LinkGuardian
Mitigating the impact of packet corruption loss with link-local retransmission

Raj Joshi, Qi Guo, Nishant Budhdev, Ayush Mishra, Mun Choon Chan, Ben Leong
What is corruption packet loss?

Corrupted (bit flips)

Dropped by the Receiver MAC

Ethernet Checksum Fails

1100101

1101101
What causes packet corruption?

Switch-to-switch Links - Optical
Support high link speeds (up to 400 Gbps)
over long distances (~100m)

Optical links – susceptible to corruption

Fiber Contamination
Airborne dust particles

Fiber Bending

Decaying Transmitters

Photo Credits: Zhou et al., SIGCOMM 2017
Why do we care?

Corruption Packet Loss – Significant

Comparable to Congestion Loss
Large-scale study
(350K links, 15 datacenters)
[Zhuo et al., SIGCOMM'17]

Packet drops affected customers
due to corruption
[Zhou et al., SIGCOMM'20]
Why do we care?

Affects Application Performance
(like any packet loss)

Increase in FCTs
For latency-sensitive flows

Drop in throughput
For throughput-sensitive flows
How can we fix packet corruption?

**Physical Repair**
Several hours to days

**Mitigate**
Effects of packet corruption

Until then
Existing Solutions to Mitigate Packet Corruption

Avoid corrupting links [e.g. NetPilot, CorrOpt]

End-to-end redundancy [e.g. RAIL, CloudBurst]

Link-local redundancy [e.g. Wharf]

- Localize Impact: No network-wide disruption
- Capacity Proportional to corruption loss rate
- End-host Agnostic: No end-host changes

Disabling the links: Network-wide disruption: re-routing of 1000's of flows

- Incur higher reduction in capacity even for small corruption loss rate (e.g. $10^{-3}$)
LinkGuardian

Key Idea

**Link-local Retransmission**
Within the network to recover corruption pkt loss

**Wireless Networks**
Not explored in the context of datacenter networks
Link-local ReTx in DC networks is non-trivial

A **complete** link-local ReTx scheme
- Detect the packets lost
- Hold on the transmission until lost packet is retransmitted
- Put packets back in order to continue transmission

**Challenging:**
- High link speeds
- Dataplane h/w constraints

In this paper – first step

Investigating whether a simple **out-of-order retransmission** scheme could work in datacenter networks
In this talk

- **Small measurement study**
  - Potential “recovery delay”:
    - out-of-order reTx scheme

- **Insights**

- **LinkGuardian**
  - Design & Implementation

- **Evaluation Results**
Recovery Delay for out-of-order ReTx

Simple out-of-order link-local ReTx scheme

Measurement Study
- 10 Gbps network with h/w timestamping on switches

* more details in the paper
**Measurement Study – potential recovery delay**

<table>
<thead>
<tr>
<th></th>
<th>Link-local ReTx</th>
<th>End-to-end (kernel)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>2.59</td>
<td>32.73</td>
</tr>
<tr>
<td><strong>50%</strong></td>
<td>2.59</td>
<td>32.50</td>
</tr>
<tr>
<td><strong>99%</strong></td>
<td>2.60</td>
<td>44.00</td>
</tr>
</tbody>
</table>

Measured Recovery Delays in µs

- Smaller recovery delay → significantly reduce the increase in FCT due to packet loss
Implication for Throughput-sensitive TCP flows

TCP's Reordering Window reduces cwnd

Duplicate ACKs

First assumes, reordering and not loss

@10 Gbps: 3.69µs

Does not immediately react to loss

Reacts only after 3 subsequent data pkts

Link-local recovery delay: 2.60µs (measurements)

We can retransmit a lost packet out-of-order within TCP's reordering window:

→ Prevent triple duplicate ACKs and thus reduction of cwnd
Key Insight

- **Fast out-of-order Recovery** in the network dataplane
- **Loss \(\rightarrow\) reorder** for end-host TCP
- **Prevent performance degradation** by avoiding end-host based loss recovery & reduction of cwnd
LinkGuardian

Network w/ LinkGuardian
Runs normally when no link is corrupting

Monitor the links for packet corruption
Using existing techniques

Activate LinkGuardian
LinkGuardian protocol runs entirely in the dataplane
LinkGuardian Protocol

**Sender Switch**
- seqNo=6
- latestRxSeqNo=6
- Buffer
- latestRxSeqNo=6
- Cumulative ACK

**No Loss Scenario**
(no pkt gets corrupted)

**Receiver Switch**
- latestRxSeqNo=6
- PktGen
- Periodic ACKs

**Diagram Notes**
- Periodic ACKs
- Cumulative ACK
**LinkGuardian Protocol**

**Sender Switch**
- `seqNo=6`
- `latestRxSeqNo=0`

**Buffer**
- `5 4 3 2 1`
- `latestRxSeqNo=0`

**Receiver Switch**
- `latestRxSeqNo=6`

**Loss Scenario**
- (pkt 4 gets corrupted)

**Packet sequence at TCP receiver**
- `6 4 5 3 2 1`

**ReTxRequests**
- `4`

**High priority**

**Low priority**

**Periodic ACKs**

**Loss Detection**

**Loss notification pkt** (high priority)
LinkGuardian Protocol

Loss detection at the receiver switch
Comparing seq # of current packet and the latestRxSeqNo

Packet buffering at the sender switch
Currently achieved through recirculation

Refer to the paper for more details
Evaluation

Implementation
- Intel Tofino ~900 lines of P4

Evaluation setup
- 10 Gbps 3-stage Clos network: Intel Tofino switches and Xeon servers
- Random generator on Tofino: emulate corruption packet loss (different rates)
- TCP flows: CUBIC and DCTCP

Compare
- No mitigation
- Mitigation using Wharf [NetCompute ’18]
  • Provides link-local redundancy
Mitigate impact on throughput-sensitive flows

<table>
<thead>
<tr>
<th>Loss rate</th>
<th>0 (baseline)</th>
<th>10^{-5}</th>
<th>10^{-4}</th>
<th>10^{-3}</th>
<th>10^{-2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBIC</td>
<td>9.49</td>
<td>9.48</td>
<td>7.28</td>
<td>3.43</td>
<td>1.33</td>
</tr>
<tr>
<td>+ Wharf</td>
<td>n/a</td>
<td>9.13</td>
<td>9.13</td>
<td>9.13</td>
<td>7.91</td>
</tr>
</tbody>
</table>

* similar results for DCTCP (see paper)

**Wharf**: FEC redundancy overhead for all packets

**LinkGuardian**: retransmits only the lost packets
Mitigate impact on latency-sensitive flows

FCT distribution (in $\mu$s) for 45KB “affected” DCTCP flows

<table>
<thead>
<tr>
<th>Loss rate $\rightarrow$</th>
<th>0</th>
<th>$10^{-3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>Loss</td>
</tr>
<tr>
<td>min</td>
<td>113</td>
<td>193</td>
</tr>
<tr>
<td>mean</td>
<td>197</td>
<td>707</td>
</tr>
<tr>
<td>50%</td>
<td>180</td>
<td>419</td>
</tr>
<tr>
<td>90%</td>
<td>311</td>
<td>2421</td>
</tr>
<tr>
<td>95%</td>
<td>315</td>
<td>3216</td>
</tr>
<tr>
<td>99%</td>
<td>329</td>
<td>4192</td>
</tr>
</tbody>
</table>

* more results in the paper

↓ ~85%  
↑ 10x
Overheads

Packet buffering at the sender switch
- Packet buffer: 5.44 KB (3-4 packets) @10 Gbps

Periodic ACK packets
- Link capacity overhead: 1%

Overall, the overheads for LinkGuardian remain low.
Future directions

Scalability
– Currently, works for a 100 Gbps link, as long as individual TCP flows <= 10 Gbps
– **Investigate:** can support individual TCP flows > 10 Gbps

Preserve packet ordering (ordered in-network retransmission)
– Beneficial for end-point transports that may not be reordering tolerant (e.g. RDMA)

Buffering packet copies without recirculation
– Next gen programmable switches (Intel Tofino2) provide primitives with which this seems plausible
Conclusion

Packet Corruption Loss
- Can be significant in datacenter networks. Affects application network performance

LinkGuardian

Fast out-of-order Recovery in the network dataplane

Loss $\rightarrow$ reorder for end-host TCP

Prevent performance degradation
By avoiding end-host based recovery & reduction of cwnd

Preliminary results $\rightarrow$ promising approach for mitigating pkt corruption
Thank you!

Credits: This presentation template was created by Slidesgo, including icons by Flaticon.

Flaticon icons by: Becris, Vectors Market, Flat Icons, Maxim Basinski Premium, Darius Dan, noomtah, iconixar, Kiranthastry, Bombasticon Studio, fjstudio, Vectorslab, Chattapat, Freepik, Andy Horvath, itim2101, Pavel Kozlov, srip, Icongeek26, mangsaabguru, kank, and Pixel perfect.