September 2020

Fast Packet-drop Analysis and Network Self-Recovery

N/A

Fast Packet-drop Analysis and Network Self-Recovery

ABSTRACT

For large networks, e.g., that include hundreds of thousands to millions of network devices, the event of a packet drop somewhere in the network is fairly frequent. Packet drops can result in data loss. Analyzing and recovering from packet drops is currently a time-consuming process. This disclosure describes techniques that incorporate packet-drop intelligence into network devices, such that debugging and root-cause analysis of packet drops is made thorough, efficient, and real-time. A debugging engine is incorporated into a network device such as a switch and runs as part of the switch operating system. The engine leverages the device hardware to perform various counter collections, trap network flows, collect debugging data, inject debug packets, etc. The techniques enable efficient and economic debugging of network issues such as connectivity problems, control-plane protocol time-outs, memory errors, hardware failures, congestion drops, etc.

KEYWORDS

- Packet drop
- Control-plane drops
- Protocol packet drops
- Forwarding decision drops
- Network self-recovery
- Network diagnostics
- Network congestion
- Debug packet
- Recovery workflow

BACKGROUND

The probability of packet drop per network device is typically quite small. However, for large size networks, e.g., that include hundreds of thousands to millions of network devices, the event of a packet drop somewhere in the network is fairly frequent. Packet drops can result in data loss. Analyzing and recovering from packet drops is currently a time-consuming process.

Packet drops can be caused by a variety of reasons. For complex network topologies, identifying the point of loss in the network can be tricky. Narrowing down the exact node where the packet dropped (the root cause of the drop) is tedious. Intermittent drops, e.g., drops that occur occasionally but are hard to reproduce in a controlled setting, are particularly hard to debug. Since packet forwarding is done in the hardware and drops are often tied to various datapath (hardware) pipelines, it is not possible for an external entity to identify dropped packets or network flows.

Network probing or diagnostic analyzers exist that send random sets of packets towards various network devices in the network in an attempt to isolate packet-dropping nodes in the network. If a drop persists for over a certain duration, e.g., five minutes, an alert is raised. Other techniques to isolate packet drops include counters for various packet drops that are collected and analyzed by a central entity such as a network controller. In such techniques, most of the packet drops are not fully analyzed, and the packet-dropping network device is recovered by draining, e.g., redirecting packets away from the device and rebooting it. This is time-consuming and expensive, and can result in loss of data.

DESCRIPTION

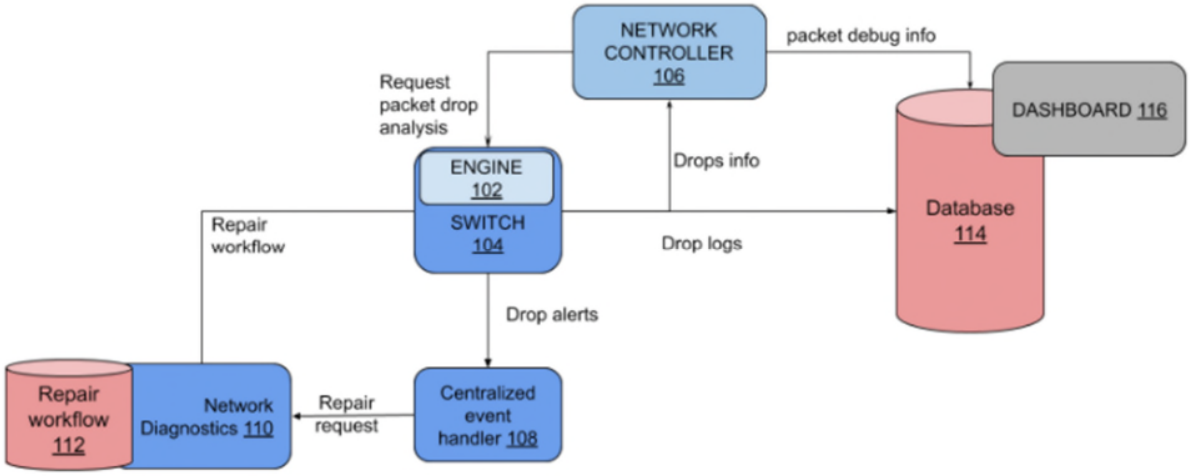

Fig. 1: Fast packet-drop analysis and self-recovery using a debug engine in a network device

This disclosure describes techniques that incorporate packet-drop intelligence into network devices, such that debugging and root-cause analysis of packet drops is made thorough, efficient, and real-time. Per the techniques, illustrated in Fig. 1, a debugging engine (102) is incorporated into a networking device (e.g., a switch or router, 104), and runs as part of the device operating system (OS). The engine utilizes the device hardware to perform various counter collections, trap network flows, collect debugging data, inject debug packets, etc. The engine can be implemented in software, hardware, or a combination thereof.

A network controller (106) manages several network devices. Events of importance, e.g., packet drops within the switch, are reported to a centralized event handler (108) which in turn can make a repair request of a network diagnostic module (110). The network diagnostic module can utilize a variety of repair workflows (112) to effect network recovery. The repair workflow can be highly automated, but in certain cases, e.g., when a physical change in a transceiver is indicated, a human may be involved. A database (114) is utilized to collect event logs, e.g., the

number and kinds of drops, outages, debug information, etc. and is accessible by a dashboard (116),

Per the techniques, packet drops are classified into the following categories:

- *Forwarding-decision drops*: Forwarding decision rules are typically entered in by a human networking operator using a tool. A forwarding-decision drop can be indicative of a misconfiguration.
- *Congestion drops*: Congestion can be caused by a lack of buffers or bandwidth at an (egress or ingress) port of a networking device.
- *Protocol-packet (control-plane packet) drops*: Drops of control-plane packets, which define the network topology and the routing table, has an immediate repercussion on data plane packets. Ascertaining the loss of control-plane packets is time-consuming and difficult, as it entails the awaiting of time-outs and the transmission of trace-route messages to pinpoint the location of the loss. A loss in control-plane packets is often mistaken for the loss of a node, resulting in incorrect changes to network topology at neighboring nodes.
- *Drop caused by memory errors in the networking device:* Although the probability of hardware errors (e.g., memory bits flipping) is quite small, in a network with millions of devices (and tens of millions of memory chips), the event of an undetected (or uncorrected) bit-flip somewhere across the network can be a frequent, even daily, occurrence. Present techniques to detect memory errors include round-robin polling for errors, a process that can take several minutes, equivalent to terabytes of network flow, to uncover a memory error.

As illustrated in Fig. 1, the debugging engine can be controlled by and/or configured through the network controller. For example, the network controller can start or stop packet drop analysis by the engine. Alternatively, the engine can be accessed, configured, and controlled by

directly logging into the network device, e.g., bypassing the network controller, if any. The engine can be configured, e.g., using open-configuration standards, to trap the types of packet drops of interest; to generate particular types of alerts; to take specific actions while observing packet drops; to enable custom drop analysis workflows by an external entity; etc.

The debugging engine enables counter collection on the types of packet drops, e.g., flow-forwarding decision drops for each flow; congestion drops per port per priority; protocol packet drops for each control plane protocol; generic drops based on hardware look-up table decisions; etc. The drop counters are collected and logged using registered, configurable, interfaces.

When packet drops persist beyond a certain (configurable) threshold interval, the engine samples the dropped network flow at a (configurable) rate. The network-sampling rate can be based on performance; a typical rate might be, e.g., one packet per flow per minute.

The dropped flows are logged and, for further analysis, the registered entity is notified of such flows. Information pertaining to the packet flow is pushed to the engine from the hardware for analysis that can include parsing the information and running configured workflows. Some examples follow.

- *Workflow for congestion drops*: Packets dropped due to congestion can be reported to a network controller, monitoring entity, or human operator. Such a monitoring entity or human operator can ease congestion, e.g., by deprioritizing certain types of packets or selectively dropping certain types of packets.
- *Workflow for forwarding decision drops*: Packets dropped due to forwarding decisions are traced to forwarding-decision rules.
- *Workflow for protocol-packet drops*: Packet drops traced to protocol-packet drops are reported to neighboring network devices, the network controller, etc. Proactively informing

neighboring nodes of a loss in protocol packets forestalls the formation of an incorrect image of network topology at the neighbors. For example, such reporting can prevent the neighbors from the incorrect deduction that the node that lost protocol packets is now unreachable.

- *Workflow for memory-error drops*: In case of a packet drop attributable to a memory error, a copy is made of the header (and associated metadata) of the packet and forwarded to a network-monitoring entity. Forwarding tables are examined to determine the source of the memory error. Hardware tables are checked for memory errors by comparing the configurations between software and hardware. If there is a mismatch between hardware and software configurations, then the hardware table is corrected in accordance with the software information. The described workflow thereby arrests the loss of packets to just a few packets, rather than terabytes of network flow. Also, a reboot of the network device, currently a common yet disruptive response to memory-error packet drops, is reduced or eliminated.

A debug packet, e.g., a dummy packet with a custom header atop a packet from the sampled flow, is crafted and injected into the hardware to mimic the arrival of a packet from the same ingress port as the original flow packet. In this manner, the original flow is mimicked, various statistics (e.g., table look-up details, actions taken on the packet, etc.) from the hardware are collected, and a decision is reached for the flow. To enable further offline analysis, the engine also stores in the database debug data collected for the flow and actions taken from the workflow.

When the drop tolerance threshold is met, the engine also generates a list of configurable alerts. These alerts can be consumed by various external entities to invoke their workflows. For example, a repair workflow can be triggered in response to a hardware failure.

The described engine supports use of an allow list (whitelist) for drop types, e.g., drop types that are ignored as expected or benign drops. The engine also supports a plugin model for the analysis workflow for various drop types.

Some analysis workflows that can be supported by the engine include:

- Check configuration, e.g., for forwarding-decision drops.
- Notify the controller for protocol packet drops.
- Invoke memory reconciliation for memory errors.
- Generate alerts for drop types (including congestion drops), and transmit alerts to an event-handling entity or other registered external entity.
- Log information from packet drops, including information from sample and debug packets.

In this manner, the techniques of this disclosure enable the specification and collection of packet-drop statistics for each network flow or table in the network device itself. The techniques define a drop tolerance threshold that can be used to trigger drop analysis, logging, alerts, etc. A dropped packet (sample flow) can be trapped from hardware to software. A debug packet specific to the hardware, e.g., with custom header, can be crafted to simulate the sample flow in the hardware processing pipeline to collect details pertaining to the processing of the dropped packet. Debug data pertaining to the debug packet can be analyzed and appropriate recovery workflows invoked to mitigate the packet drops using internal processing, external processing, or a combination thereof.

The described packet-drop analysis engine, when deployed in a production network, enables debugging of issues such as connectivity problems, control-plane protocol time-outs, memory errors, hardware failures, congestion drops (per port per queue level, in real-time), etc. The described techniques result in more thorough debugging; faster packet-drop detection;

reduction in use or elimination of an external network-probing entity (leading to substantial savings); the possibility of hardware acceleration for debugging tasks such as counter collection, packet trapping, debug-packet injection (leading to faster performance); etc.

CONCLUSION

This disclosure describes techniques that incorporate packet-drop intelligence into network devices, such that debugging and root-cause analysis of packet drops is made thorough, efficient, and real-time. A debugging engine is incorporated into a network device such as a switch and runs as part of the switch operating system. The engine leverages the device hardware to perform various counter collections, trap network flows, collect debugging data, inject debug packets, etc. The techniques enable efficient and economic debugging of network issues such as connectivity problems, control-plane protocol time-outs, memory errors, hardware failures, congestion drops, etc.