Accurate Timeout Detection Despite Arbitrary Processing Delays

Sixiang Ma, Yang Wang
The Ohio State University
Timeout is Widely Used in Failure Detection
Timeout Detection Can be Inaccurate

When timeout happens, it is hard to tell between:

- sender crash failure
- heartbeat delay

Accuracy: when receiver reports timeout, sender mush have failed. [Chandra, Journal of ACM’ 96]
How to Ensure System Correctness

Approach 1: Paxos-based consensus

- ensure correctness despite inaccurate timeout detection
- high cost and complexity
- examples: ZooKeeper, Chubby, Spanner, etc.
How to Ensure System Correctness

Approach 2: Set long timeout intervals

- system correctness relies on timeout accuracy
- estimate the maximum delay of the communication channel
- examples: HDFS, Ceph, Yarn, etc
- Our work aims to improve this approach
The Dilemma: Availability v.s. Correctness

- **Correctness**: require long timeout to tolerate maximum delays
- **Availability**: prefer short timeout for fast failure detection
The Dilemma: Availability v.s. Correctness

- **Correctness**: require long timeout to tolerate maximum delays
- **Availability**: prefer short timeout for fast failure detection

Can we shorten timeout intervals without sacrificing correctness?
Motivations

1. Long delays in OS and application

2. Their whitebox nature creates opportunities for better solutions
Motivations

1. Long delays in OS and application
2. Their whitebox nature creates opportunities for better solutions
Heartbeat Delay in Our Experiment

- Disk I/O: 10 seconds
- Packet processing: 2 seconds
- JVM garbage collection: 26 seconds
- Application specific delays: several minutes
  - HDFS: directories deletion before heartbeat sending
  - ZooKeeper: session close/expire flooding
Heartbeat Delay Reported in Communities

**CEPH-19335**: MDS heartbeat timeout during rejoin, when working with large amount of blocks.

**HDFS-611**: Heartbeats times from Datanodes increase when there are plenty of blocks to delete.

**ZOOKEEPER-1049**: Session expire/close flooding renders heartbeats to delay significantly.

**HDFS-9901**: Move disk IO out of the heartbeat thread.

**HBASE-3273**: Set the ZK default timeout to 3 minutes.

“Stack suggested that we increase the ZK timeout and proposed that we set it to 3 minutes. This should cover most of the big GC pauses.”

“In extreme cases, the heartbeat thread hang more than 10 minutes so the namenode marked the datanode as dead.”

Heartbeats get blocked by disk in checkBlock() over large regions.”
Delays in OS and Application Are Significant

Compared to default timeout, delays in OS and App are significant

- **HDFS**: 30 seconds
- **Ceph**: 20 seconds
- **ZooKeeper**: 5 seconds
Motivations

1. Long delays in OS and application

2. Their whitebox nature creates opportunities for better solutions
Existing Timeout Views Channel as a Blackbox

- **Blackbox**: only provides information when receiving a packet

![Diagram of network components: Sender, Network, Receiver with layers: App, OS, NIC. Estimated Maximum Delay for Whole Channel is indicated.]
Whitebox Nature of OS and Application

- **Whitebox**: can provide information such as packet pending/drop
Whitebox Nature of OS and Application

- **Whitebox**: can provide information such as packet pending/drop
- Can we utilize whitebox nature to design better solution?
Overview of SafeTimer

• **Goal**: if the receiver reports timeout, the sender must have failed

• **Assumptions** of SafeTimer
  - Delays in whitebox can be arbitrarily long
  - SafeTimer relies on existing protocol for blackbox

• **Solutions**
  - **Receiver**: check *pending/dropped heartbeats* when timeout occurs
  - **Sender**: *blocks sender* when heartbeat sending is slow
Overview of SafeTimer

- **Goal**: if the receiver reports timeout, the sender must have failed

- **Assumptions** of SafeTimer
  - Delays in whitebox can be arbitrarily long
  - SafeTimer relies on existing protocol for blackbox

- **Solutions**
  - **Receiver**: check pending/dropped heartbeats when timeout occurs
  - **Sender**: blocks sender when heartbeat sending is slow
Background: Concurrent Packet Processing

Hardware
- NIC
- RX Queue
- Ring Buffer
- CPU0
- CPU3

Kernel
- Hard IRQ
  - Interrupt
  - Read
  - TCP/IP
- Soft IRQ
  - Backlogs
  - Socket Buffers

User space
- User Thread
Background: Concurrent Packet Processing

Hardware

NIC
RX Queue
Ring Buffer

CPU0

Kernel

Hard IRQ
Interrupt

Soft IRQ
Read

Backlogs

TCP/IP

Socket Buffers

User space

User Thread

User Thread

User Thread

User Thread

Background: Concurrent Packet Processing
Background: Concurrent Packet Processing

Hardware
- NIC
- RX Queue
- Ring Buffer
- CPU0
- CPU3

Kernel
- Hard IRQ
  - Interrupt
  - Read
  - Backlogs
- Soft IRQ
  - TCP/IP

User space
- Socket Buffers
  - User Thread

Receive Side Scaling (RSS)
Background: Concurrent Packet Processing

Hardware:
- NIC
- RX Queue
- Ring Buffer
- CPU0
- CPU3

Kernel:
- Hard IRQ
  - Interrupt
  - Read
  - Backlogs
- Soft IRQ
  - TCP/IP

User space:
- Socket Buffers
- User Thread

Receive Packet Steering (RPS)
Background: Concurrent Packet Processing

Hardware
- NIC
  - RX Queue
  - Ring Buffer
- CPU0
- CPU3

Kernel
- Hard IRQ
  - Interrupt
  - Read
- Soft IRQ
  - Backlogs
  - TCP/IP

User space
- Socket Buffers
  - User Thread
  - User Thread
  - User Thread

Receive Packet Steering (RPS)

Ring Buffer
RX Queue
Challenge: How to Check Pending Heartbeats?

- Multiple concurrent pipelines
- Packet Reordering
Challenge: How to Check Pending Heartbeats?

Pause all threads and check all buffers? ✗
SafeTimer’s Solution: Barrier Mechanism

- Receiver sends barrier packets to **itself** when timeout
- Force heartbeats and barriers to be executed in **FIFO** order

When **barriers** are processed =>

**Heartbeats** arrived before timeout must have been processed
Preserve Per-Ring FIFO Order

Hardware
- NIC
  - RX Queue
  - Ring Buffer
  - CPU0

Kernel
- Hard IRQ
  - Interrupt
  - Read
  - Backlogs
  - TCP/IP
  - STQueue

- Soft IRQ
  - Socket Buffers
  - User Thread

Avoid later-stage reordering
Redirect heartbeats & barriers

User space
- User Thread
Send Barriers to Flush Heartbeats

Hardware

Kernel

User space

Send barriers to each RX queue
Send Barriers to Flush Heartbeats

Send barriers to each RX queue
When Barriers Processed, Heartbeat Processed

Hardware

NIC

RX Queue

CPU0

Interrupt

Ring Buffer

CPU3

Soft IRQ

Backlogs

NIC

STQueue

Per-ring FIFO order preserved

Kernel

Hard IRQ

Soft IRQ

TCP/IP

Socket Buffers

User space

User Thread

User Thread

User Thread

User Thread

User Thread
Overview of SafeTimer

- **Goal**: if the receiver reports timeout, the sender must have failed

- **Assumptions of SafeTimer**
  - Delays in whitebox can be **arbitrarily long**
  - SafeTimer relies on existing protocol for blackbox

- **Solutions**
  - Receiver: check **pending/dropped heartbeats** when timeout occurs
  - **Sender**: blocks sender when heartbeat sending is slow
Problems in Existing Killing Mechanism

- Killing a slow sender is not a new idea, but
- Killing operation itself can be delayed
- Sender alive for arbitrarily long after receiver reports failure

=> Accuracy will be violated
Utilizing the Idea of Output Commit

- A slow sender may continue processing
- As long as other nodes do not observe the effects, the slow sender is indistinguishable from a failed sender [Edmund, OSDI’06]
Block Sender When It Is Slow

- Maintain a timestamp $t_{valid}$ before which sending is valid
- Extend $t_{valid}$ when sender sends heartbeats successfully
  - The definition of “success” depends on the blackbox protocol
- SafeTimer blocks sending if current time $> t_{valid}$
No Need to Include Maximal Delay For Whitebox

- Receiver doesn’t report failure if heartbeats arrived before timeout
- Sender is blocked when sender is slow
Implementation Overview

- Re-direct heartbeats and barriers to \textit{STQueue}
- Send barriers to a specific RX Queue
- Force barriers to go through NIC
- Fetch real-time drop count
- Detect heartbeat sending completion
- Block slow sender
Evaluation Overview

- Can SafeTimer achieve accuracy despite long delays in whitebox?
- What is the overhead of SafeTimer?
Evaluation: Accuracy

- **Methodology:**
  - inject delay/drop at different layers
  - compare with vanilla timeout implementation

- **Result:**
  - SafeTimer can correctly prevent false timeout report
  - vanilla implementation violates accuracy
# Accuracy: Heartbeats Delayed/Dropped on Receiver

**Sender is still alive!**

<table>
<thead>
<tr>
<th>Node</th>
<th>Instrument Position</th>
<th>Injected Event</th>
<th>SafeTimer</th>
<th>Vanilla</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver</td>
<td>System call (recv)</td>
<td>Delay</td>
<td>No timeout</td>
<td>Timeout</td>
</tr>
<tr>
<td>Receiver</td>
<td>Socket (sock_queue_recv_skb)</td>
<td>Delay/Drop</td>
<td>No timeout</td>
<td>Timeout</td>
</tr>
<tr>
<td>Receiver</td>
<td>NFQueue (nfqnl_enqueue_packet)</td>
<td>Delay/Drop</td>
<td>No timeout</td>
<td>N/A</td>
</tr>
<tr>
<td>Receiver</td>
<td>IP (ip_rcv)</td>
<td>Delay</td>
<td>No timeout</td>
<td>Timeout</td>
</tr>
<tr>
<td>Receiver</td>
<td>RPS (enqueue_to_backlog)</td>
<td>Delay/Drop</td>
<td>No timeout</td>
<td>Timeout</td>
</tr>
<tr>
<td>Receiver</td>
<td>Ethernet (napi_gro_receive)</td>
<td>Delay</td>
<td>No timeout</td>
<td>Timeout</td>
</tr>
<tr>
<td>Sender</td>
<td>System call (send)</td>
<td>Delay</td>
<td>Blocked</td>
<td>Alive</td>
</tr>
<tr>
<td>Sender</td>
<td>Socket (sock_sendmsg)</td>
<td>Delay</td>
<td>Blocked</td>
<td>Alive</td>
</tr>
<tr>
<td>Sender</td>
<td>Ethernet (dev_queue_xmit)</td>
<td>Delay</td>
<td>Blocked</td>
<td>Alive</td>
</tr>
</tbody>
</table>
Accuracy: Heartbeats Delayed/Dropped on Sender

<table>
<thead>
<tr>
<th>Node</th>
<th>Instrument Position</th>
<th>Injected Event</th>
<th>SafeTimer</th>
<th>Vanilla</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver</td>
<td>System call (recv)</td>
<td>Delay</td>
<td>No timeout</td>
<td>Timeout</td>
</tr>
<tr>
<td>Receiver</td>
<td>Socket (sock_queue_recv_skb)</td>
<td>Delay/Drop</td>
<td>No timeout</td>
<td>Timeout</td>
</tr>
<tr>
<td>Receiver</td>
<td>NFQueue (nfqnl_enqueue_packet)</td>
<td>Delay/Drop</td>
<td>No timeout</td>
<td>N/A</td>
</tr>
<tr>
<td>Receiver</td>
<td>IP (ip_rcv)</td>
<td>Delay</td>
<td>No timeout</td>
<td>Timeout</td>
</tr>
<tr>
<td>Receiver</td>
<td>RPS (enqueue_to_backlog)</td>
<td>Delay/Drop</td>
<td>No timeout</td>
<td>Timeout</td>
</tr>
<tr>
<td>Receiver</td>
<td>Ethernet (napiGroReceive)</td>
<td>Delay</td>
<td>No timeout</td>
<td>Timeout</td>
</tr>
<tr>
<td>Sender</td>
<td>System call (send)</td>
<td>Delay</td>
<td>Blocked</td>
<td>Alive</td>
</tr>
<tr>
<td>Sender</td>
<td>Socket (sock_sendmsg)</td>
<td>Delay</td>
<td>Blocked</td>
<td>Alive</td>
</tr>
<tr>
<td>Sender</td>
<td>Ethernet (dev_queue_xmit)</td>
<td>Delay</td>
<td>Blocked</td>
<td>Alive</td>
</tr>
</tbody>
</table>
Evaluation: Performance Overhead

- Ping-Pong micro benchmark
  - small overhead (up to 2.7%) for small packets
  - negligible overhead for large packets
- Benchmarks for HDFS and Ceph
  - DFSIO and RADOS Bench
  - negligible overhead
Related Work

• Synchronous systems: HDFS, Ceph, etc.
• Asynchronous systems: Spanner, ZooKeeper, etc.
• Failure detection without timeout:
  - Falcon and its following works [SOSP’11, NSDI’13, EuroSys’15]
  - Work if whole channel is a whitebox
  - Use timeout as a backup
Related Work

- Real-time OS
  - Support: real-time scheduling; prioritized interrupts and threads, etc.
  - Guidelines: implement functions in low layers; pin memory; avoid disk I/Os, etc.
  - Still cannot provide hard real-time guarantees
Summary

- SafeTimer achieves **accurate** timeout detection despite arbitrary processing delays
- Users can set **shorter** timeout intervals without sacrificing accuracy
- The **overhead** of SafeTimer is **small**
Questions?