CPSC 416 Distributed Systems

Winter 2023 Term 1 (October 3, 2023)

Tony Mason (fsgeek@cs.ubc.ca), Lecturer

Logistics

Teaching Assistants

Andy Hsu (andy.hsu@alumni.ubc.ca)

Hamid Ramezanikebrya (hamid@ece.ubc.ca)

Jonas Tai (jonastai@student.ubc.ca)

Cathy Yang (kaiqiany@student.ubc.ca)

Office Hours

Remember: Use Piazza for **all** official course-related communications

- Not on Piazza? Not official.
- Canvas “comments/messages” **are not monitored**

Office Hours:

Who	When	Where
Tony	Monday 14:00-15:00 Wednesday 16:00-17:00	Discord
Andy	Thursday 19:00-20:30	Discord
Hamid	Friday 16:30-18:00	Kaiser 4075
Jonas	Thursday 11:00-12:30	X150, Table 1&2
Cathy	Friday 09:00-10:30	X237

Self-Assessment

This week

- Post-lecture self-assessment activity – Due Thursday (October 5 @ 17:00)

Next week

- Post-lecture self-assessment activity – Due Tuesday (October 10 @ 17:00)
- **No class Thursday (October 12, 2023)** – Monday classes instead!

Note:

- You are strongly encouraged to collaborate with others on this
- You should use tools at your disposal to answer these questions
- **Do not forget to submit it.**

Today’s Failure

Google Cloud Platform

Event start: June 2, 2019 @ 11:45 PT

Event ends: June 2, 2019 @ 15:40 PT

TL;DR Version

- Network connectivity issues
- Degrade/Disrupt
    - Compute Engine (up to 33% packet loss)
    - App Engine (23.2% decrease in requests per second) – many timeouts
    - Cloud Endpoints – 50% of service configuration push workflows failed
    - Cloud Interconnect – packet loss up to 100%
    - Cloud VPN – multiple gateways unreachable
    - Cloud Console – slow/failed page loads
    - Etc. (6 more impacted services including G Suite)

Cause

Two “normally benign” misconfigurations

One software bug

(1) Network control plane jobs configured to be stopped for maintenance event
(2) Cluster management software instances included in “relatively rare mainentance event”
(3) Maintenance event software initiator bug: allows descheduling multiple independent software clusters *at the same time*. Clusters were geographically distributed.

Mitigations

Network data plane continues to run without control plane presence.

BGP routing stopped: connectivity interrupted (“can’t get there from here”)

Google engineering knew of the issue within *two minutes*.

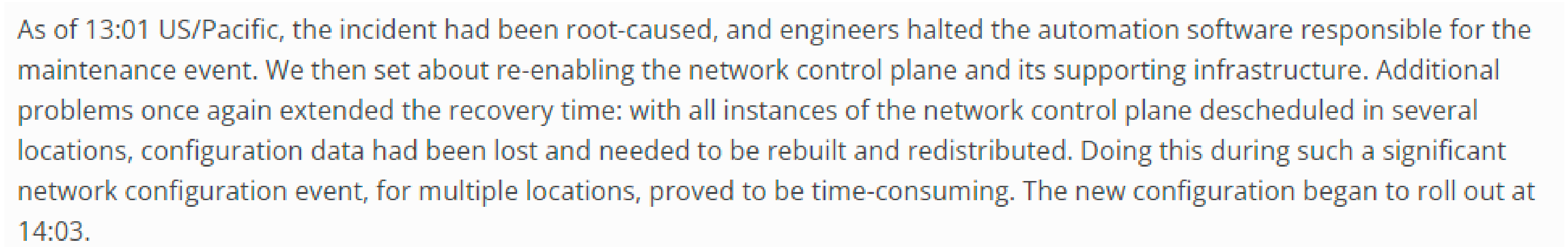
Tools don’t work: “Debugging the problem was significantly hampered by failure of tools competing over use of the now-congested network.”

Mitigations

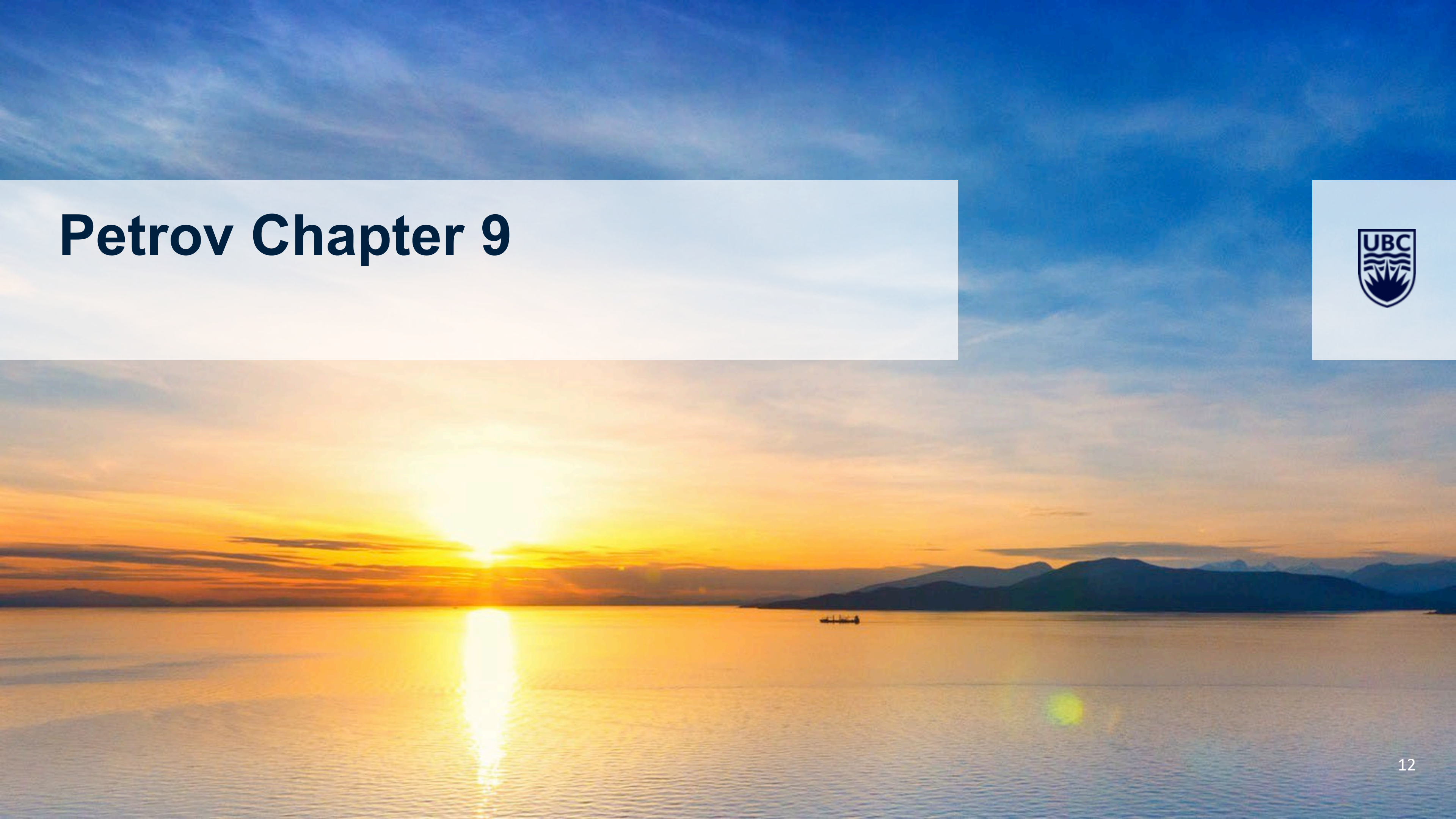
When your communications fail…

“The defense in depth philosophy means we have robust backup plans for handling failure of such tools, but use of these backup plans (including engineers travelling to secure facilities designed to withstand the most catastrophic failures, and a reduction in priority of less critical network traffic classes to reduce congestion) added to the time spent debugging. Furthermore, the scope and scale of the outage, and collateral damage to tooling as a result of network congestion, made it initially difficult to precisely identify impact and communicate accurately with customers.”

Recovery


As of 13:01 US/Pacific, the incident had been root-caused, and engineers halted the automation software responsible for the maintenance event. We then set about re-enabling the network control plane and its supporting infrastructure. Additional problems once again extended the recovery time: with all instances of the network control plane descheduled in several locations, configuration data had been lost and needed to be rebuilt and redistributed. Doing this during such a significant network configuration event, for multiple locations, proved to be time-consuming. The new configuration began to roll out at 14:03.

Cascading failures – again. This is a common (and recurring theme) in distributed systems failures.



Petrov Chapter 9

Learning Goals (Petrov Chapter 9)

Understand the fundamental challenges in failure detection in distributed systems.

Learn about different failure detection models and their limitations.

Explore practical scenarios where specific failure detection models are applicable.

Failure Detection

What is failure detection?

Why is it important in distributed systems?

Limitations:

- No global clocks (most of the time)
- Myriad of failures:
  - Node failure
  - Network partition
  - Network misbehaviour (dropped packets, reordered packets)

Heartbeats and Pings

Scenario: A simple distributed system where immediate detection of node failures is crucial, e.g., a load-balanced set of web servers.

Explanation: Heartbeats and pings are simple and effective for detecting failures quickly in scenarios where nodes need to be aware of each other's status in real-time.

Timeout-free Failure Detector

Scenario: Systems where network latency is variable and unpredictable, e.g., a globally distributed database system.

Explanation: A timeout-free failure detector can be beneficial in environments with high network variability as it doesn’t rely on predefined timeout values to detect failures.

Outsourced Heartbeats

Scenario: A system with resource constraints where offloading heartbeat processing is essential, e.g., IoT devices in a smart home environment.

Explanation: Outsourced heartbeats can help in reducing the load on the resource-constrained devices while maintaining effective failure detection.

Phi-Accrual Failure Detector

Scenario: Systems requiring adaptive failure detection sensitivity, e.g., a microservices architecture with varying workloads.

Explanation: The Phi-Accrual Failure Detector can adapt to the system's behavior and workload, providing more flexibility and accuracy in detecting failures.

Gossip and Failure Detection

Scenario: Large-scale distributed systems where scalability is crucial, e.g., a peer-to-peer file-sharing network.

Explanation: Gossip protocols can efficiently disseminate failure information in large-scale systems, ensuring that failure detection scales with the system size.

Reversing Failure Detection Problem Statement

Scenario: Systems where understanding the failure detection from the perspective of the detector is crucial, e.g., a monitoring system observing multiple services.

Explanation: Reversing the problem statement can provide insights into the reliability and availability of the system from the detector’s viewpoint.

Note: instead of detecting *failure* we detect *liveness*.

Questions?

THE UNIVERSITY OF BRITISH COLUMBIA