Lecture 9

When The CRC and TCP Checksum Disagree

Jonathan Stone, Craig Partridge

Advanced Operating Systems

30 November, 2011

- as much as one packet in 1100 can fail the TCP checksum
- this happens even if the corresponding CRC is correct
- it means that transmission links aren't the ones causing the errors
- then who?

- CRC checksum used to detect link-layer errors
- Do we need checksums at every layer? Why?
- One reason is that you can not rely on lower layers doing error checking for you
- Thus, TCP has its own checksum

- TCP computes its checksum by using a pseudo-header
- Why?
- The explanation comes straight from the designer, David Patrick Reed
- http://www.postel.org/pipermail/end2end-interest/2005-February/004616.html

- What happens if we _do_ rely on lower layers for error checking?
- SUN did that
- Because checksumming takes a long time, SUN's NFS implementation disabled it in UDP
- What happened?
- Power fluctuations on busses caused random bits being shuffled
- SUN's current implementation of NFS runs with checksumming enabled

Most important thing to realize

- Never take anything for granted

- capture as many errors as possible
- try to categorize errors that cause checksum failure
- define ways of eliminating those errors

- use libpcap to analyze traffic. The more the merrier
- try to match each bad packet with its retransmission (twin packets)
- look at the error patterns by examining each pair

```
08:27:02.907787 X.X.X.X.22 > Y.Y.Y.Y.38201: P
3286558421:3286558441(20) ack 1212716141 win 25144
    4500 003c d7e4 4000 f506 9029 XXXX XXXX
    YYYY YYYY 0016 9539 c3e4 e6d5 4848 946d
    5018 6238 9e26 0000 0000 000a 7476 b63b
    203f a89e 751f fa39 5e13 f425
```

Figure 1: A Bad Twin ...

```
08:27:02.907787 X.X.X.X.22 > Y.Y.Y.Y.38201:
[tcp sum ok]
3286558421:3286558441(20) ack 3221241833 win 8760
    4500 003c d7e7 4000 f506 9026 XXXX XXXX
    YYYY  YYYY 0016 9539 c3e4 e6d5 0848 d455
    5010 2238 9e06 0000 0000 000a 7476 b63b
    203f a89e 751f fa39 5e13 f425
```

Figure 2: and Matching Good Twin.

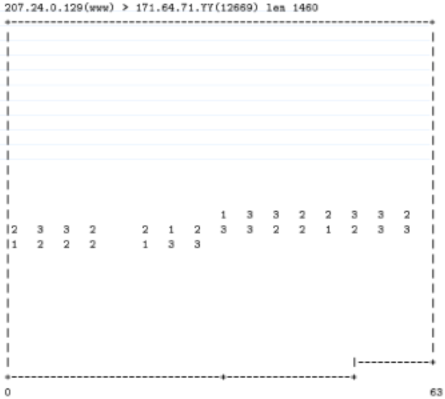

Figure 4: pretty-printer output with every 4th byte bad

- try to morph the good packet into the bad packet
- do this to understand how the error might have occured
- block errors can be caused by buggy DMA engines
- individual byte errors may be caused by UARTs with interrupts for each byte. This can cause overruns on SLIP links.
- try to find similar patterns by manual examination :)
- correlate the patterns with the hardware and software configurations of the network in which you captured the packets

Introduction

Looking for errors

Results

Conclusions

Questions

Trace Name	Total Pkts	Errors	Protocol		
			IP	UDP	TCP
CAMPUS	1079M	33851	0	8878	24973
DoE-LAB	600M	37295	0	173	37122
DORM	94M	11578	1278	613	9687
WEB-CRAWL	436M	396832	0	0	396832
Total		479556	1278	9664	468614

Table 1: Trace Sites and Basic Statistics.

- end-host hardware errors
- end-host software errors
- router memory errors
- link-level errors

- network interfaces may be buggy
  - they may change bits before adding the CRC trailer
  - they may change bits after receiving the packet
  - usually drivers take care of hardware bugs (if possible): http://lxr.linux.no/linux+*/drivers/net/forcedeth.c#L5591
- failures can also affect other hardware components
  - memory errors can occur
  - busses can malfunction
  - see the SUN NFS story above

- ACK-of-FIN bug
- Bad LF in CR/LF
- In conclusion, bugs in software that has direct access to packet structure are bad.

- Same as end-host errors

- Complex interactions cause higher level errors
- Compression algorithms are the most likely cause
- Misinterpretation of RFCs describing these algorithms lead to these errors
- Thus, they can be considered as software bugs too

- Errors might occur that get past both checksums, with the probability:
- $P_{ue} = 1 - P_{ef} - P_{ead} - P_{edp}$
  - P_{ef} – error free packets
  - P_{ead} – errors always detected
  - P_{edp} – errors detected probabilistically

Trace Name	P_{edp}	P_{ue} Range	
		Low	High
DORM	0.0000628404	0.0000000010	0.0000000614
CAMPUS	0.0000090361	0.0000000001	0.0000000088
DOE-LAB	0.0000171166	0.0000000003	0.0000000167
CRAWL	0.0000075436	0.0000000001	0.0000000074

Table 5: Estimated Rates of Undetected Errors

- Don't trust hardware
- Report host errors. ICMP could me modified to do this automatically.
- Report router errors. Use specialized software.
- Protect important data.

- If your application handles sensitive data (financial, military, etc.)...
- You might want to implement some sort of application layer error handling
- Then again, if the code responsible for error handling runs on faulty hardware... :)