Abusing Resources to Process 7TB of PCAP Data

… Or how not to fork-bomb yourself

Roy Hooper
Demand Media
OARC Fall 2013 Workshop

Demand
Media

Introduction

- Interisle’s report raised lots of new questions about DNS collisions
- Triggered further analysis by NTAG and others
- ICANN’s 3 week window for comments didn’t leave much time for analysis
- Interisle’s work took a week for each pass, with only 8 cores…
- DNS-OARC’s policies require analysis work be done at DNS-OARC
- This could be challenging…

DITL Data Size

- Examined RAW DITL data for 2012 and 2013
- Processed only root servers

Year	Size	Queries	Root Servers
April 2012	5.2 TB	45 billion *	10
May 2013	1.8 TB	39 billion *	11
Combined	7 TB	84 billion	

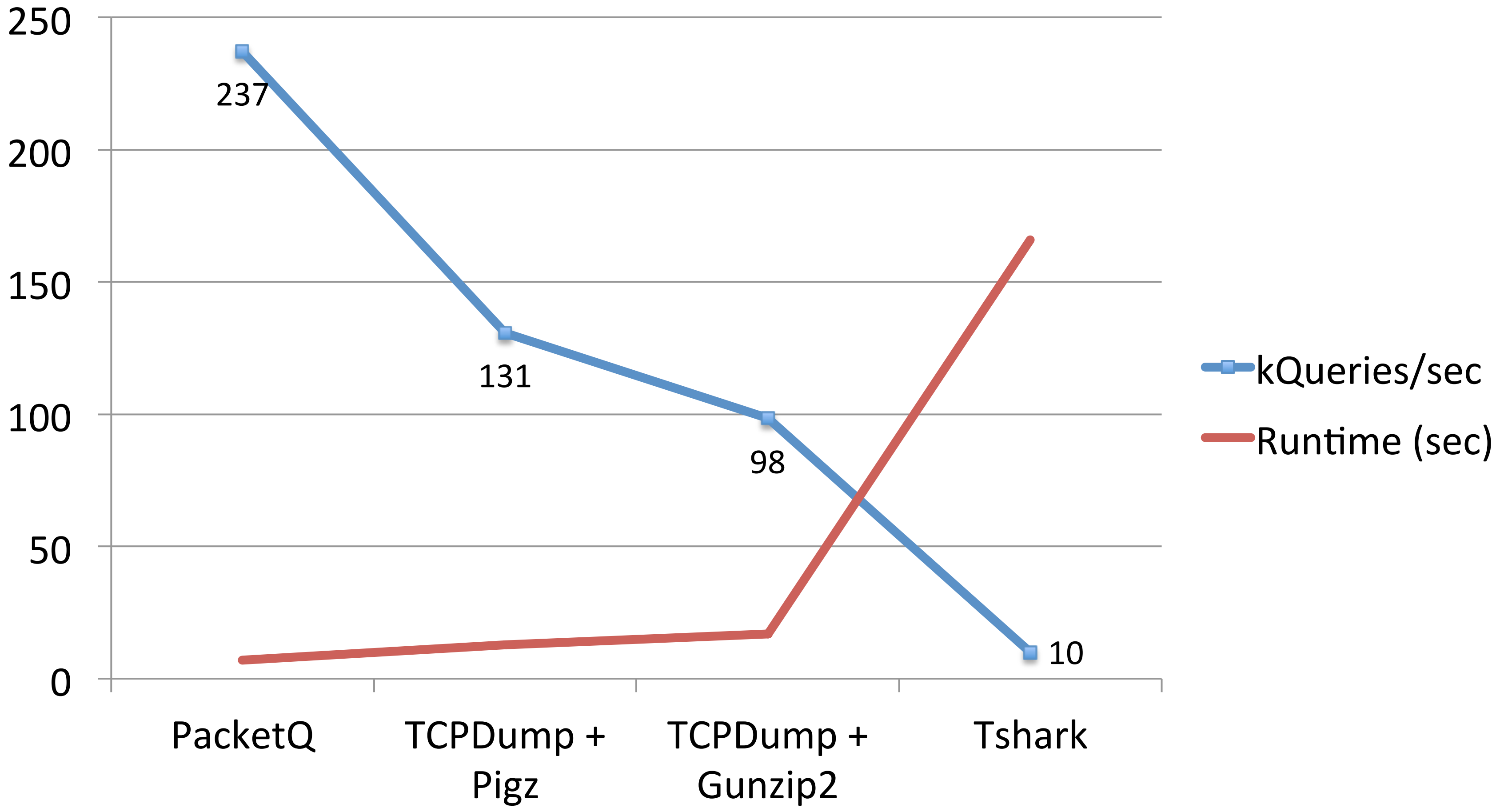
- 650,000 gzip-Compressed PCAP files
- 7 TB is big, but not huge
- Processing environment could make time constraints a serious problem

PCAP Conversion Tools

- PacketQ
- Tcpdump
- Tshark
- And others… like pypcap for Python and Net::Pcap for Perl

- Which one’s right for the job?
- Benchmarked a small sample of PCAP files

PCAP tool performance



Demand Media

PCAP Conversion Tools

Tool	Queries Found	Performance	Internal Gunzip	Difference % (vs Tshark)
PacketQ	1657799	🐇🐇🐇🐇🐇🐇	Yes	-0.040%
TCPDump	1658156	🐇🐇🐇	No	-0.018%
Tshark	1658449	🐢	Yes	

- Wanted something other than PacketQ
- Interesting difference in packet counts… Hope it's not a problem! Pretty close, however.

- Went with tried and true tcpdump

Tool Invocation

Tool	Example Command
PacketQ	packetq --csv -s "SELECT src_addr,qname,qtype from dns WHERE qr=0” pcap.gz | …
Tcpdump	pigz -dc pcap.gz | tcpdump -n -r - |…
TShark	Tshark -n -r pcap.gz -R "dns.flags.response == 0" -T fields -e dns.qry.name -e ip.src -e dns.qry.type | …

Tcpdump will be more complex to work with (string processing)
PacketQ still faster…
… but we didn’t have a recent version yet
… and wanted to get same results with different software

Demand
Media

Conversion and Filtering

- Decompress, parse PCAP and convert to text, then filter out gTLDs
- Take intermediate results and produce per-gTLD files sorted by SLD
- Have 128 threads of execution to take harness
- Validate our extraction using Interisle's report

- How are we going to do this quickly?

Hardware Donation

- Some members got together and donated of a pair of Dell R810s
  - 32 cores (64 threads)
  - 144gb RAM
  - 4TB of local storage
- Up to 128 threads of execution and 288gb of RAM to ~~ab~~use!
- Acceptable compute power … hopefully!

Extraction Goals

- Include only queries for the new gTLDs
- Produce a compact data set suitable for further analysis
- Do it quickly! Less than 3 weeks.
- Match the Interisle results

Opportunity for Parallelism

- Two machines, 64 cores, and lots of CPU work to do with tcpdump
- A number of tools at our disposal:
  - Pigz – Multi-threaded gzip
  - Pipes!
  - pigz | tcpdump | filter
  - xargs -P

Challenges

- Keeping the CPUs busy
- Avoiding swap
- NFS can be slow
- Be wary of local disk bottlenecks
- Dataset doesn't fit on local disk (rsync server not a good option)
- Babysitting multi-day jobs
- Finding problems after 2 days of processing and starting over

Processing Challenges

- Used Perl because of it’s string processing performance
- Perl Regexes too slow for the patterns needed for tcpdump output (Despite being fast)
- Devel::NYProf to the rescue
- Had to use things like rindex() and substr()
- Ugh, manual string processing! Reminds me of C…

Extraction Process

- Produced an intermediate set of files
  - One file per PCAP file
  - Text files: srcaddr qtype qname
  - Converted qname to lowercase
  - Gzipped
- Pipelined
- pigz | tcpdump | parse+filter.pl | pigz
- That’s only 4 processes…

Going Parallel

- Split work between an3 and an4 by splitting file list in half
- xargs -P40 -n 1
- Going higher didn’t yield apparent speed gain
- CPUs pretty busy (load avg in 50s, 80%+ CPU use)
- NFS Bottleneck?
- Local IO Bottleneck?

xargs –P: Easy Parallelism

- xargs parallel mode
- Automatically spawns new processes as work is finished
- Needs a small script to manage the remainder of the pipeline
- Wrote a shell script wrapper:
  - create output dirs
  - run filter
  - pipe to pigz
- Could have had the filter do that... And did in later variants of the scripts

Producing per TLD files

- Re-process compressed intermediate files
- Parse out qname into TLD, SLD
- Split into one file per TLD
- Eventually sort per-TLD files by SLD
- Eventually compress
- Potentially slow
- Parallelize!

Going Parallel

- Relatively CPU hungry
- Use xargs -P32 again, but with -n 100
- Take advantage of O_APPEND flag to open(2)
- Multiple writers safe with use of write(2)
- Each splitter process opens per-TLD files as needed
- Exhausted all File Handles on first try!

Hitting Limits

- Had to increase kern.maxfiles, kern.maxfilesperproc
- Up to 45000 open files with xargs -P32
- IO bound by local disk, not CPU
  - Would be better if we could compress first!
- Reduced parallelism to around -P20

What about that forkbomb?

- At one point, I decided to fork one gzip per TLD per process
- I didn’t do the math first… 1400*32… 44,800
- Noticed what was happening and shut it down quickly
- Need to be careful when forking!

Final sorting and compression

- The unix sort(1) utility allows in-memory sorting via -S
- Avoids temporary files
- Selected a size suitable to the bulk of the intermediate files with 20 processes running
- Sorted by SLD
- Files ready for use!

Shrinking Data

- Input: 7TB, 84 billion queries
- Intermediate: 30GB (excluding .com)

- Final data:
  2012: 1.3% of input (0.6 billion queries)
  2013: 2.4% of input (1.4 billion queries)

- Much easier to process per-gTLD files

Differences in Results

- As expected, tcpdump and PacketQ have slightly different output
- Total query counts for new gTLDs are close:
  - 2012: 2.0% fewer queries than Interisle (862M vs 881M)
  - 2013: 1.4% more queries than Interisle (1334M vs 1316M)
- Size on disk of input data matches Interisle report (page 21)

.com Zonefile intersection

- Repeated initial extraction of intermediate data, but included only .com
- Processed a zone file that overlapped with the 2012 and 2013 DITL runs to produce an SLD string list for each
- Wrote a simple filter daemon
  - Load entire string list into memory in a hash table
  - Pre-fork some children
  - Read lines of input from network connections
  - Respond with entries for SLDs not in zone file

.com Zonefile Intersection

- Used xargs to feed intermediate file contents in parallel to the filter
- Pipeline involved pigz | nc | pigz
- Produced a series of compressed output files, one per intermediate file
- Generated query counts and unique string list from those

Possible Improvements

- Local disk is slow! Avoid storing uncompressed data?
- Perhaps run one listener per TLD that receives network streams from multiple clients and writes them compressed to disk
- Just expand everything after DITL runs and keep intermediate data around?
- All this seems familiar
- Big Data problems? Well, not that big, but big enough.

Demand Media

What’s Next?

- Pre-process all DITL runs into a Hadoop cluster (or similar)?
  - Hive, Hbase, Cassandra?
- Cluster would be nice
  - Avoid re-inventing the wheel every time someone wants to analyze DITL data
  - Would lets scientists dive right in and analyze
  - Needs hardware and someone to import data
  - Bonus: Distributed archive of DITL data to protect it

What’s Next?

- Why the differences between tcpdump, PacketQ and TShark?
  - Handling of corrupt packets and queries?
  - Other?
- What kinds of raw UTF-8 TLDs are making it to the root (rather than punycode)?
- Help improve PacketQ – it’s damn good!