WiPal

IEEE 802.11 traces manipulation software

This manual is for WiPal (version 5.0, updated 13 January 2010.)

WiPal is a piece of software dedicated to IEEE 802.11 traces manipulation. It comes as a set of programs and a C++ library. A distinctive feature of WiPal is its merging tool, which enables merging multiple wireless traces into a unique global trace. WiPal's key features are flexibility, ease of use, and efficiency.

Table of Contents

1 The programs

This part documents the programs WiPal features. Looking for a specific command? See [Program index], page 23.

1.1 Invocation

WiPal's programs all use the same invocation scheme:

```
wipal-<command> [options] [inputs] [outputs]
```

The command line may include no options and, depending on the program, there may be no inputs or no outputs. Most programs expect at least one input however. See the specific documentation for each program in order to know how many inputs and outputs each program expects.

Inputs, outputs, and options may be mixed on the command line, e.g.:

```
wipal-simple-merge -n -P input1.pcap input2.pcap output.pcap
wipal-simple-merge input1.pcap input2.pcap output.pcap -P -n
wipal-simple-merge input1.pcap -n input2.pcap -P output.pcap
...
```

are all equivalent.

WiPal's programs use getopt(3) to parse options, so they only have short options (no long options) composed of a dash followed by a letter (e.g. ‘-a’, ‘-t’, etc.) Option letters *always* have the same meaning whatever the program. All options are not available for all programs though (some options do not make sens with some programs). For instance, ‘-P’ always means the invoked program should consider frames with non-zero Prism fields as invalid. In order to know which options a program accept, use the ‘-h’ option.

Finally, some options expect an extra argument right after they are provided:

```
wipal-test-uniqueness -a hsh_80211 input.pcap
                         ^^^^^^^^^
                         This is not an input
```

1.1.1 Available options

‘-8’ When comparing two packets, only compare IEEE 802.11 frames. Do *not* compare Prism or PCAP headers. This option is incompatible with traces of PCAP link type EN10MB.

‘-a’ See Section 1.7 [Unique frames], page 9. Specify which attributes the program must use to identify unique frames. An attribute specifier must follow this option on the command line. To see a list of valid attribute specifiers, use the ‘-h’ option.

‘-b’ When comparing two packets, only compare packet bytes. Do *not* compare PCAP headers.

‘-c’ Do not print column headers. This is the default when standard output is not a TTY.

‘-C’ Do print column headers. This is the default when standard output is a TTY.

'`-d`' When comparing two packets, compare everything: PCAP headers and packet bytes. This is the default.

'`-e`' In table outputs, do *not* use a column to report error values. This is the default.

'`-E`' In table outputs, *do* use a column to report error values.

'`-f`' Dump an hexadecimal summary of 802.11 frames on a single line.

'`-F`' Provide a full hexadecimal dump for each frame.

'`-g`' Enable debugging output. As of now this only makes WiPal programs to display their options right after they parse the command line.

'`-h`' Help. Print a short summary describing how one should invoke the program, which options it accepts, and possibly which attribute specifiers are accepted for option '`-a`'.

'`-i`' In table outputs, do *not* print frame indices.

'`-I`' In table outputs, *do* print frame indices. This is the default.

'`-m`' Specify a MAC address mapping file.

Some WiPal programs need to map MAC addresses to other identifiers. For instance, `wipal-extract-unique-frames` with the '`seq_bss_tmp`' attributes maps MAC addresses to 32-bit integers for performance reasons. `wipal-anonymize` maps real addresses to anonymous ones. Each program stores these mappings into a file so they can be reloaded and reused latter. This option allows users to control the name of this file.

When not specified, the '`MAC.map`' filename is used.

The file is just a plaintext file where each line contains a value and the corresponding MAC identifier.

A filename should follow this option. The file might not exist (in which case it will be created). If it exists, it might be extended, but will not be truncated.

'`-n`' Consider Prism headers are little endian. This is the default when the corresponding PCAP file is little endian. Note that some broken traces are big endian yet have little endian Prism headers. Thus this option.

'`-N`' Consider Prism headers are big endian. This is the default when the corresponding PCAP file is big endian.

'`-o`' When comparing two traces with `wipal-cmp`, compare everything (PCAP headers and packet bytes, as with option '`-d`') and count how many bytes differ. The count is printed on standard output. The exit status remains unchanged.

'`-p`' In Prism headers, do not consider noise fields have a special meaning. This is the default.

'`-P`' In Prism headers, consider non-null noise fields indicate a PHY error, and thus an invalid frame. Such frames will be ignored, e.g. with `wipal-cat` they will not appear in the output.

This option implicitly implies the input trace is composed of Prism headers (as PCAP link type).

‘-q’ Quiet. Produce minimal output.

‘-r’ Blacklist a given reference frame. The reference frame will then be ignored and will not be used during synchronization. See Section 1.6 [Synchronization], page 9.

A reference frame identifier must follow this option, e.g. `42-51` (indicating the reference frame composed of the unique frames 42 and 51).

You may use this option multiple times, e.g.

```
wipal-simple-merge -r 42-51 -r 666-505 \
    input1.pcap input2.pcap output.pcap
```

will blacklist both references `42-51` and `666-505`.

‘-s’ Specify an ESSID mapping file.

This option works as ‘`-m`’ but for files that map ESSIDs to other values. For instance, `wipal-anonymize` maps valid ESSIDs to anonymous ESSIDs.

See ‘`-m`’ for details. The default mapping file is ‘`ESSID.map`’.

‘-t’ When comparing two packets, only compare IEEE 802.11 frames, along with some timestamps (e.g. PCAP time, Prism MAC time, etc). Which timestamps are used depends on the traces’ link types (and whether options ‘`-y`’ or ‘`-Y`’ are provided as well). Compare time values with a precision of 106 microseconds by default (that is, assume two values are equal when they are spaced by less than 107 microseconds). You can change the precision using option ‘`-x`’.

‘-u’ In table outputs, do *not* print microsecond timestamps. This is the default.

‘-U’ In table outputs, *do* print microsecond timestamps.

‘-v’ Display the program’s version (actually the version of the WiPal package the program come from).

‘-x’ Change precision for timestamps comparisons (e.g. when using `wipal-cmp` with ‘`-t`’ or when merging or synchronizing traces).

By default, when the duration between two timestamps is 106 microseconds or less, WiPal programs consider the timestamps are equal. The rationale for this behavior is 106 microseconds is half the shortest frame interarrival time between two IEEE 802.11 frames (in infrastructure mode). Thus, this is the largest value one can afford when synchronizing IEEE 802.11 traces.

Use ‘`-x`’ to change this value. The new expected precision should be right after ‘`-x`’ on the command line.

‘-y’ Force the use of PCAP timestamps.

Some traces contain multiple timestamps for each frame. For instance, PCAP traces with link type `PRISM_HEADER` have the standard PCAP timestamps plus extra PHY-level timestamps provided by the network adapter. WiPalprograms’ policy is to use the most precise timestamps (that is, to ignore PCAP timestamps when something else is available). This option alters this behavior and forces programs to use PCAP timestamps.

‘-Y’ Force the use of PHY-level timestamps when available. This is the default. See option ‘`-y`’ for a more detailed explanation.

1.1.2 Input syntax

Basic usage

You may provide the name of a PCAP trace file as input.

```
wipal-cat input.pcap output.pcap
```

Input concatenation

You may provide the name of several PCAP traces separated with columns (do not include any space). This tells the program to consider the concatenation of each trace as a single input.

```
wipal-cat input1.pcap:input2.pcap:input3.pcap output.pcap
```

will put into ‘output.pcap’ the content of ‘input1.pcap’, followed by the content of ‘input2.pcap’ and then ‘input3.pcap’.

Every program understands this syntax. Note that specifying multiple traces with columns makes no sense for outputs:

```
wipal-cat input1.pcap:input2.pcap output1.pcap:output2.pcap
```

will concatenate ‘input1.pcap’ and ‘input2.pcap’ into a single file named ‘output1.pcap:output2.pcap’!

Address specification

Some programs (e.g. wipal-merge with attributes hsh_en2) might need the IPv4 address of the machine that generated a trace to work properly. Attach such an address to a trace as follows:

```
wipal-merge -a hsh_en2 foo.pcap=192.168.1.1 bar1.pcap:bar2.pcap=192.168.1.2
```

The rationale is that, in some cases, timestamps of emitted frames are not as precise as timestamps of received frames, and thus emitted frames should be ignored during synchronization.

Special characters

When your traces’ filenames contain the special characters : or = they need to be escaped with a backslash (\):

```
wipal-cat weird\=file\:name.pcap out.pcap
wipal-merge -a hsh_en2 weird\=1:weird\:2=192.168.1.1 foo.pcap=192.168.1.2
```

1.2 Concatenation (and Prism noise filtering)

One may concatenate traces using the wipal-cat command. It takes exactly one input and one output. It may be useful to recombine a trace that was split, or filter out frames with Prism noise (using the ‘-P’ option).

```
wipal-cat in.pcap out.pcap
wipal-cat foo.pcap.0:foo.pcap.1 foo.pcap
wipal-cat -P in.pcap out.pcap
wipal-cat -P bar.pcap.0:bar.pcap.1:bar.pcap.2 bar.pcap
```

The first example just copies ‘in.pcap’ into ‘out.pcap’. Note that the two files might be different at the byte level, e.g. if ‘in.pcap’ is big endian and the program is run on a little endian machine.

The second example concatenate ‘foo.pcap.0’ and ‘foo.pcap.1’ and put the result into ‘foo.pcap’.

The third example copies ‘in.pcap’ into ‘out.pcap’ but removes frames that have a non-zero noise field in their Prism headers.

The fourth example both concatenates traces while filtering noisy frames out.

1.3 Comparisons

One may test two PCAP traces for equivalence using the `wipal-cmp` command. The default is to compare every bit of information (PCAP headers plus packet bytes) but you may change this behavior using the ‘-8’, ‘-b’, ‘-o’, or ‘-t’ options. Note that this is different however to using `diff` or `cmp` since traces with different endianness may contain the same packets.

By default `wipal-cmp` produces a report on the standard output indicating either that traces are equal, either which packet is the first to mismatch. Use ‘-q’ if you are only interested in the program’s exit status and do not want to produce any output. Use ‘-o’ if you are interested in counting the number of bytes that differ between two traces.

e.g.:

```
wipal-cmp foo.pcap bar.pcap
wipal-cmp -q foo.pcap bar.pcap
wipal-cmp -q -8 in1.pcap.0:in1.pcap.1 in2.pcap
...
```

1.4 Sub-traces

One may extract sub-traces of PCAP traces using `wipal-extract-subtrace`, `wipal-extract-transmitter`, or `wipal-extract-bssid`.

`wipal-extract-subtrace`

takes two dates and a PCAP trace as inputs, and produces one output. Unfortunately, it does not support any option currently.

`wipal-extract-transmitter`

takes a MAC address and a PCAP trace as input, and produces one output. Its output contains the frames from its input that were transmitted by the given address. Note that the command looks at *transmitters*, not *originators*, e.g. the transmitter of a data frame that crossed the distribution system is the output access point, not the original sender. Also note that some frames do not contain information regarding their transmitters (e.g. MAC acknowledgements) and therefore cannot appear in the output, even if they were effectively sent by the given address.

`wipal-extract-bssid`

works as `wipal-extract-transmitter`, but the MAC address represents a BSSID and the command extracts frames that belong to the corresponding BSS. Again, note that some frames do not contain information regarding their BSS. These frames therefore cannot appear in the output, even if they were effectively belonging to the given BSS.

e.g.:

```
wipal-extract-subtrace 2007-01-01 2008-01-01 \
        in.pcap.0:in.pcap.1 out.pcap

wipal-extract-subtrace \
        "2004-Aug-30 16:59:39.789221" "2004-Aug-30 16:59:39.929872" \
        kalahari-ath2 subtrace.pcap

wipal-extract-transmitter 71:19:9f:6f:71:33 in.pcap out.pcap
wipal-extract-bssid       9b:d2:d7:7f:aa:63 in.pcap out.pcap
```

1.5 Merging

One may merge two IEEE 802.11 traces into one using the `wipal-simple-merge` command.

Use the '`-h`' option to have a description of the command's syntax. It takes two inputs and produce one output. When ran, the merging process starts by synchronizing precisely both inputs (see Section 1.6 [Synchronization], page 9). Then both traces are merged and special care is given not to re-order packets or account duplicate packets twice in the output (that is, packets that are present in both traces appear only once in the output).

This command expects PCAP traces with either Prism headers, AVS headers, Radiotap headers, raw IEEE 802.11 frames, or pseudo-Ethernet II frames as link type. The '`-p`' and '`-P`' options only work with Prism headers. The following timestamps are used, unless '`-y`' is provided:

IEEE 802.11 frames
: PCAP timestamps,

Ethernet II frames
: PCAP timestamps,

Radiotap headers
: Radiotap headers' `tsft` fields. The command will fail with Radiotap headers that do not contain such fields,

AVS headers
: AVS headers' `mactime` fields,

Prism headers
: Prism headers' `mactime` fields.

e.g.:

```
wipal-simple-merge a.pcap b.pcap output.pcap
wipal-simple-merge -P -n foo-ath2.0:foo-ath2.1 bar-ath2 foo-bar-ath2
...
```

1.5.1 Notes regarding traces with Ethernet II frames as link type

See Section 1.7 [Unique frames], page 9.

Since version 4.0, WiPal is able to merge traces with Ethernet II frames as link type. This is useful because some wireless traces use this link type. These traces only contain IP packets encapsulated into pseudo-Ethernet frames.

Since these traces contain no IEEE 802.11 MAC headers one cannot use the usual attributes – that rely on these headers – to merge them. Therefore, use the `hsh_en2` attributes when merging Ethernet II traces (see option '`-a`'). Using theses attributes tell WiPal to decapsulate Ethernet frames and use the following frames as unique frames:

- OLSR packets (IPv4 and IPv6),
- IPv6 router advertisements.

Also note that machines recording PCAP traces while emitting packets generally record imprecise timestamps for emitted packets. In order to solve this issue, you might specify an IPv4 address for each trace (see Section 1.1.2 [Input syntax], page 5). Frames originating from this address in this specific trace will be ignored for synchronization.

Finally, remember that Ethernet traces only contain PCAP timestamps, and these timestamps are not as precise as PHY-level timestamps. You might want to use option '`-x`' to raise the expected precision above 106 microseconds.

1.5.2 Merging more than two traces

`wipal-simple-merge` is only able to merge two traces. In order to merge more traces, one should run successive merges following a given sequence. For instance, merging traces *A*, *B*, and *C* might involve merging *A* and *B* into *T* first, and then merging *T* and *C*. The `wipal-merge` command selects a merging sequence and runs the corresponding merge operations in turn.

e.g.:

```
wipal-merge t1.pcap t2.pcap t3.pcap
wipal-merge -n -P t11.pcap:t12.pcap:t13.pcap t21.pcap:t22.pcap t3.pcap
```

There is no rule to determine which merging sequence will give the "best" results. We consider the two traces that are the most similar should be merged first. This to avoid generating anomalies due to a lack of reference frames (see Section 1.6 [Synchronization], page 9). In order to compute similarity between two traces *A* and *B*, WiPal count the number of reference frames it is able to extract from these traces, stopping when it reaches *B*'s 250,000th unique frame (see Section 1.7 [Unique frames], page 9). Despite its issues, this technique has the advantage of being both simple to implement and fast (determining a merging sequence should not take more time than actually merging the traces).

`wipal-merge` computes its merging sequence as follows. Note that it is designed to be fast rather than to yield an optimal sequence.

1. For each trace, compute its similarity with each other trace.
2. Sort results by similarity.
3. Pick up the most similar result.
   - If it involves two non-merged traces, merge them.
   - If it involves a trace *A* that has already been merged into another trace *T*, consider merging *T* instead of *A*.
   - If it involves two traces that were already merged into the same trace, do nothing.
4. Pick up the next result in the list and repeat step 3 until all traces have been merged into one unique trace.

One may compute the similarity between multiple traces using the `wipal-similarity` command. The output is sorted by ascending order of similarity. e.g.:

```
wipal-similarity t1.pcap t2.pcap
wipal-similarity -P t1.pcap t2.pcap t3.pcap t4.pcap
```

1.6 Synchronization

In order to merge two IEEE 802.11 traces WiPal needs to synchronize them precisely. In order to do so, it first identifies some frames that appear in both inputs. These are reference frames. It uses these frames to model clock desynchronization among the traces. It then update the first trace's timestamps so they are synchronized with the second trace.

One may use the `wipal-synchronize` command to synchronize two traces. It takes two inputs and produce one output. The output contains the same packets as the first input, but with synchronized timestamps.

To extract reference frames WiPal extract some specific frames called *unique frames* (see Section 1.7 [Unique frames], page 9) from both input traces and then intersect the two obtained sets. One may use the `wipal-intersect-unique-frames` command to get the result of this operation (i.e. the list of reference frames used for synchronization of two traces).

WiPal's synchronization process synchronizes reference frames before it synchronizes other frames. One may get the result of this operation using the `wipal-synchronize-unique-frames` command.

e.g.:

```
wipal-intersect-unique-frames -n -P foo.0:foo.1:foo.2 bar.0:bar.1
wipal-synchronize-unique-frames -n -P foo.0:foo.1:foo.2 bar.0:bar.1
wipal-synchronize -n -P foo.0:foo.1:foo.2 bar.0:bar.1 foo-sync
```

1.6.1 Synchronizing more than two traces

Due to WiPal's mode of operation, it is not possible to synchronize multiple traces on a common timeline in a single operation. `wipal-merge-and-synchronize` however provide a similar feature. It behaves as follows:

1. First, merge all the traces given on the command line. At this stage, the command behaves exactly as `wipal-merge`.
2. Then, synchronize each individual trace from the command line with the timeline of the previously merged trace. Record each synchronized trace into the files '`sync-1`', '`sync-2`', etc.

For instance:

```
wipal-merge-and-synchronize t1.pcap t2.pcap t3.pcap
```

will merge '`t1.pcap`', '`t2.pcap`', and '`t3.pcap`' into the file '`merge-2`'. Then each trace will be synchronized using '`merge-2`''s timeline.

1.7 Unique frames

A frame is said to be unique when it appears on the air once and only once for the whole duration of a trace. WiPal's unique frame extraction process is an important stage of its trace synchronization process. WiPal programs' default policy is to consider all beacon frames and all non-retransmitted probe responses as unique frames.

One may use the `wipal-extract-unique-frames` command to get a list of the unique frames that compose a trace. Run `wipal-extract-unique-frames -h` to get its invocation syntax.

In practice, WiPal does not extract and load full unique frames into memory. This would slow the process down and require an excessive amount of memory. The default is to work on MD5 frame hashes when WiPal was compiled using OpenSSL. When compiled without OpenSSL, WiPal only extracts a subset of frame fields. We call the pieces of information WiPal extracts to identify unique frames "frame attributes", or sometimes "frame identifiers".

You may specify frame attributes to use with the '`-a`' option. In practice, the difference in speed and memory consumption between attributes is negligible. There is an important difference between attributes, though. With some attributes, different unique frames may yield identical attributes (collisions). This is of course an undesirable behavior.

One may check that a given trace's unique frames are really unique w.r.t. unique frame attributes using the `wipal-test-uniqueness` command. This command finds collisions inside its input traces. You might specify different frame attributes using the '`-a`' option.

e.g.:

```
wipal-test-uniqueness -P -a tmp foo.pcap.1:foo.pcap.2
wipal-extract-unique-frames -P foo.pcap.1:foo.pcap.2 > foo-unique.txt
```

1.7.1 Special attributes

WiPal's "standard" behavior only considers non-retransmitted IEEE 802.11 probe responses and beacons to compute unique frames. Two "special" attributes however change this behavior:

`hsh_en2`
: These attributes only work with traces using the `EN10MB` PCAP link type (Ethernet II). On the opposite, "standard" attributes do not work with this link type. Using theses attributes tell WiPal to decapsulate Ethernet frames and use the following frames as unique frames:

- OLSR packets (IPv4 and IPv6),
- IPv6 router advertisements.

Beware: if you use traces that last long, OLSR packets' sequence numbers might wrap. This might result in the assumption of OLSR packets being unique not holding anymore. In such cases, you cannot use WiPal to merge your traces.

`hsh_80211_x`
: These attributes work exactly as `hsh_80211` but consider more frames as unique frames. Non-retransmitted IEEE 802.11 probe responses and beacons are still considered unique frames. In addition, WiPal programs decapsulate IEEE 802.11 data frames and consider the same frames as `hsh_en2` (previously described).

Note that these attributes are disabled by default (because it slows the compilation down and for long traces they are less robust than `hsh_80211`). To enable them, use the '`--enable-attributes`' option of WiPal's `configure` script *before* compiling WiPal.

1.8 Duplicate data frames

One may use the `wipal-find-data-dups` command to search some invalid data frames. It looks into traces on a per-sender basis for successive duplicate data frames (it only considers non-retransmitted frames). Such cases should not occur in theory - as it ignores retransmissions, successive data frames from the same sender should at least show variations in their sequence numbers. Surprisingly, some traces contain such anomalies: identical data frames that are not retransmissions and are only spaced by a few milliseconds. We have no explanations why some datasets exhibit those phenomena.

e.g.:

```
wipal-find-data-dups foo.pcap.0:foo.pcap.1:foo.pcap.2
```

1.9 Statistics

`wipal-stats` computes several figures concerning its given input PCAP traces. It displays these figures as plain text on the standard output. You might either interpret them directly or post-process them with some tools, e.g. to generate plots.

Most of the output figures are self-explanatory and therefore will not be mentioned in this manual. Some others need an explanation though:

frames from expired senders
: The computation of some figures needs `wipal-stats` to keep a state for each sender (e.g. its current sequence number). To avoid some measurement artifacts, each state expires after one minute of inactivity from its sender. This counter indicates how many frames were received which sender had expired upon reception of the frame.

sequence gap too large to make sense
: A sequence gap occurs every time a frame is received which sequence number is greater than its sender's previous sequence number plus one. Theoretically, a gap of length N (e.g. receiving frame '`42`' and then frame '`42 + N + 1`') means the sniffer missed '`N`' frames. Sometimes however the gap is too large to make sense (e.g. a gap of 2000 within a window of 500 microseconds). WiPal counts the number of occurrences of these gaps, but otherwise ignores them (e.g. when estimating the number of missed frames).

gap length frequencies
: This gives the frequencies of sequence gap lengths (see above). The data is directly suitable for Gnuplot. Use the `wipal-plot-gaplenfreqs` script to generate the plot using Gnuplot. e.g.

```
wipal-stats foo.pcap > foo.stats
wipal-plot-gaplenfreqs foo.stats freqs.eps "A title"
```

T-Fi plot
: This gives data suitable for Gnuplot to generate a T-Fi plot. Use the `wipal-plot-tfi` script to generate the plot using Gnuplot. e.g.

```
wipal-stats foo.pcap > foo.stats
wipal-plot-tfi foo.stats tfi.eps "A title"
```

One may find an explanation about T-Fi plots in the following paper: *On the fidelity of 802.11 Packet Traces*, A. Schulman, D. Levin, and N. Spring, in the proceedings of PAM 2008.

BSS figures

This gives a list of all BSSs the trace contains as well as a few other figures (e.g. number of distinct BSSs, APs and STAs corresponding to each BSS, etc.) The list is ordered by number of beacons seen for each BSS.

SSID figures

This gives the number of distinct SSIDs the trace contains as well as two lists of these SSIDs. The first one orders them by frequency, the second one orders them lexicographically.

activity This gives data that represents quantity of traffic w.r.t. elapsed time. Each line correspond to one minute. Columns respectively represent:

1. how many frames were sent (during the corresponding minute),
2. how many bytes were sent,
3. how many bytes from management frames were sent,
4. how many bytes from data frames were sent.
5. how many bytes from access points were sent. When a STA emits a beacon which is *not* belonging to an independent BSS (i.e. STA emits an infrastructure mode beacon), WiPal identifies this STA as an access point. All further frames from this STA are accounted as access point traffic.

One might use the `wipal-plot-activity` script to plot traffic rate w.r.t. elapsed time for the whole trace, only for management frames, or only for access point frames. e.g.

```
wipal-stats foo.pcap > foo.stats
wipal-plot-activity foo.stats activity.eps "A title"
```

Various growths (MAC addr., BSSID, IBSSID, SSID, AP)

Actually each “growth” section gives the same kind of statistics, but for various elements. Elements are:

MAC addr.

MAC addresses, without BSSIDs or IBSSIDs. Inspect all frames.

BSSID BSSIDs that are not IBSSIDs. That is, independent BSS frames (i.e. ad hoc mode frames) are ignored. Only inspect beacon frames, despite other frames also contain BSSIDs.

IBSSID IBSSIDs. That is, only account independent BSS frames (i.e. ad hoc mode frames). Also, only inspect beacon frames, despite other frames also contain IBSSIDs.

SSID All SSIDs. Only inspect beacon frames (e.g., ignore probe responses).

AP Sender MAC addresses from beacons. Account both normal BSS frames (infrastructure mode) and independent BSS frames (ad hoc mode).

For a given element type, “growth” data gives statistics about the evolution of the number of distinct elements. Each row represents a minute of measurement. Columns respectively represent:

1. The number of *new* distinct elements seen the last minute.
2. The total number of distinct elements seen since the beginning of the trace.
3. The number of distinct elements seen during the last minute.

For instance, if a trace contains the following elements:

first minute
A B C

second minute
A D

third minute
A B D

The corresponding rows are:

```
3 3 3
1 4 2
0 4 3
```

One might use the `wipal-plot-growth` script to plot an element growth w.r.t. elapsed time. e.g.

```
wipal-stats foo.pcap > foo.stats
wipal-plot-growth "MAC addr." foo.stats mac-growth.eps "A title"
```

ON/OFF events

When a STA emits a frame, `wipal-stats` considers it as active. A STA’s state gets back to inactive after three minutes of silence. The ON/OFF events section lists these state’s changes. The section is composed of one subsection per STA and per trace. Within these subsections, each line indicate a state change. A state change line consists of two columns. The first one indicates the event’s timestamp, and the second one the STA’s new state after the event (0 for inactive and 1 for active).

For instance:

```
begin ON/OFF T2 STA 00:00:00:00:00:42
  0             1
  60000000      0
end ON/OFF T2 STA 00:00:00:00:00:42
```

indicates that, within the third trace (first trace is referred as T0), STA 00:00:00:00:00:42 is active between timestamp 0 and timestamp 60000000.

One might use `wipal-plot-onoff` to generate a PDF file containing a visual representation of this section. Beware this script only gets installed if Python is present on your system, and will only work with a proper Pycairo installation. e.g.

```
wipal-stats foo.pcap > foo.stats
wipal-stats bar.pcap > bar.stats
wipal-plot-onoff foo.stats bar.stats > foo-bar-onoff.pdf
```

per STA counters

For each IEEE 802.11 station, `wipal-stats` maintains various counters. This section lists these counters. It is composed of several subsections which contain the same information sorted differently (e.g. by traffic per STA, by activity periods ("on time"), etc.)

Inside a given subsection, each row contains information about a peculiar station. Each row has the following columns:

1. The MAC address of the station the row is about.
2. Total number of emitted bytes. This includes MAC frames and their payloads.
3. Average rate when on. That is, *size* / *time_on* where *size* is the total number of emitted bytes and *time_on* the total duration the station is active ("on") inside traces. Values are in bytes per microseconds.
4. Total duration the station is active ("on") inside traces. For instance, if a station is active for 3 minutes somewhere at the beginning of the trace and then active for 4 more minutes at another moment in the trace, this column holds 7 minutes. Values are in microseconds.
5. Proportion of stations that have been printed so far. For instance, if the trace contains 10 distinct stations, the first row's value is 0.1, the second 0.2, etc. This is useful for scripts that compute cumulated distributions.
6. Total number of bytes emitted, cumulated with previous rows. This is useful for scripts that compute cumulated distributions.
7. Average rate when on, cumulated with previous rows. This is useful for scripts that compute cumulated distributions.
8. Total number of frames emitted.
9. Average number of frames per microsecond when active. That is, *count* / *time_on* where *count* is the total number of frames emitted. *time_on* has the same meaning as above.

Three scripts use the "per STA counters" section: `wipal-plot-t-dist`, `wipal-plot-t-c-dist`, and `wipal-plot-ot-dist`.

`wipal-plot-t-dist`

Plots the distribution of traffic (and average rate when on) per STA.

`wipal-plot-t-c-dist`

Plots cumulated distributions of traffic (and average rate when on) per STA.

`wipal-plot-ot-dist`

Plots the distribution of total activity periods ("on time").

1.9.1 Plotting scripts

`wipal-plot-all` is a wrapper that that call all of WiPal's plotting scripts. e.g.:

```
     $ wipal-stats foo.pcap > foo.stats
     $ wipal-plot-all foo.stats
     $ ls
     foo.pcap                 foo.stats.I-growth.eps  foo.stats.gaplenfreqs.eps
     foo.stats                foo.stats.M-growth.eps  foo.stats.tfi.eps
     foo.stats.A-growth.eps  foo.stats.S-growth.eps
     foo.stats.B-growth.eps  foo.stats.activity.eps
```

`wipal-plot-activity` and `wipal-plot-growth` use PCAP timestamps for the x axis. Usually, PCAP timestamps use GMT. However, traces are not necessarily recorded in a GMT zone. You might use the `WP_TZ` environment variable to fix this. This variable specifies to WiPal's plot scripts a time adjustment in minutes.

e.g., if you recorded a trace in a GMT-4 zone, plot its statistics with:

```
WP_TZ=$((-4 * 60)) wipal-plot-activity foo.stats
```

1.10 Anonymization

`wipal-anonymize` is a program to anonymize IEEE 802.11 traces. It takes one input and one output: the filename of the trace to anonymize, and the filename of the anonymous trace to produce. The output contains the same frames as the input with the following modifications:

- NIC specific parts of MAC addresses are anonymized.
- ESSIDs are anonymized with a prefix-preserving scheme. For instance, a valid anonymization could map ‘`operator-4251`’, ‘`operator-DODO`’, and ‘`foobar`’ to ‘`abcdefgh*x0yz`’, ‘`abcdefgh*9876`’, and ‘`zxycba`’. The anonymization scheme also preserves character classes, i.e. alpha-numerical characters are anonymized to other alpha-numerical characters, printable characters stay printable, and ASCII extended characters (128 to 256) stay extended.
- Data frames are truncated so the output only contains MAC headers.

`wipal-anonymize` stores valid-to-anonymous MAC and ESSID mappings into files so these mappings can be re-used latter. `wipal-anonymize` also reads these file at start-up when they exist. This enables the creation of distinct anonymous traces with consistent MAC addresses and ESSIDs. By default these mapping files' names are ‘`MAC.map`’ and ‘`ESSID.map`’. Use the ‘`-m`’ and ‘`-s`’ options to change this. See Section 1.1.1 [Available options], page 2.

1.11 Miscellaneous programs

`wipal-list-frames` just list a trace's frames. This is a pretty dumb program, yet one may use it to display a trace's timestamps. e.g.:

```
$ wipal-list-frames foo.pcap | head
1
2
3
```

```
4
5
6
7
8
9
10
$ wipal-list-frames -C -U foo.pcap | head
foo.pcap
Frame ID        Microseconds
========        ============
1               1258703194
2               1258704299
3               1258704368
4               1258705143
5               1258709302
6               1258709362
7               1258709784
```

1.12 Undocumented programs

WiPal's `configure` script has two options '`--enable-probe-stats`' and '`--enable-wit-import`'. These options enable the build of several programs, namely `wipal-probe-stats`, `wit-create-datafiles`, `wit-create-tables-and-load-data`, and `wit-import`. By default the build of those programs is *disabled*.

Those are legacy programs that were useful to somebody once, yet are incomplete and flawed. They will *not* be updated later, and are *not* documented here. Build and use at your own risks!

2 The library

A C++ library also compose WiPal. WiPal programs all use this library. At a low level it provides various convenience tools (PCAP file input/output, random access to PCAP traces, support for various static C++ techniques, etc.) At an upper level it provides a generic IEEE 802.11 frame parser that is easy to customize and re-use. Finally, it provides various mechanisms to synchronize and merge PCAP traces directly from C++ code.

The library is called `libwipal` and its headers are located in `$(prefix)/include/wipal`. You should be able to include them as follows:

```
#include <wipal/pcap/stream.hh>
// ...
```

You will then need to provide the '`-lwipal`' option to your compiling/linking tools.

The main documentation for this library is provided as a Doxygen documentation. It should be installed into WiPal's package data directory, into the '`doxygen`' subdirectory. By default this gives '`/usr/local/share/wipal/doxygen/`'. This documentation is however a bit messy, and lacks some parts. The best entry point to learn how to use the library is to look at some of WiPal's tools' source code (e.g. into '`src/misc/wipal-find-data-dups.cc`'). You may also want to have a look at WScout which is another program that uses WiPal (some versions of WScout embeds WiPal under the name *trace-tools*).

3 FAQ

3.1 What systems does WiPal support?

WiPal was mostly designed using standard C++ and portable libraries. It however uses a few GCC extensions. Yet WiPal should run fine on most systems (e.g. GNU/Linux, WhateverBSD, Mac OS, Windows, ...).

WiPal is however exclusively tested on Debian GNU/Linux (amd64 and, to a lower extent, powerpc). Which means you might experience problems on other systems, which the developers might not be aware of. In this case, please give feedback to them so they can fix it. Anyway, there should be no major obstacle to WiPal's portability.

3.2 What are WiPal's requirements?

WiPal needs:

- GCC.
- The Boost C++ libraries. More specifically:
  - any,
  - array,
  - conversion/lexical_cast,
  - date_time,
  - filesystem,
  - foreach,
  - format,
  - multi_array,
  - optional,
  - preprocessor,
  - smart_ptr,
  - string_algo,
  - tuple.
- The GNU MP Bignum Library,
- OpenSSL.

3.3 How do I install WiPal?

WiPal's packaging follows the GNU conventions. An installation documentation is provided in the '`INSTALL`' file in the package's root directory. However, with a standard system, the following commands should do the trick:

```
mkdir _build
cd _build
../configure
make
make install-strip
make check
```

On some systems, you might have to customize the '`configure`' script's invocation. e.g.:

```
mkdir _build
cd _build
../configure CPPFLAGS=-I/foo/bar/libgmp
make
make install-strip
make check
```

3.4 Are there any options to optimize WiPal when building it?

You might want to compile WiPal with the `NDEBUG` preprocessor symbol defined. If you use GCC you might also want to use its `-O3` option. You can do that by running '`configure`' with the following options:

```
./configure CPPFLAGS=-DNDEBUG CXXFLAGS=-O3
```

3.5 Gee! WiPal's compilation takes long and requires a lot of memory!

WiPal heavily uses static C++ mechanisms and a full build requires instantiating many templates. This results in a long build process that requires much memory. You may disable some template instantiations to have a faster and lighter build process. This will however remove some features at the end. You may invoke `configure` with the following options:

'`--enable-linktypes=LT1:LT2:...`'

will only enable the listed PCAP link types when compiling WiPal. The available link types are:

`IEEE802_11`

raw IEEE 802.11 frames,

`IEEE802_11_RADIO`

Radiotap headers,

`IEEE802_11_RADIO_AVS`

AVS headers,

`PRISM_HEADER`

Prism headers,

`EN10MB` Ethernet II.

'`--enable-attributes=A1:A2:...`'

will only enable the listed unique frame attributes (see Section 1.7 [Unique frames], page 9) when compiling WiPal. The list's first attribute specifier is the default one (when '`-a`' is not provided on the command line). Available attributes are:

- tmp
- seq_tmp

- dst_tmp
- src_tmp
- bss_tmp
- src_bss_tmp
- seq_bss_tmp
- seq_dst_bss_tmp
- seq_src_bss_tmp
- hsh_80211 (requires OpenSSL)
- hsh_80211_x (requires OpenSSL)
- hsh_en2 (requires OpenSSL)

If you know you are going to need only one PCAP link type (e.g. Prism headers), and you do not want to test various attributes, a good choice might be:

```
./configure --enable-linktypes=PRISM_HEADER --enable-attributes=hsh_80211
```

which will only instantiate one template configuration for each WiPal utility.

3.6 Do WiPal's tools have a verbose mode to report extra information about their operation?

There is no such options that can be activated dynamically. You might want however to compile WiPal with the `WP_ENABLE_INFO` preprocessor symbol defined. This will enable the printing of some extra information in some tools as they run (e.g. number of processed frames, synchronization error, etc.). Invoke the '`configure`' script with the following option:

```
./configure CPPFLAGS=-DWP_ENABLE_INFO
```

Note however that this may slow some tools down and may require more memory.

3.7 You say WiPal is flexible and customizable. Is there a way to customize WiPal's tools beyond the options they propose?

Yes! But this requires recompiling WiPal's tools, and sometimes modifying a few lines of their source code.

- You may change WiPal's linear regression window (for trace synchronization) by defining the `WP_LRSYNC_WINDOW_SIZE` macro symbol. Use the `CPPFLAGS` environment variable for this. The default value is 4.

  e.g.:

  ```
  ./configure CPPFLAGS='-DWP_LRSYNC_WINDOW_SIZE=42'
  ```

- You may change the windowed merging algorithm's window size by defining the `WP_WMERGE_WINDOW_SIZE` macro symbol. Use the `CPPFLAGS` environment variable for this. The default value is 3.

  e.g.:

  ```
  ./configure CPPFLAGS='-DWP_WMERGE_WINDOW_SIZE=42'
  ```

- You may change the frame attributes (i.e. frame identifiers) to use in tools that do not support the '`-a`' option by modifying a few lines of their source code. This generally needs changing an include and a typedef, e.g.:

```
-#include <wipal/wifi/unique_id/seqctl_bssid_timestamp.hh>
+#include <wipal/wifi/unique_id/seqctl_source_bssid_timestamp.hh>

// ...

-       typedef wifi::seq_bss_tmp_id              unique_id;
+       typedef wifi::seq_src_bss_tmp_id          unique_id;
```

3.8 ‘configure’ complains it did not find library *X*?

Either library *X* is not installed on your system, either your system is not properly configured, so the library cannot be found.

You may use the CPPFLAGS and LDFLAGS variables to correct this behavior.

e.g., run

```
./configure CPPFLAGS=-I/custom/path/include \
            LDFLAGS=-L/custom/path/lib
```

3.9 ‘configure’ complains it found library *X*’s headers, but is unable to link?

Most probably library *X* is installed but its binaries are in a non-standard place. Use the LDFLAGS variable as described previously.

3.10 ‘configure’ complains library *X*’s headers are unusable, despite successful linking?

Most probably library *X* is installed but its headers are in a non-standard place. Use the CPPFLAGS variable as described previously.

3.11 Do you have a list of WiPal’s bugs?

No. We are not aware of any serious bug in WiPal. We take a special care at testing WiPal with an automated test suite. Do not hesitate to report unknown bugs to the package’s maintainers. We will hunt them.

With some tools, you might however encounter some strange behaviors when providing invalid inputs (e.g. running `wipal-find-data-dups a:b` with ‘b’ having a link type different from ‘a’). Consider that as a “feature”! ;-)

3.12 I have found a bug, what should I do?

Report it to the package’s maintainers.

3.13 I would really love having feature X implemented!

Give feedback to the package’s maintainers about the features you want. We might not have the time to implement them, yet it is important for us to know when important features are missing.

Regarding features you miss, you are greatly encouraged to contribute to WiPal. Again, contact the package’s maintainers so they can help you implement new features.

3.14 I have a question this file did not answer!

Mail the package's maintainers.

Index

Program index

Concept index