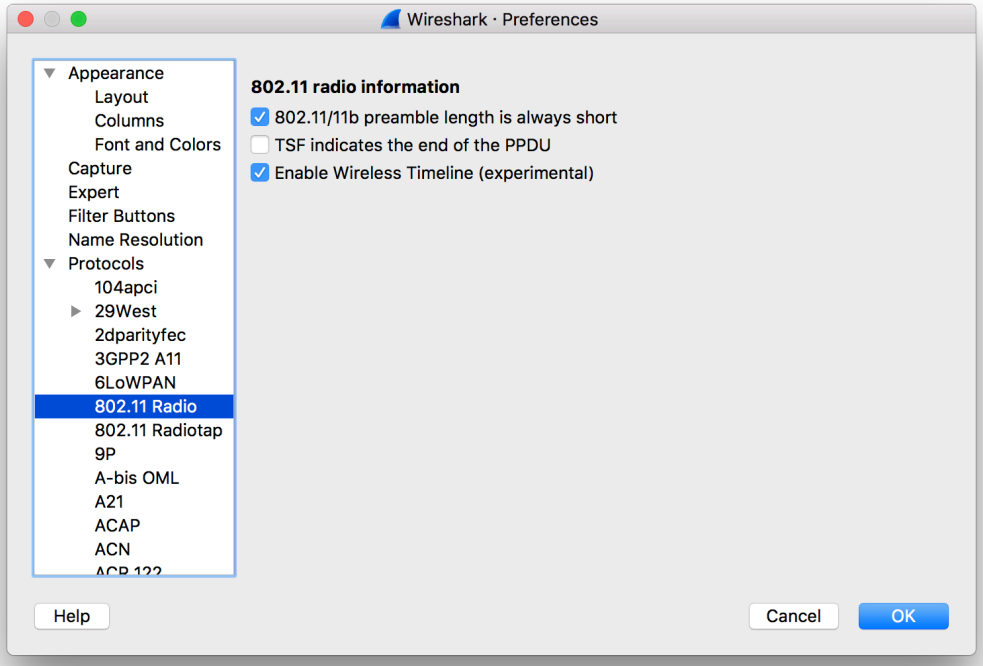
Different hardware and drivers generate these separate packets slightly differently. For instance, some hardware timestamps the packets with the time just after the preamble, before any data is received. This kind of hardware reports the same timestamp for all packets within an aggregate. Some other hardware timestamps the end of the physical frame, and on this hardware the timestamp field is only present in a packet capture on the last packet in an aggregate.

After testing with three different capture generators, we released a second patch that correctly handles the various methods we found for timestamping aggregates. This patch includes a wlan_radio.aggregate field that is set if the packet is detected to be part of an aggregate based on the surrounding packet timestamps. The preamble duration (wlan_radio.preamble) is only included for the first packet in the aggregate, and a total duration for the whole aggregate is also calculated (wlan_radio.aggregate.duration). All durations are calculated in microseconds.



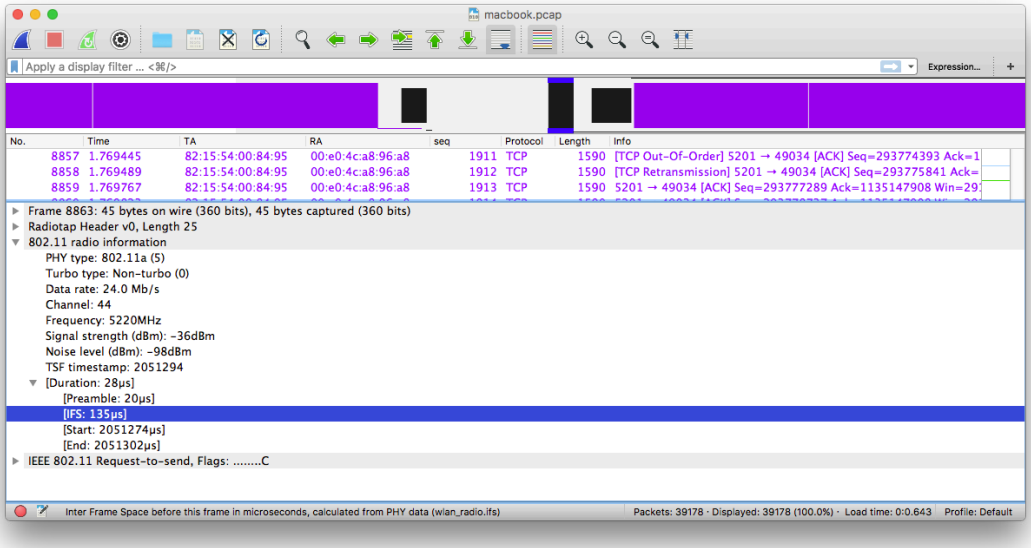
Wireshark showing a timeline with several aggregate frames in view. The selected packet is the first in an aggregate, showing an Inter Frame Space (to the prior packet), the preamble duration, the duration of the first subframe, and the duration of the whole aggregate.

In addition, with duration and timestamp information available, it's possible to calculate a timestamp for both the start of the packet (before the preamble) and the end of the packet. Since different generators use different timing references, and it's not always easy to determine what the reference is for a particular file, we added a preference to the wlan_radio dissector.[1] This preference allows you to specify whether the timestamp should be interpreted as referring to the start of the data field or the end. This preference must be set to match your generator correctly for your view of the file to be correct.



Wireshark Preferences Panel, showing 802.11 radio protocol preferences

[1] See in Wireshark v4.2.4 under Preferences → Protocols → 802.11 radio → TSF indicates the end of the PPDU.



Wireshark Preferences Panel, showing 802.11 radio protocol preferences

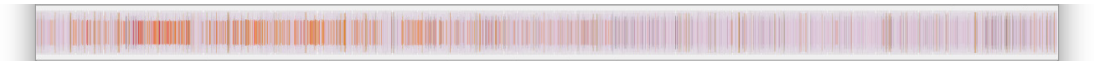
One of the important pieces of information to observe the operation of the Enhanced Distributed Channel Access (EDCA) function is the inter-frame space: the amount of time between the end of one packet and the start of the next. Now that the dissector calculates packet start and end times, it is easy to calculate the inter-frame space. This is exposed as the field wlan_radio.ifs.

Visualizing the Packets

With all of this data now available, we next added the ability to visualize captured packets on a horizontal timeline. There can be millions of packets in a capture, so how can these be shown on a timeline that may only be 1,000 pixels wide? The implementation allows the timeline to be zoomed in and out using the scroll wheel, from showing the whole packet capture across the window to showing individual microseconds.

In order for the display to be useful when zoomed out, with thousands or millions of packets on the screen, the display is fully anti-aliased. A custom paint routine renders the timeline, scanning through the packet data to calculate the brightness corresponding to how much every packet overlaps each pixel in the timeline. When zoomed out, density variations in the traffic can be clearly seen, with heavy traffic areas appearing darker.

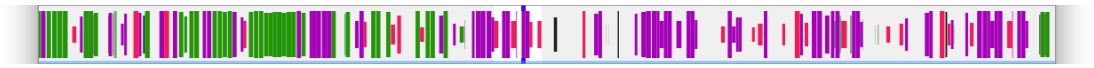
The signal strength is used to vary the height of the packet; hence, the traffic from stations with differing signal strengths forms different vertical bands, which can be helpful in understanding what is going on.



Timeline zoomed out, showing many packets across the timeline, with different signal strengths (heights) visible. Traffic is denser to the left in this example.

At this time, the timeline feature is experimental, and you must enable the timeline display in Wireshark in the 802.11 radio protocol preferences in order to see it. You should also set your generator's timing reference correctly there as well (whether the hardware timestamps indicate the start or end of each packet). To zoom on the timeline around the mouse pointer, use the scroll wheel up and down. As long as hardware timing data and physical layer modulation information is available for all packets in the capture, and there are no large backwards skips in time (wlan_radio.ifs value highly negative, which upsets the rendering), the timeline will display.

With the timeline feature enabled, a range of different 802.11 phenomena become immediately clear as compared to previously where one only had the packet list to look at. Packet captures can now be understood with greater clarity from the foreground color specified by coloring rules. Thus, by entering new coloring rules, users can easily pick out different aspects on the timeline in color. For instance, in some cases problems are caused by management traffic (in particular beacons and probe responses) being transmitted at low bitrates, and taking up a significant portion of the airtime. By creating a coloring rule for this traffic this issue can be clearly observed on the timeline.



Beacons in purple, Probe responses in green

In addition, airtime utilization can be calculated by using the wlan_radio.duration field and summing it with the iograph feature, by using tshark, or by using a Lua script. Another feature that is interesting to look at is retransmissions — these are marked by a bit in the 802.11 header, so they can be colored. As a result, it becomes easy to look over a long capture and find periods of high retransmission. Coloring can also be used to pick out packets sent by a particular station.

Traffic from one client in purple, another in blue. Retransmissions in Orange.

Wireshark allows the packet list to be filtered with a matching expression. On the timeline, we added a feature where the top half of the timeline always displays all packets in the capture, the bottom half is filtered according to the display filter. This allows the user to quickly view the filtered packets in context. It also allows for quick visual experiments; e.g., you can filter for all packets sent by a particular station and quickly see on a large capture when that station was active.

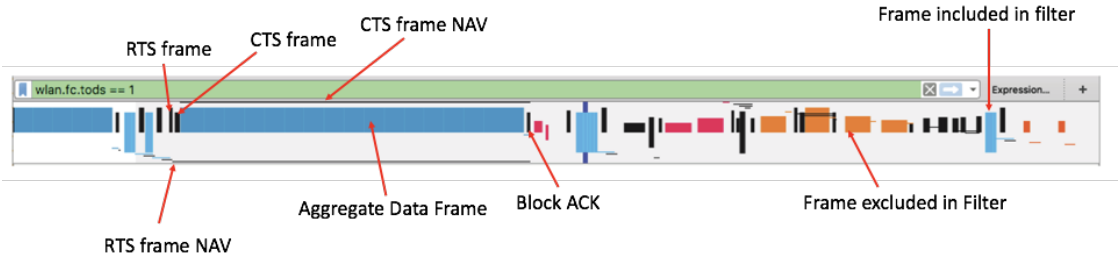
Diving Deeper

In 802.11, the header of every packet contains a duration field. This sets the Network Allocation Vector (NAV) for all stations the packet was not directed at. This sets a “virtual carrier sense” in those stations, telling them to hold off from transmitting for a duration after the packet with the NAV is received.

Showing a hidden node scenario. Nodes 1 and 2 can see each other, and nodes 2 and 3 can see each other, but node 1 cannot see node 3 and vice-versa.

This critical function helps 802.11 work well in situations where some nodes are hidden from others — for example, when one station can hear transmissions from a second station, but not the third station that the second station is communicating with. In this case, the first station might transmit over the third station’s responses to the second station, resulting in a collision at the second station, with the data corrupted and the second station unable to decode the third station’s response. The duration field helps to prevent immediate responses from being corrupted this way, and is how the RTS/CTS exchange acts to protect the immediate data exchange after it.

At higher zoom levels, this duration field, and the protection it should provide, is illustrated in the timeline by a horizontal line drawn to the right of the packet. You can clearly see in capture files how data packets set a NAV duration to protect the immediate ACK response that is expected, and how RTS and CTS packets protect their subsequent immediate DATA and ACK exchanges.



Timeline showing display filter in place. Frames to the AP are shown in full height NAV is plotted, and you can see RTS/CTS protecting several frame exchanges.

Yet another phenomenon that can be clearly seen on the timeline is collisions. In a long stream of packets during a download, a collision appears as a single packet length gap, where neither of the packets that collided could be decoded properly.

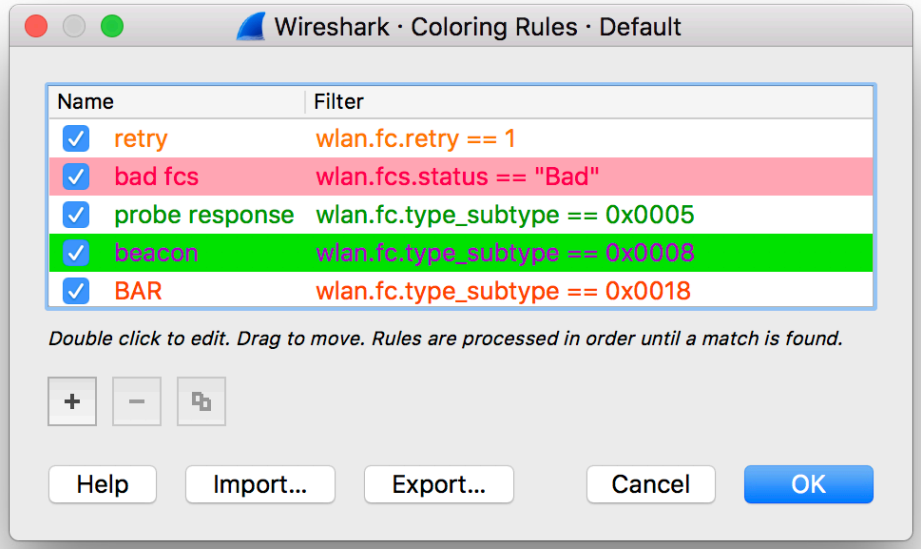
We discovered that some common 802.11 packet capture generators have a number of bugs. For example, two common generators incorrectly record the bitrate for packets where the CRC does not match the packet correctly — in this case, recording the lowest possible bitrate rather than the real bitrate. This results in the calculated duration for some packets being much longer than the duration actually was, and thus they appear much wider than they should in the timeline. Sometimes this results in a packet overlapping others in the timeline, and the calculated inter frame space (wlan_radio.ifs, the gap between frames) goes negative!

Code Release

You can try out this timeline feature today. It has been contributed to Wireshark and is included in the Wireshark 2.4.3 release.

At this time the timeline feature is experimental, and you must enable the timeline display in the 802.11 radio protocol preferences in order to see it. You should also set your generator's timing reference correctly on the timeline (whether the hardware timestamps indicate the start or end of each packet). To zoom on the timeline around the pointer, use the scroll wheel up and down. As long as hardware timing data and physical layer modulation information is available for all packets in the capture, and there are no large backwards skips in time (for instance, if the value of wlan_radio.ifs is highly negative, this will upset the rendering), the timeline will display.

It is useful to add a set of coloring rules to illustrate various aspects of the 802.11 protocol, such as:



Wireshark Coloring Rules Dialog showing some rules helpful for inspecting 802.11 packet captures on the timeline.

This tool, along with our other enhancements to Wireshark, has been instrumental in finding the root cause of numerous difficult wireless problems at Meraki. These improvements are key to enabling some ongoing performance improvements and advanced packet scheduler work that Meraki is automatically rolling out to all of our customers.