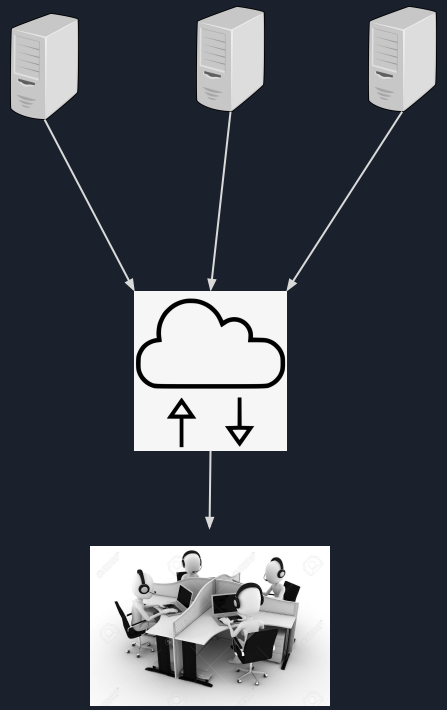
PacketStream

(a platform for analyzing network traffic in near real-time)

Leonid Isaev

Insight Data Eng. Fellowship
NY 2019

Overview



Goals:

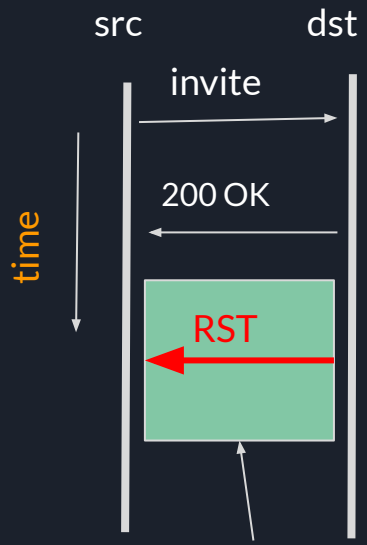
- Develop a distributed platform for troubleshooting network-related failures
  - I have configured my server correctly, but still clients can't talk to it;
  - My print/fax server doesn't produce a complete document;
  - ...
- Wireshark-like rules (network engineers have self-grown wireshark module).

Constraints:

- Must be field-ready
  - A data producer is a CentOS/Ubuntu box with standard repos;
  - No "pip install <200 packages>";

CentOS ubuntu redhat

Examples of broken networks

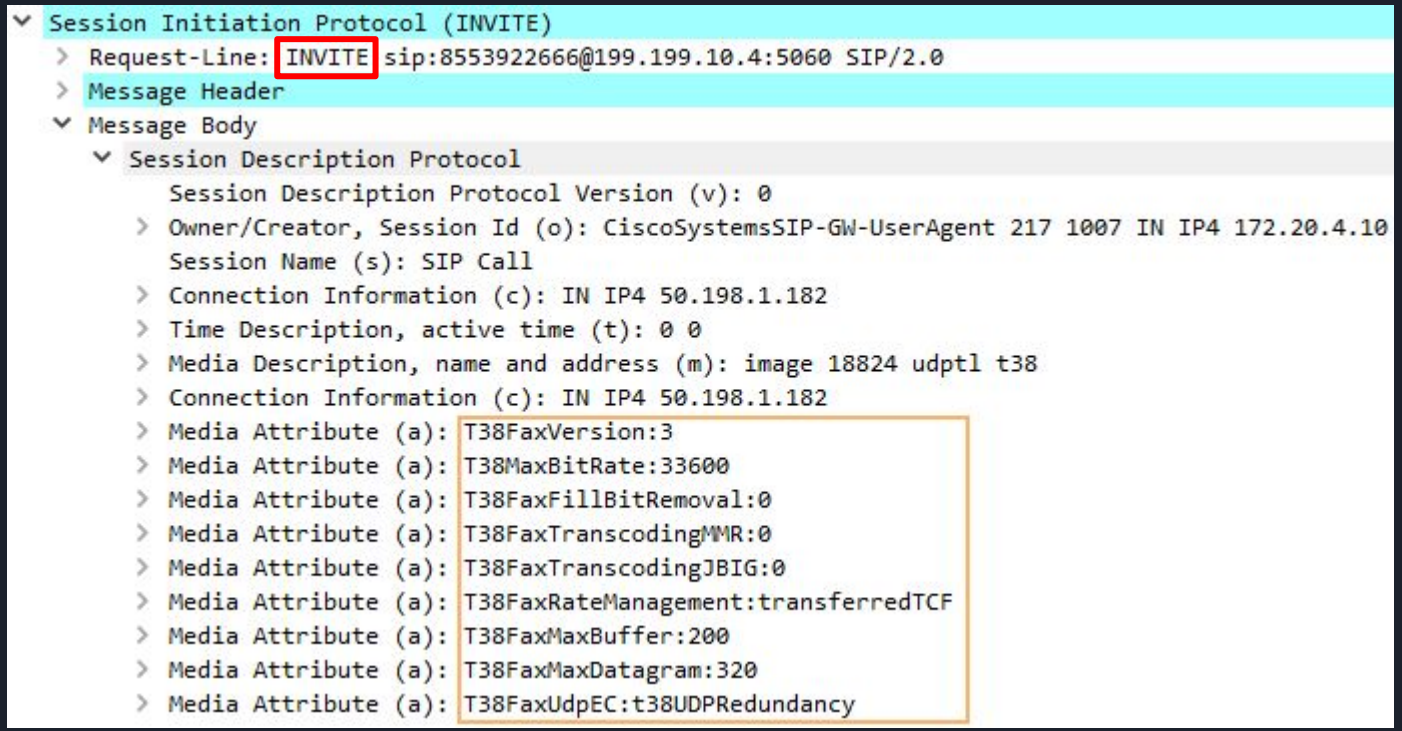


standard-mandated delay
(not OK to reset before it elapses)

- Broken firmware -> timing issues, random connection resets;
- Wrong credentials, encoding, etc.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	ipphone	Broadcast	ARP	Who has 192.168.1.10? Tell 192.168.1.151
2	0.052480	pbxhost	ipphone	ARP	192.168.1.10 is at 00:10:5a:e1:90:6e
3	0.000146	ipphone	pbxhost	SIP/SDP	Request: INVITE sip:sip%3a613@fwd.pulver.com, with session description
4	0.001480	pbxhost	ipphone	SIP	Status: 407 Proxy Authentication Required
5	0.052920	ipphone	pbxhost	SIP	Request: ACK sip:sip%3a613@fwd.pulver.com
6	0.062692	ipphone	pbxhost	SIP/SDP	Request: INVITE sip:sip%3a613@fwd.pulver.com, with session description
7	0.063859	pbxhost	ipphone	SIP	Status: 407 Proxy Authentication Required
8	0.123275	ipphone	pbxhost	SIP	Request: ACK sip:sip%3a613@fwd.pulver.com
9	0.132364	ipphone	pbxhost	SIP/SDP	Request: INVITE sip:sip%3a613@fwd.pulver.com, with session description
10	0.135772	pbxhost	ipphone	SIP	Status: 407 Proxy Authentication Required
11	0.193816	ipphone	pbxhost	SIP	Request: ACK sip:sip%3a613@fwd.pulver.com
12	0.202913	ipphone	pbxhost	SIP/SDP	Request: INVITE sip:sip%3a613@fwd.pulver.com, with session description
13	0.208640	pbxhost	ipphone	SIP	Status: 407 Proxy Authentication Required
14	0.258099	ipphone	pbxhost	SIP	Request: ACK sip:sip%3a613@fwd.pulver.com

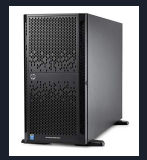
Sample packet structure (T.38 fax protocol)



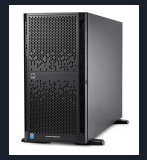
Packet structure, cont'd - broken packets

```
Frame 611: 214 bytes on wire (1712 bits), 214 bytes captured (1712 bits)
Ethernet II, Src: Fortinet_09:00:04 (00:09:0f:09:00:04), Dst: Vmware_9e:71:15 (00:50:56:9e:71:15)
Internet Protocol Version 4, Src: 4.28.93.140, Dst: 172.27.11.68
User Datagram Protocol, Src Port: 20458, Dst Port: 56008
ITU-T Recommendation T.38
    [Stream setup by SDP (frame 598)]
        [Stream frame: 598]
        [Stream Method: SDP]
    UDPTLPacket
        seq-number: 32768
        something unknown here [10.9 Unconstrained unexpected fragment count]
            [Expert Info (Warning/Undecoded): something unknown here [10.9 Unconstrained unexpected fragment count]]
                [something unknown here [10.9 Unconstrained unexpected fragment count]]
                [Severity level: Warning]
                [Group: Undecoded]
[Malformed Packet: T.38]
```

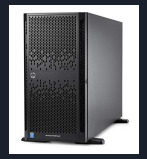
Pipeline



tcpdump | nc -l
syslog-ng

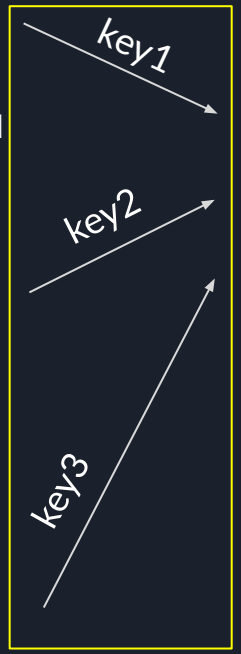


tcpdump | nc -l
syslog-ng



tcpdump | nc -l
syslog-ng

Kafka topic



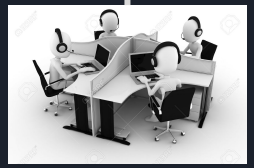
ingestion

compute

storage

packet metadata

frontend

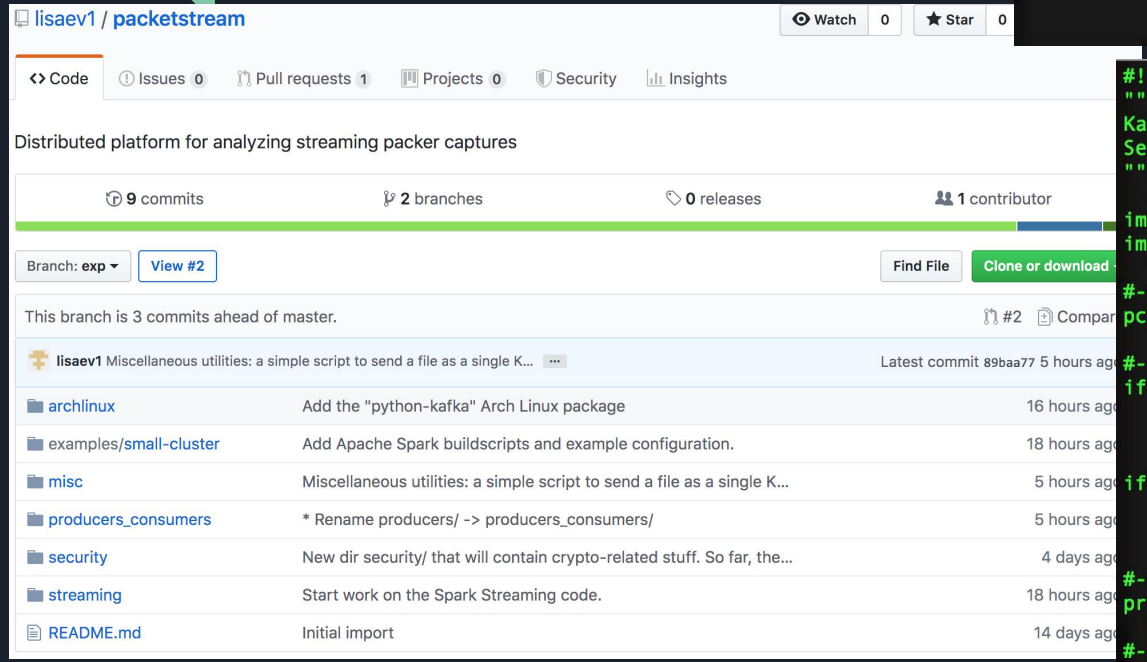


https://github.com/lisaev1/packetstream

Repository

https://github.com/lisaev1/packetstream

```
usage: scapy_prod.py [-h] [-p FILE] [-t TOPIC] [-s SERVER:PORT]

optional arguments:
  -h, --help            show this help message and exit
  -p FILE, --pcap FILE  Packet capture (pcap0 file to parse (default: none)
  -t TOPIC, --topic TOPIC
                        Kafka topic (default: sharks)
  -s SERVER:PORT, --servers SERVER:PORT
                        Kafka broker (default: none). Can be specified
                        multiple times.
```



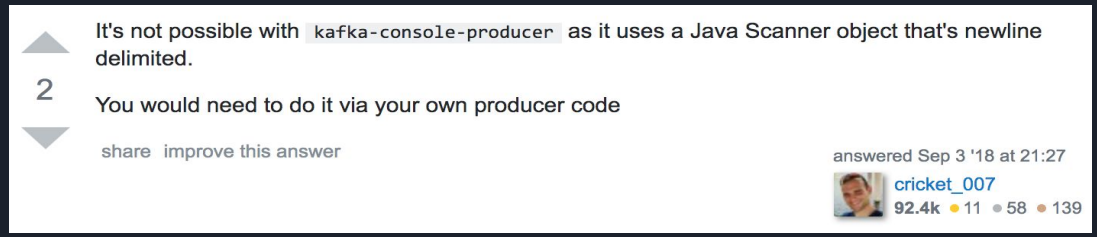

```
#!python
"""
Kafka producer for packets stored in a pcap file.
Sends 1 scapy packet per message.
"""

import os, sys, kafka, common
import scapy.all as scapy

#-- parse cmdline options
pcap_f, topic, srv_list = common._parseopts()

#-- sanity checks
if (not os.path.isfile(pcap_f)):
    print(f"File \"{pcap_f}\" does not exist! Aborting.")
    sys.exit(1)

if (len(srv_list) == 0):
    print("Please specify at least one Kafka broker! Aborting.")
    sys.exit(1)

#-- define a producer
producer = kafka.KafkaProducer(bootstrap_servers = srv_list)

#-- read the pcap file
capture = scapy.rdpcap(pcap_f)
for p in capture:
    producer.send(topic, bytes(p))

producer.close()
```

Run

```
C02RJ1JPG8WM:packetstream insight$ python3 producers_consumers/python/scapy_prod.py -p ~/Downloads/_
ifax.pcap -t "test1" $(for i in {1..3}; do echo -nE "-s 192.168.122.7${i}:9092 "; done)
```

```
lisaev@spark-node2 ~$ spark-submit --master spark://spark-node1:7077 --jars ks.jar ./echo.py 192.1
68.122.71:9092 test1

<bound method Packet.summary of <Ether  dst=00:09:0f:09:00:04 src=00:50:56:9e:71:15 type=IPv4 |<IP
version=4 ihl=5 tos=0x0 len=931 id=28856 flags=DF frag=0 ttl=64 proto=udp chksum=0xad8a src=172.27.1
1.68 dst=4.28.93.140 |<UDP  sport=sip dport=sip len=911 chksum=0x1ca8 |<Raw  load='INVITE sip:0ae9fa
eaad38d22c779775e03bb9d755@4.28.93.140:5060 SIP/2.0\r\nFrom: <sip:4012830857@sbc01-stl.sip.univergeb
lue.com>;tag=dcfef6b0-0-13c4-65014-a5c-75d5ed11-a5c\r\nTo: "PHILADELPHIA PA"<sip:2153830199@sbc01-st
l.sip.univergeblue.com>;tag=4.28.93.140+1+3218a8f7+270ff97\r\nCall-ID: 0gQAAC8WAAACBAAALxYAANYw/3hJv
01vNq45pvOsNQbJaGaS1oYg5/3tEuM5AjEX@4.28.93.140\r\nCSeq: 1 INVITE\r\nVia: SIP/2.0/UDP 172.27.11.68:5
060;branch=z9hG4bK-a60-28871c-296f3a47-dd2ff5e8\r\nSupported: 100rel\r\nMax-Forwards: 70\r\nUser-Age
nt: Brktsip/6.9.6B10 (Dialogic)\r\nContact: <sip:172.27.11.68>\r\nContent-Type: application/sdp\r\nC
ontent-Length: 286\r\n\r\nv=0\r\no=- 2208997233 0083442001 IN IP4 172.27.11.68\r\ns=no_session_name\
r\nt=0 0\r\nm=image 56008 udptl t38\r\nc=IN IP4 172.27.11.68\r\na=T38FaxVersion:0\r\na=T38MaxBitRate
:14400\r\na=T38FaxRateManagement:transferredTCF\r\na=T38FaxMaxBuffer:200\r\na=T38FaxMaxDatagram:72\r
\na=T38FaxUdpEC:t38UDPRedundancy\r\n' |>>>>>

<bound method Packet.summary of <Ether  dst=00:50:56:9e:71:15 src=00:09:0f:09:00:04 type=IPv4 |<IP
version=4 ihl=5 tos=0x0 len=491 id=49541 flags= frag=0 ttl=64 proto=udp chksum=0x9e75 src=4.28.93.14
0 dst=172.27.11.68 |<UDP  sport=sip dport=sip len=471 chksum=0x9451 |<Raw  load='SIP/2.0 100 Trying\
r\nCall-ID: 0gQAAC8WAAACBAAALxYAANYw/3hJv01vNq45pvOsNQbJaGaS1oYg5/3tEuM5AjEX@4.28.93.140\r\nCSeq: 1
INVITE\r\nFrom: <sip:4012830857@sbc01-stl.sip.univergeblue.com>;tag=dcfef6b0-0-13c4-65014-a5c-75d5ed
11-a5c\r\nTo: "PHILADELPHIA PA"<sip:2153830199@sbc01-stl.sip.univergeblue.com>;tag=4.28.93.140+1+321
8a8f7+270ff97\r\nVia: SIP/2.0/UDP 172.27.11.68:5060;branch=z9hG4bK-a60-28871c-296f3a47-dd2ff5e8\r\nS
erver: SIP/2.0\r\nContent-Length: 0\r\nUser-Agent: SoTel\r\n\r\n' |>>>>>

...
```

Challenges: Messages with $'\n' in Kafka (a.k.a Java sucks)



2

It's not possible with `kafka-console-producer` as it uses a Java Scanner object that's newline delimited.

You would need to do it via your own producer code

share improve this answer

answered Sep 3 '18 at 21:27

cricket_007

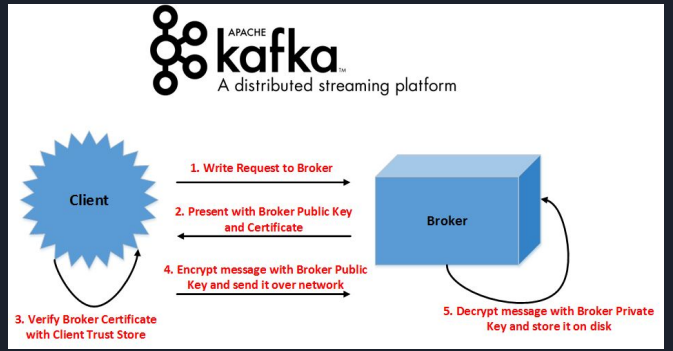
92.4k • 11 • 58 • 139

```
C02RJ1JPG8WM:packetstream insight$ cat misc/ser.py
#!python
"""
Kafka Consumer/Producer Java and Python API has problems handling messages with
$'\n' characters [1]. This program consumes a multiline message from stdin and
writes its hex-encoding to stdout. The latter can be pushed to Kafka as a singl
e
message.

[1] https://stackoverflow.com/questions/52151816/push-multiple-line-text-as-one
-message-in-a-kafka-topic
"""

import sys, binascii

for l in sys.stdin.buffer:
    sys.stdout.buffer.write(binascii.hexlify(l))

#sys.stdout.buffer.write(b'\n')
```

Challenges: encrypted traffic from producers

- Initial approach: use SSH tunnels
  - Works, but Kafka broker readvertisement doesn't happen;
  - Slow because SSH limits packet length (compared to http/https).

- Solution: use Kafka over TLS
  - Certs are a PAIN;
  - CA vs self-signed



Short bio

Background

- Indiana Univ Bloomington (condensed-matter theory & computational physics)
- Postdoc at Louisiana State U Baton Rouge, Los Alamos, JILA/NIST/CU Boulder
- Linux support and software engineer @ iFax Solutions, Inc., Norristown PA

Fun facts/hobbies:

- Real-life Sheldon
- Science-fiction, hacking on Arch Linux

Roadmap: putting all this into production

Nobody should start to undertake a large project. You start with a small *trivial* project, and you should never expect it to get large. If you do, you'll just overdesign and generally think it is more important than it likely is at that stage. So start small, and think about the details. Don't think about some big picture and fancy design. If it doesn't solve some fairly immediate need, it's almost certainly over-designed.

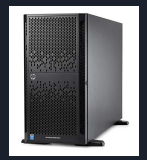
-- Linus Torvalds, 2004

- CLI interface (Dash is bful, but dysfunctional):
  - Need to look at raw packets;
  - Wireshark rules/filters;
- Richer processing rules in Spark, incl wireshark rule files;
- Dynamic provisioning of Kafka brokers and Spark nodes;
- Dedicated distributed storage (perhaps outside Cassandra).

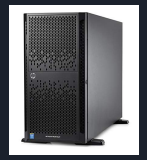
Drawbacks of Kafka streaming

- Streaming apps have to be started manually on each broker with # of instances = # of partitions in the consumed topic;
- No automatic re-spawning in case of node failure;
- Complex code (lots of async stuff);
- Processing and message handling are co-located.

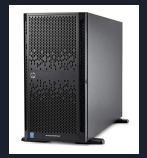
Candidate pipeline 1



tcpdump | nc -l
syslog-ng



tcpdump | nc -l
syslog-ng



tcpdump | nc -l
syslog-ng

single topic

key1

key2

key3

Apache Kafka

Kafka streams

TIMESCALE

Consuming streaming data on Kafka cluster

Faust

A library for building streaming applications in Python.

Star 3,691

build failing

Navigation

Copyright
Introducing Faust
Playbooks
User Guide
Frequently Asked Questions (FAQ)
API Reference
Changes
Contributing
Developer Guide
History

Faust - Python Stream Processing

```
# Python Streams ٩(◕‿◕)۶
# Forever scalable event processing & in-memory durable K/V store;
# w/ asyncio & static typing.
import faust
```

Faust is a stream processing library, porting the ideas from Kafka Streams to Python.

It is used at Robinhood to build high performance distributed systems and real-time data pipelines that process billions of events every day.

Faust provides both *stream processing* and *event processing*, sharing similarity with tools such as Kafka Streams, Apache Spark/Storm/Samza/Flink,

It does not use a DSL, it's just Python! This means you can use all your favorite Python libraries when stream processing: NumPy, PyTorch, Pandas, NLTK, Django, Flask, SQLAlchemy, ++

Faust requires Python 3.6 or later for the new async/await syntax, and variable type annotations.

Here's an example processing a stream of incoming orders:

```
app = faust.App('myapp', broker='kafka://localhost')

# Models describe how messages are serialized:
# {"account_id": "3fae-...", amount": 3}
class Order(faust.Record):
    account_id: str
    amount: int

@app.agent(value_type=Order)
async def order(orders):
    async for order in orders:
```