Report abstract

- This is my last presentation here at CAIA. I guess you want to know, what I have been up to ...
Talk outline

- Where we stopped last time (Review)
- Paper 2: Caching paper
- Paper 1: RTT
- Conclusions
- Questions
- Thanks

Review

- Netsniff deployment
- Traffic analysed
- Logfile parser
- Database

Paper 2: ACE2005
Paper 1: RTT
Conclusion
Questions
Thanks
Netsniff deployment

Netsniff is currently deployed at Irena’s, Jason’s and Sebastian's home.

Traffic analysed

Traffic of the following protocols is currently analysed.

- ARP
- ICMP
- DNS
- HTTP
- TLS
- FTP
- IMAP
- POP3
- SMTP

Traffic characteristics are logged into text files for later processing.
Logfile parser

Output from Netsniff needs to be parsed and stored in a manner, which makes data analysis easy and efficient.

→ Relational database

Currently the back-ends SQLite, SQLite3 and MySQL are supported. Where the later performs a lot better for large and complex queries.

Database

- Current schema has around 40 tables
- Some of them are still unused
Paper 2: ACE 2005
(ACM SIGCHI International Conference on Advances in Computer Entertainment Technology)

Overview

Idea: Derive possible improvements on user experience from the data we have gathered with Netsniff over the last three months, in the context of ICE.

→ “rationale” for web caching

- “A rationale for web caching in consumer ISPs: The impact of DNS lookup times and HTTP session characteristics” - TR

- “A rationale for web caching in consumer ISPs: Evaluating the impact of DNS lookup times and HTTP session characteristics on consumer ISP web traffic” - Paper
“Final results”

Let’s start at the end of the paper ...

- Match HTTP and DNS (bottom two)
- RTT distribution (second and third)
- Object sizes (top part)
Match HTTP and DNS

- $DNS_Q = \pi_{qtime, id}(\delta(...))$
- $DNS_R = \pi_{rtime, id, ip_res}(\delta(...))$
- $HTTP = \pi_{htime, dst_ip}(\delta(...))$
- $DNS_{QR} = DNS_Q \times DNS_R.id = DNS_R.id DNS_{QR}$
- $HTTP_{DNS0} = DNS_{QR} \times DNS_{QR}.ip_res = HTTP.dst_ip HTTP$
- $HTTP_{DNS1} = \sigma_{htime \geq rtime}(HTTP_{DNS0})$
- $HTTP_{DNS} = \gamma(qtime, rtime), \min(htime)(HTTP_{DNS1})$

The actual implementation is a bit more complex. This is due to indexing and the deletion of temporary tables, once they are unused.

Match HTTP and DNS (2)

From the query in the previous slide (and some others), we get the following result:

These show that around 60% are cached in either a browser cache or the operating system. Further 18% are locally cached and for another 22% the DNS server most likely had to recurse to retrieve the data.
RTT distribution

We analyzed the RTT of the TCP streams related to HTTP. The results are shown in the cumulative distribution below.

![RTT Distribution Graph](image)

95% are below 630ms, 50% below 320ms and the average is 348 ms.

Object sizes

The last part is the load time of the objects. We assumed a typical DSL line speed and calculated an estimated load time.

![Object Sizes Graph](image)

The results are very similar to what Sebastian obtained two years ago.
Some other results

**Hop count histogram**

**Content types by count**

- image/gif (3.26%, 39.25%)
- text/html (5.06%, 14.51%)
- image/jpeg (0.07%, 18.34%)
- application/x-javascript (1.17%, 3.43%)
- text/css (0.04%, 1.81%)
- image/png (0.06%, 1.31%)
- text/javascript (0.53%, 0.16%)
- application/octet-stream (0.01%, 0.25%)
- image/x-icon (0.01%, 0.18%)
- application/pdf (0.01%, 0.18%)
- text/plain (0.06%, 0.4%)
- application/x-shockwave-flash (0.01%, 0.64%)
- image/jpeg (0.07%, 18.34%)
- text/html (5.06%, 14.51%)
- image/gif (3.26%, 39.25%)
- application/x-gzip (0.01%, 0.04%)
- multipart/byteranges (0.0%, 0.02%)
- previously cached (0.0%, 8.99%)
Some other results

Content types by size

- text/plain [0%, 27.89%]
- application/x-tar [0%, 16.01%]
- image/jpeg [0.03%, 13.1%]
- text/html [0.53%, 7.84%]
- image/gif [0.07%, 7.41%]
- application/x-gzip [0%, 5.35%]
- video/quicktime [0%, 4.39%]
- video/mpeg [0%, 3.26%]
- application/pdf [0%, 2.24%]
- audio/x-pn-realaudio [0%, 1.56%]
- application/octet-stream [0%, 1.45%]
- application/pdf [0%, 1.36%]
- application/x-shockwave-flash [0.01%, 1.23%]
- application/x-javascript [0.11%, 1.09%]
- application/x-msdownload [0%, 1.2%]
- video/x-ms-wmv [0%, 1.14%]
- audio/mpeg [0%, 1.07%]
- image/png [0%, 1.06%]
- text/css [0.01%, 0.59%]

etc.
Paper 1: RTT

Jason submitted a paper called “Passive TCP Stream Estimation of RTT and Jitter Parameters”, rejected due to “lack of experimental verification”.

- Clients upload (via scp) data to the server. Various configurations were used.
- TCP stack and Netsniff estimated RTTs are observed.
TCP options

We want to compare the TCP stack estimated RTT and Netsniff's estimate.

- Put data to observe in TCP header on clients → modifications in the FreeBSD kernel

- Read data to observe with Netsniff → modifications in Netsniff

TCP kernel modifications

- FreeBSD
  - netinet/tcp.h → magic numbers for new TCP options (2 lines)
  - netinet/tcp_input.c → output of options in TCP header (16 lines)
  - netinet/tcp_hostcache.c → flush hostcache and RTTSCALE (50 lines)
  - Compile, Reboot, done

- Netsniff
  - A couple of changes here and there
  - Easier, since our code, we know where goes what.
Test runs

We run around 132 test configurations. Of which three are shown in the paper.

FreeBSD →

Netsniff →

Conclusion

- Very good and interesting time
- Improved my knowledge of computer networking
- Improved my C++ skills (I come from the Java world ...)
- Got to know quite some knew tools
- Have gained insight in an academic work environment
- I'm top on Google for the search key “Urs Keller” (but only in Australia) :-)
Questions?

Thanks