

---

# **pcapgraph Documentation**

*Release 1.3.2*

**Ross Jacobs**

**Nov 08, 2018**



---

## Contents:

---

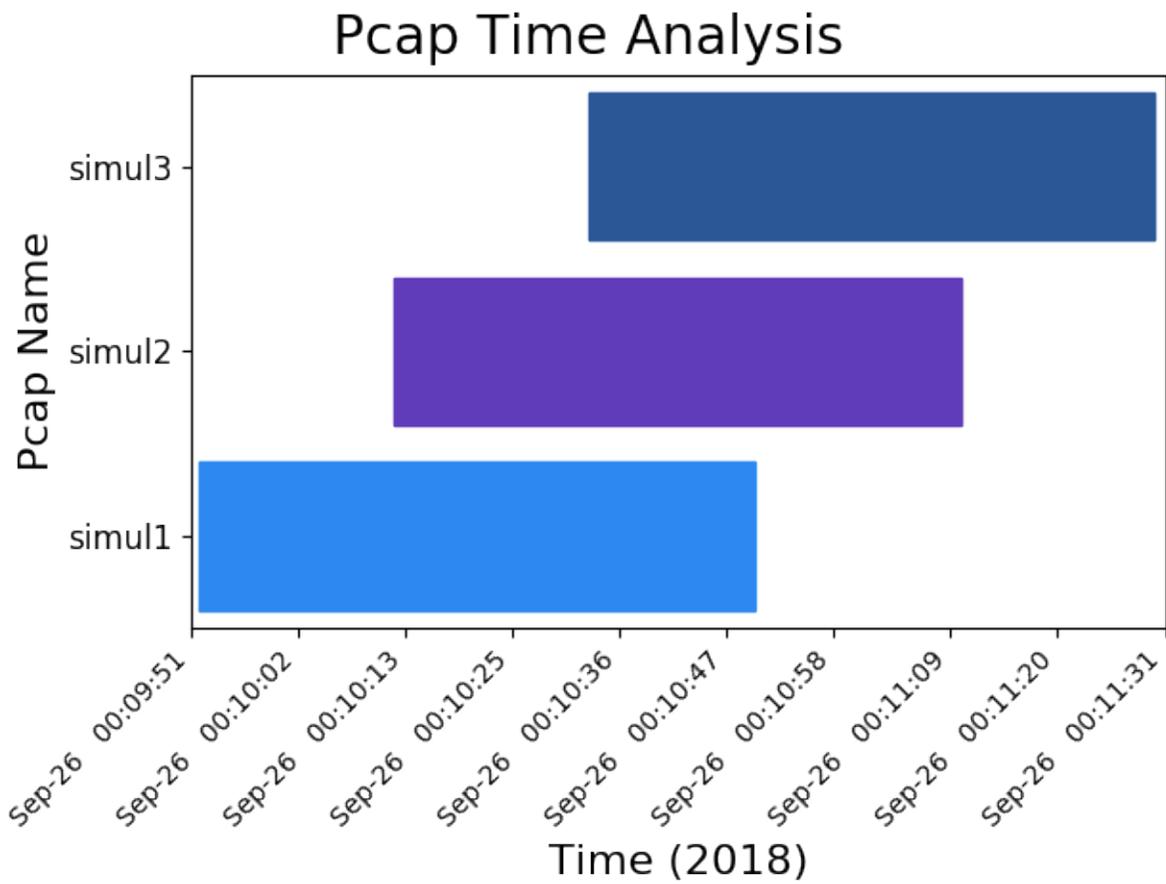
<b>1</b>	<b>About</b>	<b>3</b>
1.1	Platforms . . . . .	4
1.2	Description . . . . .	4
1.3	License . . . . .	4
	<b>Python Module Index</b>	<b>35</b>



Create bar graphs with [packet capture](#) timestamps.

---





*Three packet captures taken of the same network traffic, staggered by 20 seconds.*

## 1.1 Platforms

Linux, macOS, Windows

## 1.2 Description

- Assists with flow-based troubleshooting where there are at least 3 pcaps. See [Usage](#) for detailed use cases.
- Create a horizontal bar graph to visualize when pcaps were taken.
- Use set operations to find patterns among multiple packet captures in ways that Wireshark is not able to.
- If an output format is not specified, the default behavior is to print to stdout and send a [matplotlib](#) graph to the screen (thus the name).

## 1.3 License

Apache 2.0

---

### 1.3.1 Install

*Python is not the only language being used for network automation but the combination of being an easy to learn language with many code samples and utilities has made it a go-to language for network engineers.*

—Cisco DevNet

---

#### Prerequisites: Wireshark

- These package managers have it in their repositories: `apt`, `dnf`, `pacman`, `brew`, `choco`, ...
  - To download and install precompiled binaries, visit [Wireshark's website](#).
- 

#### Installing PcapGraph

##### -> Install PcapGraph with Python pip

```
pip install --user pcapgraph
```

#### Notes:

- This project requires python3.5 or later. `pip` is bundled with Python starting with python3.4.
- You can check your version of Python with `python -V` in a terminal.
- To download and install precompiled Python binaries, visit [python's website](#).

- macOS comes with Python 2.7 by default. If installing python separately, make sure to add alias 'python=python3' to your .bashrc.

### -OR- Install from source

```
git clone https://github.com/poccc/pcapgraph
cd pcapgraph
make install
```

### -OR- Use the standalone executable

Download the latest executable for your OS from the [releases page](#) and run:

```
./pcapgraph
```

---

**Note:** Using an executable will have startup lag when PyInstaller extracts required files to a temporary folder.

---

## Install Errors

---

**Note:** These are some misconfiguration errors I came across during testing on Ubuntu. If you have trouble installing, please create an [issue](#).

---

### \_tkinter not installed

On ubuntu, you may need to install the `python3-tk` package to use the tkinter parts of matplotlib.

### ImportError: cannot import name 'multiarray'

If you have versions of numpy or matplotlib that were installed on different minor versions of python, you may need to reinstall both.

```
python -m pip uninstall -y numpy matplotlib
python -m pip install --user numpy matplotlib
```

## Testing Install

Test whether pcapgraph is working:

```
pcapgraph -V
```

## 1.3.2 CLI Usage

### PcapGraph args

**PcapGraph:** Create bar graphs out of packet captures.

#### USAGE:

```
pcapgraph [-abdeisuvwx23] (<file>)... [--output <format>]...
pcapgraph (-V | --version)
pcapgraph (-h | --help)
```

#### DESCRIPTION:

Analyze packet captures with graphs and set operations. Graphs will show the temporal overlap of packets. Set operations can help with flow-based troubleshooting across multiple interfaces or devices. The default behavior for output is a graph (hence the name).

Official documentation: <https://pcapgraph.readthedocs.io/>

#### OPTIONS:

##### SET OPERATIONS:

- b, --bounded-intersection** Bounded intersection of packets.
- d, --difference** First packet capture minus packets in all succeeding packet captures.
- e, --inverse-bounded** Shortcut for applying *-b* to a group of pcaps and then subtracting the intersection from each.
- s, --symmetric-difference** Packets unique to each packet capture. (see Set Operations > symmetric difference).
- u, --union** All unique packets across all packet captures. (see Set Operations > union).
- i, --intersection** All packets that are shared by all packet captures (see Set Operations > intersection).

##### OUTPUT OPTIONS:

- a, --anonymize** Anonymize packet capture file names with fictional place names and devices.
- o, --output <format>** Output results as a file with format type.
- w** Open pcaps in Wireshark after creation. (shortcut for `--output pcap --output wireshark`)
- x, --exclude-empty** eXclude empty pcaps generated by a set operation from being saved. Exclude empty input pcaps from being graphed.
- 2, --strip-l2** Remove layer2 bits and encode raw IP packets. Use if pcaps track flows across layer 3 boundaries or L2 frame formats differ between pcaps (e.g. An AP will have Ethernet/Wi-Fi interfaces that encode 802.3/802.11 frames).
- 3, --strip-l3** Remove IP header and encode dummy ethernet/IP headers. Use if pcaps track flows across IPv4 NAT. -3 implies -2. This flag is IPv4 only as IPv6 should not have NAT.

##### MISC OPTIONS:



```
$ pcapgraph --intersect --strip-l3 file1.pcap file2.pcap --output pcap
```

Imagine that you are troubleshooting an issue on a natting firewall and you are looking at traffic on lan and wan ports. Using strip-l3 with intersect will find all common traffic, even though NAT changes various values. strip-l3 replaces all l3 fields that would change with generic values. Export this traffic as pcap to review in wireshark.

Note that traffic will look like the following in wireshark:

```
<RAW IP> 10.0.0.1 -> 10.0.0.2 ICMP echo reply DATA1
<RAW IP> 10.0.0.1 -> 10.0.0.2 UDP 16298 -> 53 DATA2
<RAW IP> 10.0.0.1 -> 10.0.0.2 TCP 28274 -> 80 DATA3
```

#### 4. Difference between traffic on a switchport and the uplink

```
$ pcapgraph --difference switch_uplink.pcap switchport3.pcap
```

Find all packets in switch\_uplink.pcap that are also in switchport3.pcap and remove them. This might be helpful if you know that all traffic coming out of switchport3 is noise for what you are looking for. Difference is generally helpful as another means to filter pcaps.

#### 5. Union to help diagnose a broadcast storm:

```
$ pcapgraph --union pcap_dir/
```

Union, without an output format defaults to a matplotlib graph and text to stdout. This text will contain the ASCII hexdump of the 10 most common packets along with their count. Knowing what the MAC (and potentially IP) of the causative packets may be helpful in identifying a switching loop.

Use spanning tree and set a root bridge once you have figured it out.

#### 6. Find unique traffic in the same timeframe across all pcaps

```
$ pcapgraph file1.pcap file2.pcap --inverse-bounded -w
```

Assume that you are looking at two packet captures: file1.pcap that has pings to a remote destination and file2.pcap that should have those pings, but doesn't. You know that there will be other traffic on this link like TCP, HTTP, etc. Normally, you might find an ip.id of a packet early in one packet capture and search for it in the other with 'ip.id==0xabcd' for example. Then find the latest packet in both using the same method and then filter both packet captures by frame number. This function automates that process.

Finds all traffic in the bounding intersection that is unique to each packet capture and opens all of them in wireshark.

**SET OPERATIONS:** All set operations require packet captures and do the following:

1. Find all unique packets by their ASCII hexdump value.
2. Strip L2 and L3 headers if those options are specified
3. Apply the operation and generate a list of packets.
4. Reencode the packets in a pcap using text2pcap.

**difference:** Remove all packets that are present in one pcap from another.

**intersection:** Find all packets that two pcaps have in common.

**union:** Find all unique packets found in all provided pcaps.

**symmetric difference:** Find all packets that are unique to each pcap.

**bounded (time intersection):** Find the first and last frames in the frame intersection of all pcaps according to their timestamp Use these two frames as upper and lower limits to return all frames in each pcap that are between these two frames. This can help to identify traffic that should be in both packet captures, but is in only one.

**inverse bounded (time intersection):** Finds which packets are unique to each packet capture in a given time frame and saves each as a packet capture.

See also:

**pcapgraph** (<https://pcapgraph.readthedocs.io>): Comprehensive documentation for this program.

**wireshark** (<https://www.wireshark.org/>): Read packet captures to troubleshoot networks.

**wireshark utils** (<https://www.wireshark.org/docs/man-pages/>): CLI utils that contain or enhance wireshark functionality. These were used in PcapGraph: editcap, mergecap, reordercap, text2pcap, tshark

**pyshark** (<https://kiminewt.github.io/pyshark/>): Python wrapper for tshark.

**scapy** (<https://scapy.readthedocs.io/en/latest/>): Python program to manipulate frames.

**matplotlib** (<https://matplotlib.org/>): Python package to plot 2D graphs.

### 1.3.3 Pcap Preparation

This program will be most useful if packet captures are filtered for relevant traffic. The smaller the packet captures are, the faster pcapgraph is at processing them and the easier it will be to draw conclusions from exported graphs and packet captures.

#### Filtering for Relevant Traffic

tshark is a utility bundled with Wireshark that can use filter a pcap with display filters and save to pcap.

```
tshark -r <in.pcap> -Y "<display filter>" -w <out.pcap>
```

For example, to filter for ICMP traffic going to/from Cloudflare's DNS service, use "icmp && ip.addr==1.1.1.1" in place of "<display filter>".

More information about tshark usage can be found on the [tshark manpage](#).

#### Decreasing the Size of the Packet Capture

If your packet capture is very large even after filtering, you may want to split it into multiple files.

```
editcap -c <packets per split> source.pcap split.pcap
```

Additional examples can be found at [Packet Life](#).

#### Modifying Timestamps

Sometimes, packet captures are taken by devices whose system clocks are off. If you took the packet capture on a unix-like system, you can get the time offset with `ntptime -q time.nist.gov`.

To modify a packet capture to have the correct timestamps, use editcap:

```
editcap -t <offset> <infile> <outfile>
```

More information about editcap usage can be found on the [editcap man page](#).

### 1.3.4 Using PcapGraph

---

**Note:** *examples/* contains all packet captures, pngs and txt files used as examples here. You can get the *examples/* directory by cloning this repo. You can also use `pcapgraph --output generate-pcaps`, which will generate the starting pcaps for you (and then follow the commands below to create the desired file).

---

#### About

All set operations use the raw frame's hex value to determine uniqueness. This ensures that unless ARP traffic is involved (which has relatively few fields), unique frames are going to be correctly identified as such.

---

**Tip:** These set operations are most useful when packet captures have already been filtered for the traffic that is most relevant. See [Pcap Preparation](#) for more details.

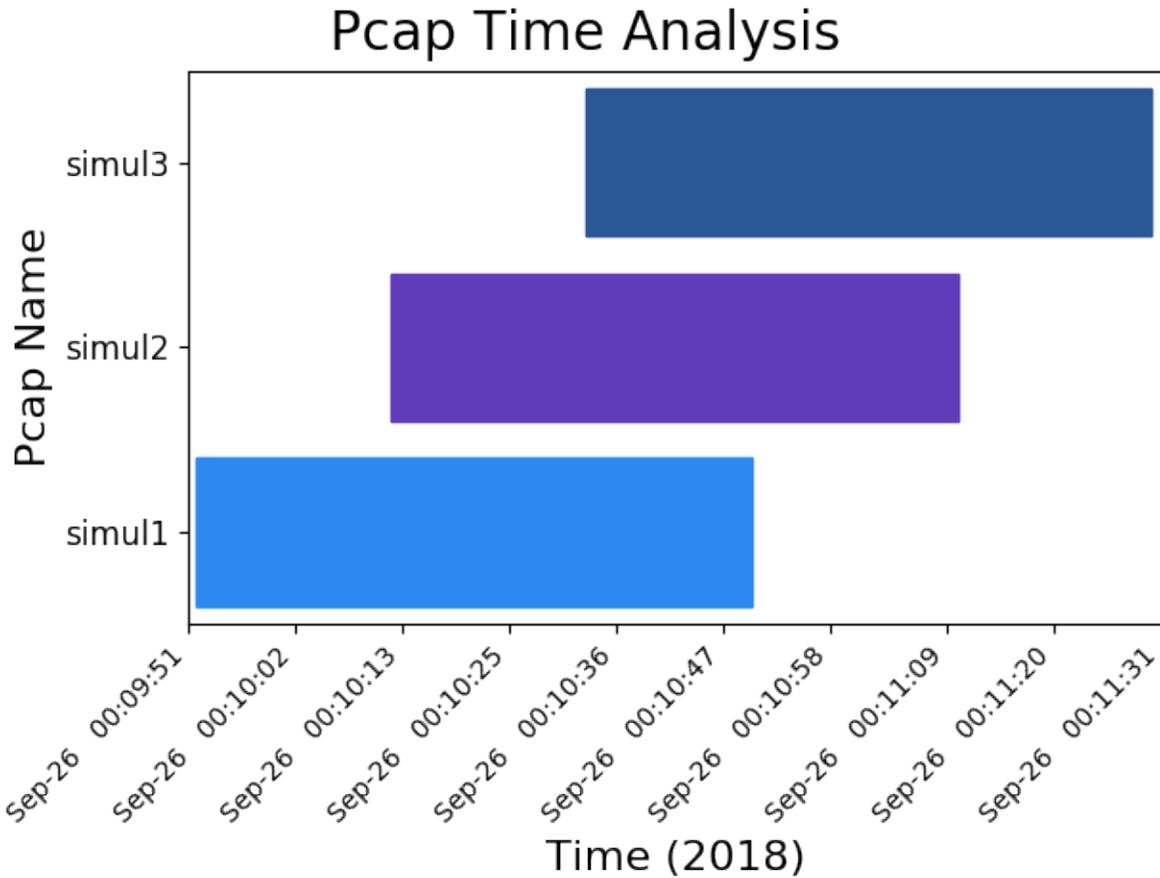
---

#### Gut check: Visualize your packet captures

```
pcapgraph examples/ --output png --output txt
```

#### Default Image

Quickly check whether pcaps were taken around the same time with a graph. Let's say that it is necessary for packet captures to be of the same traffic, taken on different interfaces. If it is clear from a graph that pcaps were taken on different days, then you've saved yourself time looking at pcaps. In this scenario, you might ask for additional pcaps that do or do not demonstrate the issue you are troubleshooting.



#### Default Text

Produces the same data as above, but in text.

PCAP NAME	DATE 0	DATE \$	TIME 0	TIME \$	UTC 0	UTC \$
simul1	Sep-26	Sep-26	00:09:52	00:10:49	1537945792.6673348	1537945849.
↪9369159						
simul2	Sep-26	Sep-26	00:10:12	00:11:11	1537945812.7556646	1537945871.
↪086899						
simul3	Sep-26	Sep-26	00:10:32	00:11:30	1537945832.8390837	1537945890.
↪855496						

#### Default Pcap

**Does not exist:** no set operations are specified.

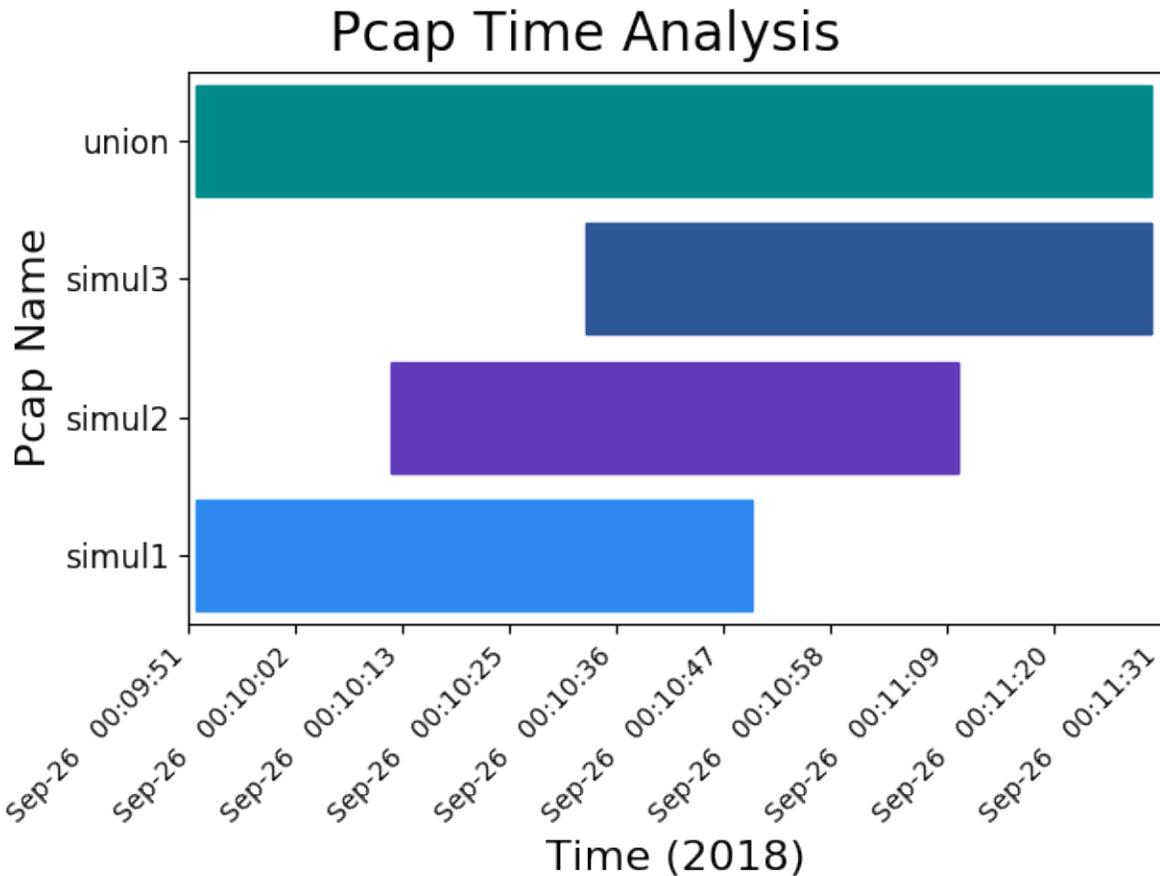
#### Union: Troubleshoot broadcast storms

Union will include all unique packets, and so will include the first and last packets of all captures.

## Union Image

```
pcapgraph examples/ --union --output png
```

Union image is not very useful as its bar will always span the graph.



## Union Text

```
pcapgraph examples/ --union --output txt
```

For a packet capture that contains a broadcast storm, this function will find unique packets and packet counts. This information will not be directly useful because a switching loop, once started, doesn't depend on the instigators. However, it may point your troubleshooting in the right direction to help find the loop.

Use the `-union` of pcaps to find the most frequent packets among all packet capture(s). By default, using the union flag will print the top ten most common frames in ASCII hexdump format to stdout along with their count:

```
Count: 3
0000 88 15 44 ab bf dd 24 77 03 51 13 44 08 00 45 00
0010 00 54 7b af 40 00 40 01 92 2a 0a 30 12 90 08 08
0020 08 08 08 00 ae 46 62 8b 00 01 e8 30 ab 5b 00 00
0030 00 00 88 cd 0c 00 00 00 00 10 11 12 13 14 15
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
```

(continues on next page)

(continued from previous page)

```
0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060 36 37
```

Count: 3

```
0000 88 15 44 ab bf dd 24 77 03 51 13 44 08 00 45 00
0010 00 38 20 40 00 00 40 11 b2 b5 0a 30 12 90 0a 80
0020 80 80 ba dc 00 35 00 24 cb 35 a3 f6 01 00 00 01
0030 00 00 00 00 00 00 06 61 6d 61 7a 6f 6e 03 63 6f
0040 6d 00 00 01 00 01
```

Count: 3

```
0000 24 77 03 51 13 44 88 15 44 ab bf dd 08 00 45 00
0010 00 68 f7 f9 40 00 40 11 9a cb 0a 80 80 80 0a 30
0020 12 90 00 35 ba dc 00 54 1e c2 a3 f6 81 80 00 01
0030 00 03 00 00 00 00 06 61 6d 61 7a 6f 6e 03 63 6f
0040 6d 00 00 01 00 01 c0 0c 00 01 00 01 00 00 00 15
0050 00 04 b0 20 67 cd c0 0c 00 01 00 01 00 00 00 15
0060 00 04 cd fb f2 67 c0 0c 00 01 00 01 00 00 00 15
0070 00 04 b0 20 62 a6
```

Count: 3

```
0000 24 77 03 51 13 44 88 15 44 ab bf dd 08 00 45 20
0010 00 54 ef c6 00 00 79 01 24 f3 08 08 08 08 0a 30
0020 12 90 00 00 b6 46 62 8b 00 01 e8 30 ab 5b 00 00
0030 00 00 88 cd 0c 00 00 00 00 00 10 11 12 13 14 15
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060 36 37
```

Count: 3

```
0000 88 15 44 ab bf dd 24 77 03 51 13 44 08 00 45 00
0010 00 54 7b fa 04 00 40 01 91 df 0a 30 12 90 08 08
0020 08 08 08 00 74 29 62 93 00 01 e9 30 ab 5b 00 00
0030 00 00 c1 e2 0c 00 00 00 00 00 10 11 12 13 14 15
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060 36 37
```

Count: 3

```
0000 88 15 44 ab bf dd 24 77 03 51 13 44 08 00 45 00
0010 00 38 20 8b 00 00 40 11 b2 6a 0a 30 12 90 0a 80
0020 80 80 ea ea 00 35 00 24 69 94 d5 89 01 00 00 01
0030 00 00 00 00 00 00 06 61 6d 61 7a 6f 6e 03 63 6f
0040 6d 00 00 01 00 01
```

Count: 3

```
0000 24 77 03 51 13 44 88 15 44 ab bf dd 08 00 45 00
0010 00 68 f7 fc 40 00 40 11 9a c8 0a 80 80 80 0a 30
0020 12 90 00 35 ea ea 00 54 bd 23 d5 89 81 80 00 01
0030 00 03 00 00 00 00 06 61 6d 61 7a 6f 6e 03 63 6f
0040 6d 00 00 01 00 01 c0 0c 00 01 00 01 00 00 00 14
0050 00 04 b0 20 62 a6 c0 0c 00 01 00 01 00 00 00 14
0060 00 04 b0 20 67 cd c0 0c 00 01 00 01 00 00 00 14
0070 00 04 cd fb f2 67
```

Count: 3

```
0000 24 77 03 51 13 44 88 15 44 ab bf dd 08 00 45 20
```

(continues on next page)

(continued from previous page)

```
0010 00 54 f1 7a 00 00 79 01 23 3f 08 08 08 08 0a 30
0020 12 90 00 00 7c 29 62 93 00 01 e9 30 ab 5b 00 00
0030 00 00 c1 e2 0c 00 00 00 00 00 10 11 12 13 14 15
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060 36 37
```

Count: 3

```
0000 88 15 44 ab bf dd 24 77 03 51 13 44 08 00 45 00
0010 00 54 7c 4e 40 00 40 01 91 8b 0a 30 12 90 08 08
0020 08 08 08 00 8e 09 62 9f 00 01 ea 30 ab 5b 00 00
0030 00 00 a6 f6 0c 00 00 00 00 00 10 11 12 13 14 15
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060 36 37
```

To view the content of these packets, subtract the count lines, add and save to <textfile>, and then run

```
text2pcap <textfile> out.pcap
wireshark out.pcap
```

## Union Pcap

```
pcapgraph examples/ --union --output pcap
```

This pcap can be useful for any situation where you need to find all unique packets. This function can be lossy with timestamps as duplicate packets are excluded, so information can be lost.

**Union file:** *examples/set\_ops/union.pcap*

---

**Tip:** If you want to combine pcaps without loss of duplicate packets, use mergecap instead. mergecap is included by default in Wireshark installations.

```
mergcap (<file>) [<file>...] -w union.pcap
```

---

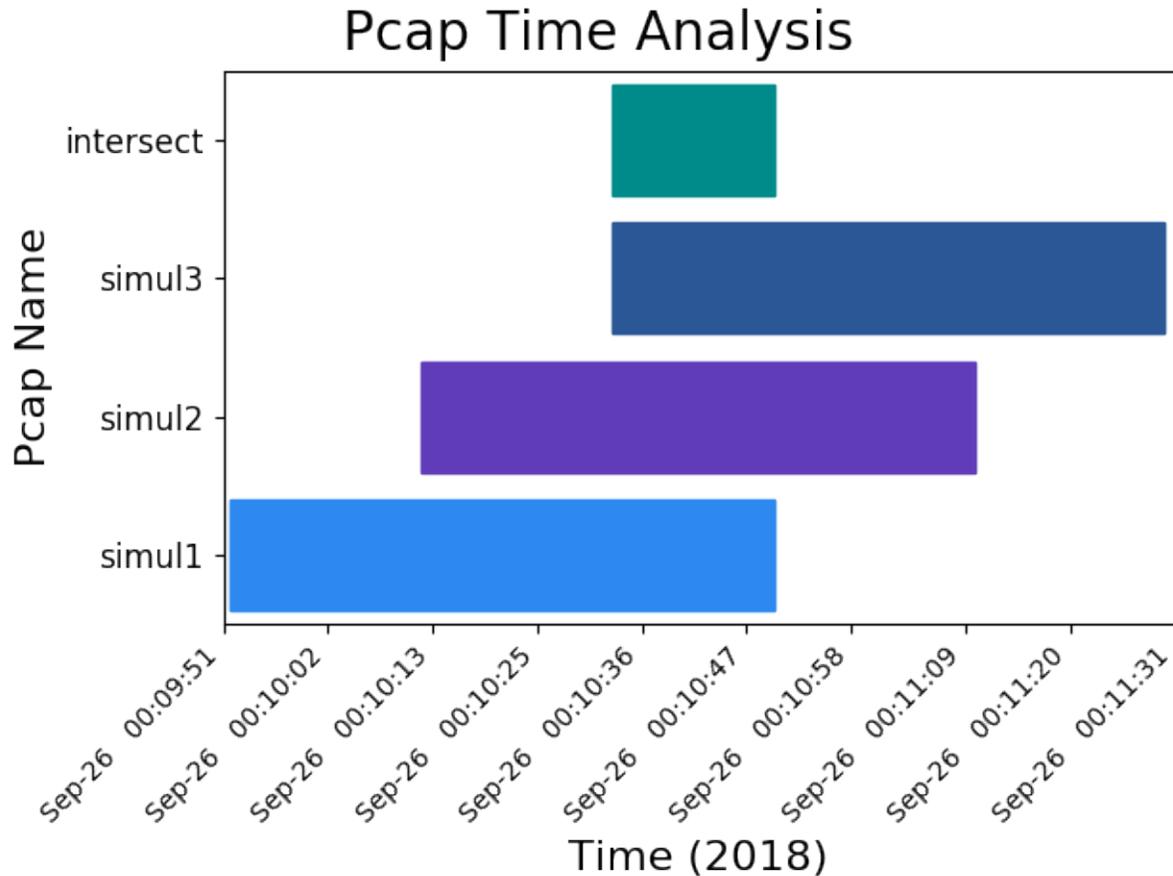
## Intersection: Find common traffic

Find all packets that are shared between all packet captures.

## Intersection Image

The image produced in the graph can be useful in identifying where and at what times frame overlap is occurring.

```
pcapgraph examples/ --intersect --output png
```



### Intersection Text

Intersection text will provide the percentage of packets that are the same across multiple packet captures. Especially if packet captures are filtered before sending to PcapGraph, this can be used to determine what percent of traffic is failing across multiple interfaces in flow-based troubleshooting.

Intersection will alert you if the intersection has no packets.

SAME %	PCAP NAME
31%	examples/simul1.pcap
31%	examples/simul2.pcap
31%	examples/simul3.pcap

### Intersection Pcap

```
pcapgraph examples/ --intersect --output pcap
```

Taking the intersection of multiple packet captures can provide information on what traffic has made it through all relevant devices/interfaces. Given pcaps A-F, where A and F are the endpoints, you can find all packets that have made it from A to F and all points in between.

**Intersection file:** *examples/set\_ops/intersect.pcap*

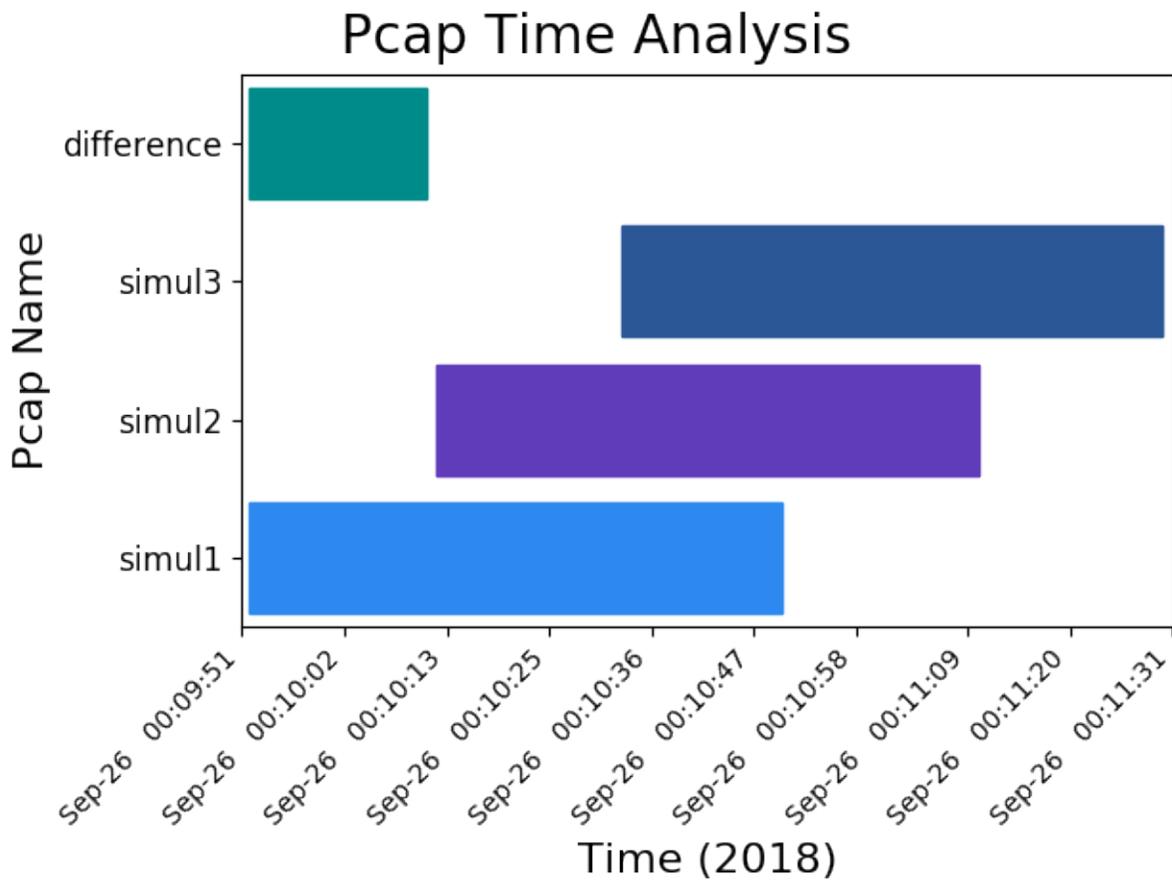
### Difference: Remove shared packets

Find all packets that are unique to the first packet capture.

### Difference Image

The difference image can be useful in telling at what time shared traffic between two packet captures starts or stops.

```
pcapgraph examples/ --difference --output png
```



### Difference Text

Difference will alert you if the difference has no packets (i.e. the minuend packet capture is a subset of the remaining packet captures).

```
pcapgraph examples/ --difference --output txt
```

## Difference Pcap

```
pcapgraph examples/ --difference --output pcap
```

Taking the difference between two packet captures can help find traffic of interest that is present in one packet capture, but not another.

**Difference file:** *examples/set\_ops/diff\_simul1-simul3.pcap*

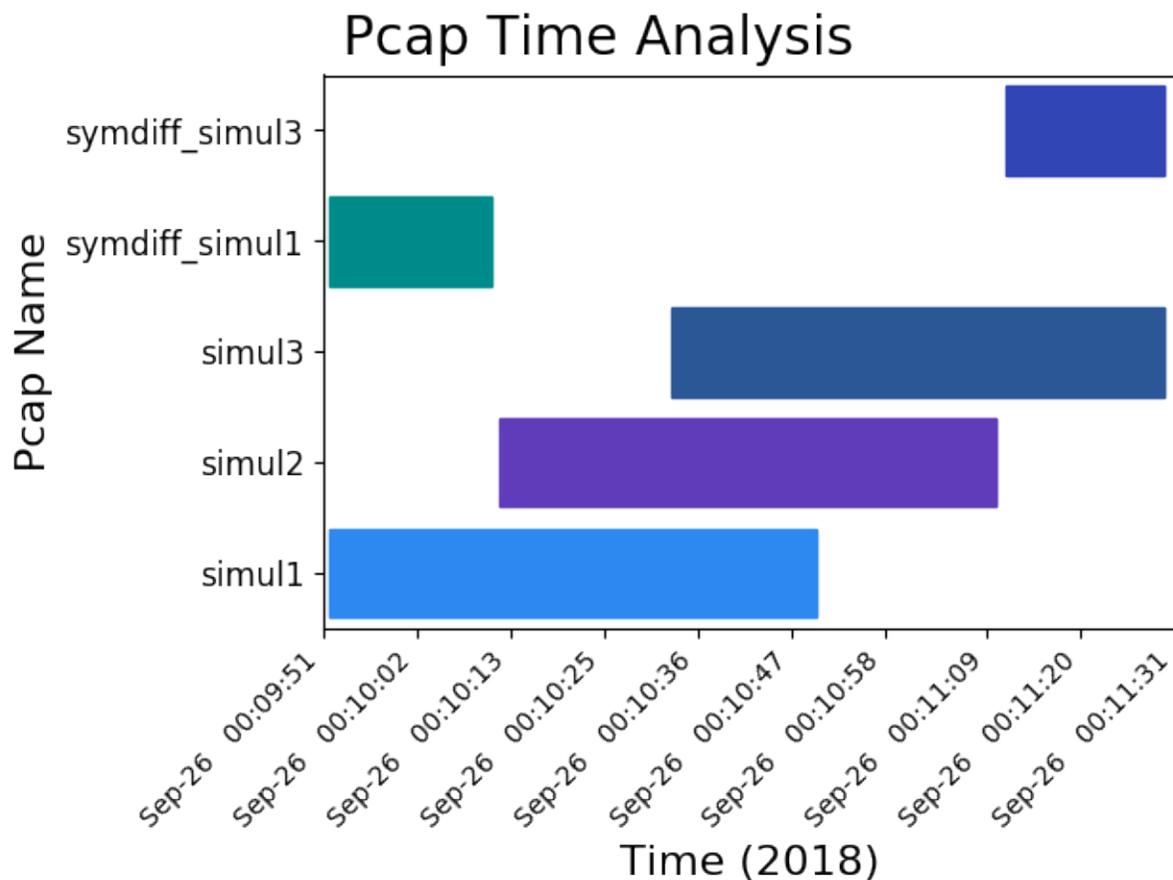
## Symmetric Difference

The symmetric difference includes only unique packets from each packet capture.

## Symmetric Difference Image

The symmetric difference is essentially the difference applied between the first packet capture and every successive one.

```
pcapgraph examples/ --syndiff --output png
```



## Symmetric Difference Text

Doesn't produce any text; however will alert if a packet capture has no unique packets.

## Symmetric Difference Pcap

```
pcapgraph examples/ --difference --output pcap
```

The symmetric difference can help identify which packet captures have unique traffic and exactly what that is. This can be useful if you have multiple packet captures in which you want to get all unique packets exported on a per-packet capture basis.

**Difference file:** *examples/set\_ops/symdiff\_simul1.pcap examples/set\_ops/symdiff\_simul3.pcap*

---

## Timebounded Intersection

### Description

It is sometimes useful when doing flow-based troubleshooting to find all packets between the earliest shared frame and the latest shared frame. It may also be useful to find all traffic that is between two timestamps. These time-bounded operations are built with, but are not bound by the constraints of set operations.

### Example Operation

Let 2 packet captures have the following packets and assume that traffic originates behind the device that Initial 1 is capturing on:

The algorithm will find that packet A is the earliest common packet and that G is the latest common packet.

Initial 1	Initial 2	Intersect	TB Intersect 1	TB Intersect 2
A	W	A	A	A
B	X	B	B	B
C	A	C	C	F
D	B	F	D	M
E	F	G	E	C
F	M		F	G
G	C		G	
H	G			
I	L			

(TB = Time-bounded)

---

### Note:

- In Pcap2, M does not exist in Pcap1
  - In Pcap2, C and F are out of order compared to Pcap1
  - The intersection does not include these interesting packets that are in one pcap, but note the other.
-

## Timebound Intersection Text

**Does not exist:** None created.

## Timebound Intersection Pcap

*Trim packet captures to a timeframe*

Create a packet capture intersection out of two files by finding the first and last instances of identical frames in multiple packet captures. This is something that you might manually do by finding a shared ip.id at the top of both packet captures and the ip.id at the bottom of both packet captures and then filtering out all traffic not between the frame numbers corresponding to the packets with those ip.ids.

This function automates the described manual process.

```
pcapgraph examples/ --bounded-intersect --output pcap
```

## Inverse Timebounded Intersection

### Description

The difference of the intersection and the time-bounded intersection for each packet capture. By definition, the intersection and time-bounded intersection have the exact same starting and ending packets. What may be useful for troubleshooting is determining in that timeframe which packets are different across pcaps and why.

### Example operation

Initial 1	Initial 2	Intersect	Inv TB Intersect 1	Inv TB Intersect 2
A	W	A	D	M
B	X	B	E	
C	A	C		
D	B	F		
E	F	G		
F	M			
G	C			
H	G			
I	L			

(Inv TB = Inverse Time-bounded)

The key here is to subtract the intersection from each initial packet capture to find the interesting packets that are unique to each during the intersection time period.

## Inverse Timebound Intersection Text

**Does not exist:** None created.

## Inverse Timebounded Intersect Pcap

*Find what interface traffic fails at*

Use the inverse bounded intersection to find traffic that occurred between two frames in all packet captures, but is not shared between all of pcaps. This can be useful when troubleshooting a flow to determine where it fails.

```
pcapgraph examples/ --inverse-bounded --output pcap
```

---

## Have fun with your Downloads folder

If you take a lot of packet captures, you can use pcapgraph to visualize your Downloads folder. Use `pcapgraph --dir ~/Downloads` to see what it looks like! (It may take a while to process hundreds of packet captures).

**bash on Linux/Macos:**

```
pcapgraph ~/Downloads/
```

---

**command prompt on Windows:**

```
pcapgraph %USERPROFILE%\\Downloads
```

---

## Examples of all output formats

---

**.pcap: Use all 6 set flags**

```
pcapgraph examples/ -bdeisu --output pcap
```

---

### Output

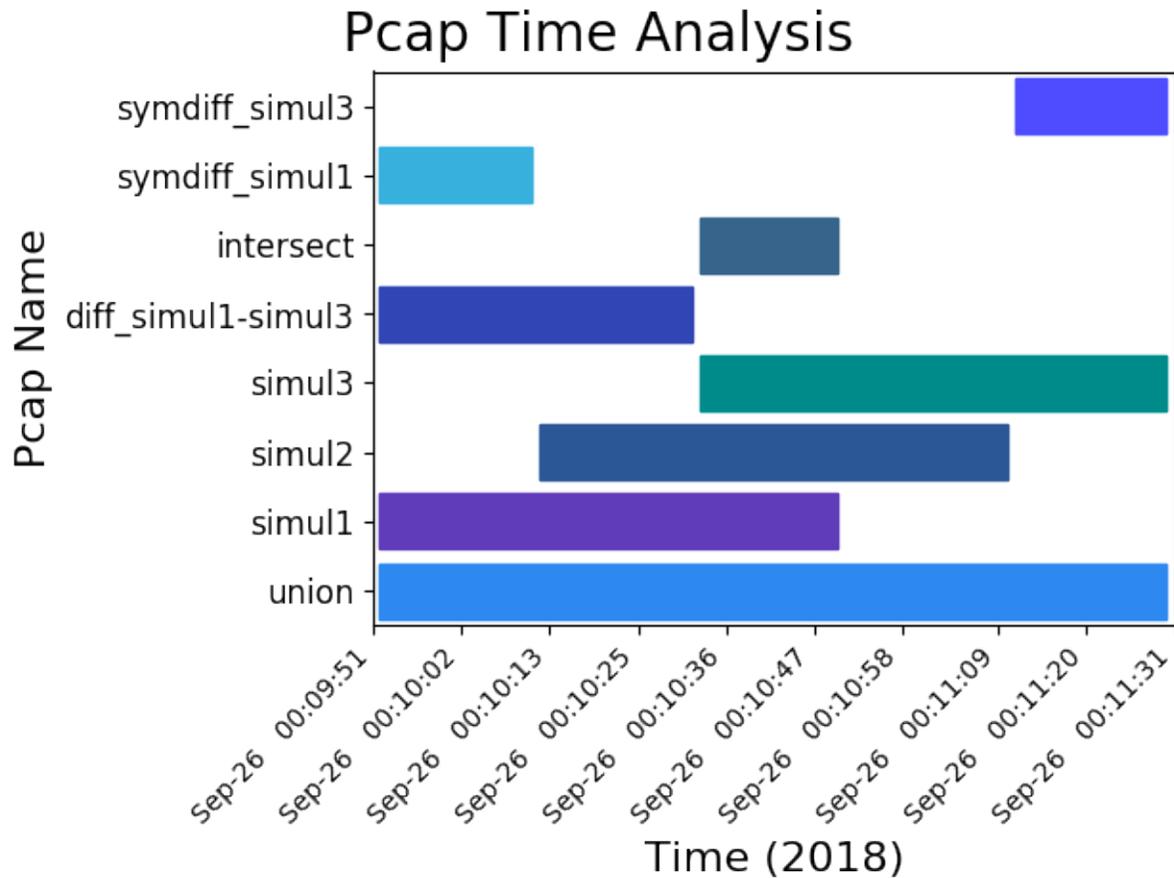
```
bounded_intersect-simul1.pcap
bounded_intersect-simul2.pcap
bounded_intersect-simul3.pcap
diff_bounded_intersect-simul1.pcap
diff_bounded_intersect-simul2.pcap
diff_bounded_intersect-simul3.pcap
intersect.pcap
symdiff_simul1.pcap
symdiff_simul2.pcap
symdiff_simul3.pcap
union.pcap
```

**Using -x as well will remove these empty files from output:**

```
symdiff_simul2.pcap
diff_bounded_intersect-simul1.pcap
diff_bounded_intersect-simul2.pcap
diff_bounded_intersect-simul3.pcap
```

.png: union, difference, intersect, symmetric difference

```
pcapgraph examples/ -disu --output png
```



These images contain many set operations applied at the same time. This is more of a demonstration than anything else, as there isn't much of a use case to use all of them at the same time.

### 1.3.5 Addenda

#### Set Theory

---

**Note:** If you want a set primer, you may want to check out the [set operations](#) Wikipedia article.

---

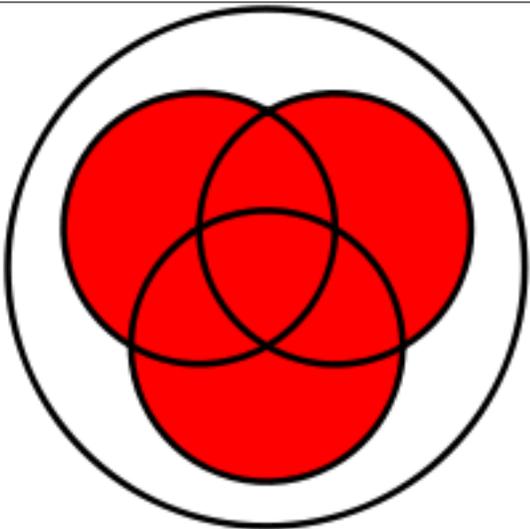
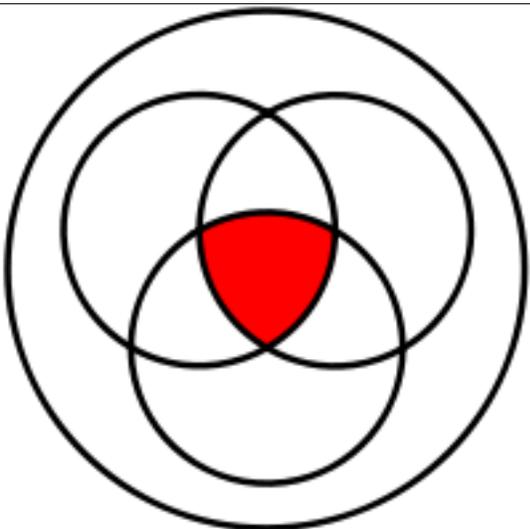
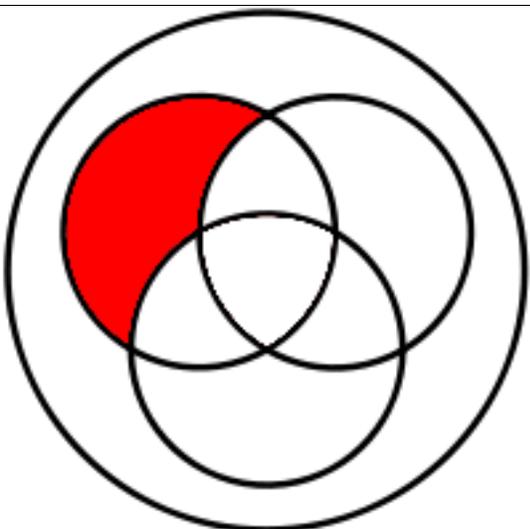
#### Basic Set Operations

The terminology used here is based on set theory. For example, given sets

$$A = (1, 2, 3)$$

$$B = (2, 3, 4)$$

$C = (3, 4, 5)$

Set Operation	Applied to (A, B, C)	Definition
 <p>Union</p>	(1, 2, 3, 4, 5)	All unique elements.
 <p>Intersection</p>	(3)	All common elements.
 <p>Diff</p>	(1)	All elements in the first set not in latter sets.

### Packet Uniqueness

By definition, a set only has unique elements. The result of any set operation is also a set. This program uses the entire frame as an element to determine uniqueness, which ensures fewer duplicates. The FCS may be stripped by the NIC depending on network drivers, and so may not necessarily be available for packet identification (I have only seen Juniper devices take packet captures that contain the FCS).

### Set Caveats

#### Symmetric Difference

Symmetric Difference is included for sake of set operation completeness. It is the equivalent to the set difference applied to all pcaps where each pcap is at some point the pivot. If the difference contains no packets, it is discarded.

Technically, this usage of symmetric difference is incorrect because it produces multiple packet captures with unique packets instead of one containing all of them.

### Generating Demo Packet Captures

---

**Note:** Generating the demo packet captures is optional if you have cloned the repository as these pcaps can be found in `examples/`. Above all else, this is documentation of the pcap generation script.

---

To generate pcaps by letting tshark decide the default interface, enter

```
pcapgraph --generate-pcaps
```

If tshark decides to use a non-active interface, you can specify the interface name manually. To find your active interface, enter `ifconfig` (unix-like), or `ipconfig` (Windows) and find which one has an IP address and non-zero Rx/Tx counts.

```
pcapgraph --generate-pcaps --int <interface-name>
```

**Warning:** On unix-like systems, Wireshark will prompt you during installation to allow/disallow unprivileged users to take packet captures. If you have disallowed unprivileged users, you may need to use `sudo` to capture generated traffic.

### Generation Explanation

`pcapgraph/generate_example_pcaps.py` is the relevant file.

The script creates 3 packet captures, each lasting 60 seconds and starting at 0s, 20s, 40s. After 100s, the script will stop. Packet capture 0s should have 66% in common with pcap 20s and 33% in common with pcap 40s. Indeed, this is what we see in the graph.

### 1.3.6 API Documentation

**Warning:** The CLI is the official interface of this project. The API is documented here for sake of completeness and is not explicitly designed to be one.

- `genindex`
- `modindex`

#### pcapgraph module

version file.

`pcapgraph.get_tshark_status()`  
Errors and quits if tshark is not installed.

On Windows, tshark may not be recognized by `cmd` even if Wireshark is installed. On Windows, this function will add the Wireshark folder to path so `tshark` can be called.

Changing `os.environ` will only affect the `cmd` shell this program is using (tested). Not using `setx` here as that could be potentially destructive.

**Raises `FileNotFoundError`:** If `wireshark/tshark` is not found, raise an error as they are required.

#### pcapgraph.draw\_graph

Draw graph will draw a text or image graph.

`pcapgraph.draw_graph.draw_graph(pcap_packets, input_files, output_fmts, exclude_empty, anonymize_names)`

Draw a graph using matplotlib and numpy.

##### Parameters

- **`pcap_packets`** (*dict*) – All packets, where key is pcap filename/operation.
- **`input_files`** (*list*) – List of input files that shouldn't be deleted.
- **`output_fmts`** (*list*) – The save file type. Supported formats are dependent on the capabilities of the system: [png, pdf, ps, eps, and svg]. See [https://matplotlib.org/api/pyplot\\_api.html#matplotlib.pyplot.savefig](https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.savefig) for more information.
- **`exclude_empty`** (*bool*) – Whether to exclude empty pcaps from graph.
- **`anonymize_names`** (*bool*) – Whether to change filenames to random values.

`pcapgraph.draw_graph.export_graph(pcap_names, save_fmt)`

Exports the graph to the screen or to a file.

##### Parameters

- **`pcap_names`** (*list*) – List of pcap\_names
- **`save_fmt`** (*str*) – File extension of output file

`pcapgraph.draw_graph.generate_graph` (*pcap\_vars*, *empty\_files*, *anonymize\_names*)  
Generate the matplotlib graph.

**Parameters**

- **pcap\_vars** (*dict*) – Contains all data required for the graph. {<pcap>: {'pcap\_start': <timestamp>, 'pcap\_end': <timestamp>}, ... }
- **empty\_files** (*list*) – List of filenames of empty files.
- **anonymize\_names** (*bool*) – Whether to use pseudorandom names for files.

`pcapgraph.draw_graph.get_graph_vars_from_file` (*filename*)  
Setup graph variables.

This function exists to decrease the complexity of generate graph. The order of return vars start\_times\_array, end\_times\_array, and pcap\_names should all match. In other words, the start\_times\_array[5] is for the same pcap as end\_times\_array[5] and pcap\_names[5].

**Parameters** **filename** (*str*) – Name of file

**Returns** File start/stop times if file has 1+ valid packets.

**Return type** (dict)

`pcapgraph.draw_graph.get_x_minmax` (*start\_times*, *end\_times*)  
Determine the horizontal (x) min and max values.

This function adds 1% to either side for padding.

**Parameters**

- **start\_times** (*np.array*) – First packet unix timestamps of all pcaps.
- **end\_times** (*np.array*) – Last packet unix timestamps of all pcaps.

**Returns** min\_x, max\_x to be used for graph

**Return type** (tuple)

`pcapgraph.draw_graph.make_text_not_war` (*pcap\_times*)  
Make useful text given pcap times.

**Parameters** **pcap\_times** (*dict*) – Packet capture names and start/stop timestamps.

**Returns** Full textstring of text to written to file/stdout

**Return type** (str)

`pcapgraph.draw_graph.output_file` (*save\_format*, *pcap\_packets*, *exclude\_empty*,  
*anonymize\_names*)  
Save the specified file with the specified format.

`pcapgraph.draw_graph.remove_or_open_files` (*new\_files*, *open\_in\_wireshark*, *delete\_pcaps*)  
Remove or open files.

delete\_pcaps and open\_in\_wireshark should not both be true, because that would mean that wireshark would try to open deleted files.

**Parameters**

- **new\_files** (*set*) – Set of new filenames to do something with
- **open\_in\_wireshark** (*bool*) – Whether to open files in wireshark
- **delete\_pcaps** (*bool*) – Whether to delete generated pcaps

`pcapgraph.draw_graph.set_horiz_bar_colors` (*barlist*)

Set the horizontal bar colors.

Color theme is Metro UI, with an emphasis on darker colors. If there are more horiz bars than in the color array, loop and continue to set colors.

**Parameters** `barlist` (*list*) – List of the horizontal bars.

`pcapgraph.draw_graph.set_xticks` (*first, last*)

Generate the x ticks and return a list of them.

**Parameters**

- **first** (*float*) – Earliest timestamp of pcaps.
- **last** (*float*) – Latest timestamp of pcaps.

**Returns** `x_ticks` (*list(float)*): List of unix epoch time values as `xticks`. `x_label` (*string*): Text to be used to label X-axis.

**Return type** (*tuple*)

---

## pcapgraph.generate\_example\_pcaps

Script to create three packet captures to demonstrate PcapGraph.

`pcapgraph.generate_example_pcaps.generate_example_pcaps` (*interface=None*)

This script will create 3 packet captures, each lasting 60 seconds and starting at 0s, 20s, 40s. After 100s, this script will stop. Packet capture 0s should have 66% in common with pcap 20s and 33% in common with pcap 40s. Indeed, this is what we see in the graph.

**Parameters** `interface` (*string*) – Optional interface to specify for wireshark.

---

## pcapgraph.get\_filenames

Parse CLI options and return a list of filenames.

`pcapgraph.get_filenames.get_filenames` (*files*)

Return a validated list of filenames.

**Parameters** `files` (*list*) – List of file params entered by user

**Returns** List of files validated to be packet captures.

**Return type** (*list*)

`pcapgraph.get_filenames.get_filenames_from_directories` (*directories*)

Get all the files from all provided directories.

This function is not recursive and searches one deep.

**Parameters** `directories` (*list*) – List of user-inputted directories.

**Returns** Filenames of all packet captures in specified directories.

**Return type** (*list*)

`pcapgraph.get_filenames.parse_cli_args` (*args*)

Parse args with docopt. Return a list of filenames

---

**Parameters** `args` (*dict*) – Dict of args that have been passed in via docopt.

**Returns** List of filepaths

**Return type** (list)

---

### pcapgraph.manipulate\_frames

Parse the frames from files based upon options.

Create the same JSON style with `tshark -r examples/simull1.pcap -T json -x` Note that the `<var>_raw` is due to the `-x` flag.

Frame JSON looks like this:

```
{
  '_index': 'packets-2018-11-03',
  '_type': 'pcap_file',
  '_score': None,
  '_source': {
    'layers': {
      'frame_raw': ['881544abffdd2477035113440800450000380b5d0000...
      'frame': {'frame.encap_type': '1', 'frame.time': 'Sep 26, 2...
      'eth_raw': ['881544abffdd2477035113440800', 0, 14, 0, 1],
      'eth': {'eth.dst_raw': ['881544abffdd', 0, 6, 0, 29], 'eth...
      'ip_raw': ['450000380b5d00004011c7980a3012900a808080', 14, 2...
      'ip': {'ip.version_raw': ['4', 14, 1, 240, 4], 'ip.version'...
      'udp_raw': ['ea6200350024a492', 34, 8, 0, 1],
      'udp': ['udp.srcport_raw': ['ea62', 34, 2, 0, 5], 'udp.srcp...
      'dns_raw': ['9b1301000001000000000000006616d617a6f6e03636f6d...
      'dns': {'dns.id_raw': ['9b13', 42, 2, 0, 5], 'dns.id': '0x00...
    }
  }
}
```

Many of these functions interact with this frame dict format or directly with the frame string (seen in `'frame_raw'`). The frame string is a string of the hex of a packet.

`pcapgraph.manipulate_frames.anonymous_pcap_names` (*num\_names*)

Anonymize pcap names.

Creation of funny pcap names like `switch_wireless` is intended behavior.

**Parameters** `num_names` (*int*) – Number of names to be returned

**Returns** Fake pcap name list

**Return type** (list)

`pcapgraph.manipulate_frames.decode_stdout` (*stdout*)

Given stdout, return the string.

`pcapgraph.manipulate_frames.get_flat_frame_dict` (*pcap\_json\_list*)

Given the pcap json list, return the frame dict.

**Parameters** `pcap_json_list` (*list*) – List of pcap dicts (see `parse_pcaps` for details)

**Returns** {<frame>: <timestamp>, ... }

**Return type** frame\_list (dict)

`pcapgraph.manipulate_frames.get_frame_from_json` (*frame*)

Get/sanitize raw frame from JSON of frame from *tshark -x -T json ...*

**Parameters** `frame` (*dict*) – A dict of a single packet from tshark.

**Returns** The ASCII hexdump value of a packet

**Return type** (*str*)

`pcapgraph.manipulate_frames.get_frame_list_by_pcap` (*pcap\_json\_dict*)

Like `get_flat_frame_dict`, but with pcapname as key to each frame list

**Parameters** `pcap_json_dict` (*dict*) – List of Pcap JSONs.

**Returns** [[<frame>, ...], ...]

**Return type** (*list*)

`pcapgraph.manipulate_frames.get_homogenized_packet` (*ip\_raw*)

Change an IPw4 packet's fields to the same, homogenized values.

Replace TTL, header checksum, and IP src/dst with generic values. This function is designed to replace all IP data that would change on a layer 3 boundary

Note that these options are found only in IPv4. TTL is expected to change at every hop along with header checksum. IPs are expected to change for NAT.

**Parameters** `ip_raw` (*str*) – ASCII hex of packet.

**Returns** Packet with fields that would be altered by l3 boundary replaced

**Return type** (*str*)

`pcapgraph.manipulate_frames.get_packet_count` (*filename*)

Given a file, get the packet count.

**Parameters** `filename` (*str*) – Path of a file, including extension

**Returns** How many packets were in that pcap

**Return type** `packet_count` (*int*)

`pcapgraph.manipulate_frames.get_pcap_as_json` (*pcap*)

Given a pcap, return a json with *tshark -r <file> -x -T json*.

**tshark -r <pcap> -w -** Pipes packet capture one packet per line to stdout

**tshark -r -** Read file from stdin

**tshark -r <in.pcap> -x | text2pcap - <out.pcap>** Prints hex of pcap to stdout and then resaves it as a pcap. This WILL delete packet timestamps as that is not encoded in hex output.

**Parameters** `pcap` (*string*) – File name.

**Returns** List of the pcap json provided by tshark.

**Return type** (*list*)

`pcapgraph.manipulate_frames.get_pcap_frame_dict` (*pcaps*)

Like `get_flat_frame_dict`, but with pcapname as key to each frame list

**Parameters** `pcaps` (*list*) – List of pcap file names.

**Returns** {<pcap>: {<frame>:<timestamp>, ...}, ... }

**Return type** (*dict*)

`pcapgraph.manipulate_frames.parse_pcaps` (*pcaps*)

Given pcaps, return all frames and their timestamps.

**Parameters** `pcaps` (*list*) – A list of pcap filenames

**Returns**

All the packet data in json format. [{"pcap": {PCAP JSON}}, ...]

**Return type** `pcap_json_list` (*list*)

`pcapgraph.manipulate_frames.strip_layers` (*filenames, options*)

Get the PCAP JSON dict stripped per options.

**strip-l3:** Replace layer 3 fields src/dst IP, ttl, checksum with dummy values

**strip-l2:** Remove all layer 2 fields like FCS, source/dest MAC, VLAN tag...

**Parameters**

- **filenames** (*list*) – List of filenames.
- **options** (*dict*) – Whether to strip L2 and L3 headers.

**Returns** The modified packet dict

**Return type** (*dict*)

---

## pcapgraph.pcap\_math

Do algebraic operations on sets like union, intersect, difference.

**class** `pcapgraph.pcap_math.PcapMath` (*filenames, options*)

Bases: `object`

Prepare PcapMath object for one or multiple operations.

Every PcapMath object should start with the data structures filled with the data that each operation needs to function.

**Parameters**

- **filenames** (*list*) – List of filenames.
- **options** (*dict*) – Whether to strip L2 and L3 headers.

**bounded\_intersect\_pcap** ()

Create a packet capture intersection out of two files using ip.ids.

Create a packet capture by finding the earliest common packet by and then the latest common packet in both pcaps by ip.id.

**Returns** Filenames of generated pcaps.

**Return type** (*list(string)*)

**difference\_pcap** (*pivot\_index=0*)

Given sets A = (1, 2, 3), B = (2, 3, 4), C = (3, 4, 5), A-B-C = (1).

**Parameters** [**int**] (*pivot\_index*) – Specify minuend by index of filename in list

**Returns** Name of generated pcap.

---

**Return type** (string)

**get\_bounded\_pcaps** ()

Get the pcap frame list for bounded\_intersect\_pcap

Create a bounding box around each packet capture where the bounds are the min and max packets in the intersection.

**Returns** A list of frame\_dicts

**Return type** bounded\_pcaps (list)

**get\_minmax\_common\_frames** ()

Get first, last frames of intersection pcap.

**Returns** Packet strings of the packets that are at the beginning and end of the intersection pcap based on timestamps.

**Return type** min\_frame, max\_frame (tuple(string))

**Raises** `assert` – If intersection is empty.

**intersect\_pcap** ()

Save pcap intersection. First filename is pivot packet capture.

**Returns** Filename of generated pcap.

**Return type** (str)

**inverse\_bounded\_intersect\_pcap** (*bounded\_filelist=False, intersect\_file=False*)

Inverse of bounded intersection = (bounded intersect) - (intersect)

**Parameters**

- **bounded\_filelist** (*list*) – List of existing bounded pcaps generated by bounded\_intersect\_pcap()
- **intersect\_file** (*string*) – Location of intersect file.

**Returns** Filenames of generated pcaps.

**Return type** (list(string))

**parse\_set\_args** (*args*)

Call the appropriate method per CLI flags.

difference, union, intersect consist of {<op>: {frame: timestamp, ...}} bounded\_intersect consists of {pcap: {frame: timestamp, ...}, ...}

**Parameters** *args* (*dict*) – Dict of all arguments (including set args).

**Returns**

**List of all files, including ones generated** by set operations.

**Return type** filenames (list)

**static print\_10\_most\_common\_frames** (*raw\_frame\_list*)

After doing a packet union, find/print the 10 most common packets.

This is a work in progress and may eventually use this bash:

```
<packets> | text2pcap - - | tshark -r - -o 'gui.column.format:"No.", "%m", "VLAN", "%q", "Src
MAC", "%uhs", "Dst MAC", "%uhd", "Src IP", "%us", "Dst IP", "%ud", "Protocol", "%p", "Src
port", "%uS", "Dst port", "%uD"
```

Alternatively, just use the existing information in pcap\_dict.

The goal is to print frame#, VLAN, src/dst MAC, src/dst IP, L4 src/dst ports, protocol

This should likely be its own CLI flag in future.

**Parameters** `raw_frame_list` (*list*) – List of raw frames

**symmetric\_difference\_pcap** ()

For sets A = (1, 2, 3), B = (2, 3, 4), C = (3, 4, 5), ABC = (1, 5)

For all pcaps, the symmetric difference produces a pcap that has the packets that are unique to only that pcap (unlike above where only one set is the result).

**Returns** Filenames of generated pcaps.

**Return type** (list(str))

**union\_pcap** ()

Given sets A = (1, 2, 3), B = (2, 3, 4), A + B = (1, 2, 3, 4).

**About:** This method uses tshark to get identifying information on pcaps and then mergepcap to save the combined pcap.

**Returns** Name of generated pcap.

**Return type** (string)

---

## pcapgraph.save\_file

Save file.

`pcapgraph.save_file.convert_to_pcaptext` (*raw\_packet*, *timestamp*="")

Convert the raw pcap hex to a form that text2cap can read from stdin.

hexdump and xxd can do this on unix-like platforms, but not on Windows.

`tshark -r <file> -T json -x` produces the “in” and text2pcap requires the “out” formats as shown below:

Per Text2pcap documentation: “Text2pcap understands a hexdump of the form generated by `od -Ax -tx1 -v`.”

In format:

```
247703511344881544abbfd0800452000542bbc00007901e8fd080808080a301290000
082a563110001f930ab5b00000000a9e80d000000000101112131415161718191a1b1c
1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637
```

Out format:

```
0000 24 77 03 51 13 44 88 15 44 ab bf dd 08 00 45 20
0010 00 54 2b bc 00 00 79 01 e8 fd 08 08 08 08 0a 30
0020 12 90 00 00 82 a5 63 11 00 01 f9 30 ab 5b 00 00
0030 00 00 a9 e8 0d 00 00 00 00 00 10 11 12 13 14 15
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060 36 37
```

NOTE: Output format doesn’t need an extra n between packets. So in the above example, the next line could be 0000 00 ... for the next packet.

**Parameters**

- **raw\_packet** (*str*) – The ASCII hexdump seen above in ‘In’
- **timestamp** (*str*) – An optional packet timestamp that will precede the 0000 line of the packet hex.

**Returns** Packet in ASCII hexdump format like *Out* above

**Return type** `formatted_string` (*str*)

`pcapgraph.save_file.reorder_packets` (*pcap*)

Union causes packets to be ordered incorrectly, so reorder properly.

Reorder packets, save to 2nd file. After this is done, replace initial file with reordered one. Append temporary file with ‘\_’.

**Parameters** **pcap** (*str*) – Filename of packet capture. Should end with ‘\_’, which can be stripped off so that we can reorder to a diff file.

`pcapgraph.save_file.save_pcap` (*pcap\_dict*, *name*, *options*)

Save a packet capture given ASCII hexdump using *text2pcap*

**Parameters**

- **pcap\_dict** (*dict*) – List of pcaps of frames to timestamps. Format: {<frame>: <timestamp>, ... }
- **name** (*str*) – Type of operation and name of savefile
- **options** (*dict*) – Whether to encode with L2/L3 headers.



**p**

pcapgraph, 25  
pcapgraph.draw\_graph, 25  
pcapgraph.generate\_example\_pcaps, 27  
pcapgraph.get\_filenames, 27  
pcapgraph.manipulate\_frames, 28  
pcapgraph.pcap\_math, 30  
pcapgraph.pcapgraph, 6  
pcapgraph.save\_file, 32



**A**

anonymous\_pcap\_names() (in module pcap-graph.manipulate\_frames), 28

**B**

bounded\_intersect\_pcap() (pcap-graph.pcap\_math.PcapMath method), 30

**C**

convert\_to\_pcaptext() (in module pcapgraph.save\_file), 32

**D**

decode\_stdout() (in module pcap-graph.manipulate\_frames), 28

difference\_pcap() (pcapgraph.pcap\_math.PcapMath method), 30

draw\_graph() (in module pcapgraph.draw\_graph), 25

**E**

export\_graph() (in module pcapgraph.draw\_graph), 25

**G**

generate\_example\_pcaps() (in module pcap-graph.generate\_example\_pcaps), 27

generate\_graph() (in module pcapgraph.draw\_graph), 25

get\_bounded\_pcaps() (pcapgraph.pcap\_math.PcapMath method), 31

get\_filenames() (in module pcapgraph.get\_filenames), 27

get\_filenames\_from\_directories() (in module pcap-graph.get\_filenames), 27

get\_flat\_frame\_dict() (in module pcap-graph.manipulate\_frames), 28

get\_frame\_from\_json() (in module pcap-graph.manipulate\_frames), 28

get\_frame\_list\_by\_pcap() (in module pcap-graph.manipulate\_frames), 29

get\_graph\_vars\_from\_file() (in module pcap-graph.draw\_graph), 26

get\_homogenized\_packet() (in module pcap-graph.manipulate\_frames), 29

get\_minmax\_common\_frames() (pcap-graph.pcap\_math.PcapMath method), 31

get\_packet\_count() (in module pcap-graph.manipulate\_frames), 29

get\_pcap\_as\_json() (in module pcap-graph.manipulate\_frames), 29

get\_pcap\_frame\_dict() (in module pcap-graph.manipulate\_frames), 29

get\_tshark\_status() (in module pcapgraph), 25

get\_x\_minmax() (in module pcapgraph.draw\_graph), 26

**I**

intersect\_pcap() (pcapgraph.pcap\_math.PcapMath method), 31

inverse\_bounded\_intersect\_pcap() (pcap-graph.pcap\_math.PcapMath method), 31

**M**

make\_text\_not\_war() (in module pcap-graph.draw\_graph), 26

**O**

output\_file() (in module pcapgraph.draw\_graph), 26

**P**

parse\_cli\_args() (in module pcapgraph.get\_filenames), 27

parse\_pcaps() (in module pcapgraph.manipulate\_frames), 29

parse\_set\_args() (pcapgraph.pcap\_math.PcapMath method), 31

pcapgraph (module), 25

pcapgraph.draw\_graph (module), 25

pcapgraph.generate\_example\_pcaps (module), 27

pcapgraph.get\_filenames (module), 27

pcapgraph.manipulate\_frames (module), 28

pcapgraph.pcap\_math (module), 30

pcapgraph.pcapgraph (module), 6

pcapgraph.save\_file (module), 32  
PcapMath (class in pcapgraph.pcap\_math), 30  
print\_10\_most\_common\_frames() (pcap-  
graph.pcap\_math.PcapMath static method),  
31

## R

remove\_or\_open\_files() (in module pcap-  
graph.draw\_graph), 26  
reorder\_packets() (in module pcapgraph.save\_file), 33

## S

save\_pcap() (in module pcapgraph.save\_file), 33  
set\_horiz\_bar\_colors() (in module pcap-  
graph.draw\_graph), 26  
set\_xticks() (in module pcapgraph.draw\_graph), 27  
strip\_layers() (in module pcapgraph.manipulate\_frames),  
30  
symmetric\_difference\_pcap() (pcap-  
graph.pcap\_math.PcapMath method), 32

## U

union\_pcap() (pcapgraph.pcap\_math.PcapMath method),  
32