Lecture 9
When The CRC and TCP Checksum Disagree

Jonathan Stone, Craig Partridge

Advanced Operating Systems

30 November, 2011
Introduction

Looking for errors

Results

Conclusions

Questions
Introduction

Looking for errors

Results

Conclusions

Questions
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?
Recap

- CRC checksum used to detect link-layer errors
- Do we need checksums at every layer? Why?
- One reason is that you cannot 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
Introduction

Looking for errors

Results

Conclusions

Questions
Important issues

- capture as many errors as possible
- try to categorize errors that cause checksum failure
- define ways of eliminating those errors
Capturing 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
Figure 1: A Bad Twin …

Figure 2: and Matching Good Twin.
207.24.0.129 (www) > 171.64.71.YY (12669) len 1460

Figure 4: pretty-printer output with every 4th byte bad
What to look for

- try to morph the good packet into the bad packet
- do this to understand how the error might have occurred
- 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
Outline

Introduction

Looking for errors

Results

Conclusions

Questions
<table>
<thead>
<tr>
<th>Trace Name</th>
<th>Total Pkts</th>
<th>Errors</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>IP</td>
</tr>
<tr>
<td>Campus</td>
<td>1079M</td>
<td>33851</td>
<td>0</td>
</tr>
<tr>
<td>DoE-LAB</td>
<td>600M</td>
<td>37295</td>
<td>0</td>
</tr>
<tr>
<td>Dorm</td>
<td>94M</td>
<td>11578</td>
<td>1278</td>
</tr>
<tr>
<td>Web-Crawl</td>
<td>436M</td>
<td>396832</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>479556</td>
<td>1278</td>
<td>9664</td>
</tr>
</tbody>
</table>

Table 1: Trace Sites and Basic Statistics.
Error types

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

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

- Same as end-host errors
Link layer 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
Outline

Introduction

Looking for errors

Results

Conclusions

Questions
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

<table>
<thead>
<tr>
<th>Trace Name</th>
<th>( P_{edp} )</th>
<th>( P_{ue} ) Range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>DORM</td>
<td>0.0000628404</td>
<td>0.00000000010</td>
</tr>
<tr>
<td>CAMPUS</td>
<td>0.0000090361</td>
<td>0.0000000001</td>
</tr>
<tr>
<td>DoE-LAB</td>
<td>0.0000171166</td>
<td>0.0000000003</td>
</tr>
<tr>
<td>CRAWL</td>
<td>0.0000075436</td>
<td>0.0000000001</td>
</tr>
</tbody>
</table>

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... :)

Outline

Introduction

Looking for errors

Results

Conclusions

Questions