The 1990 AT&T Long Distance Network Collapse
ELEN3020A - Assignment 1

Jonathan Taylor - 1665909

25 February 2019

Introduction

Software failures and bugs in software solutions have caused billions of dollars of losses since the beginning of replacing previous systems with software solutions by big companies in the 1940s and '50s [2]. Some of the most expensive failures have come from very reputable companies and Government agencies such as the U.S. Internal Revenue Service, Nike, and McDonald's. These software failures often occur for similar reasons. These include sloppy development practices, poor project management, stakeholder politics, commercial pressures and poor communication among customers, developers, and users [2].

Description

One of the most interesting of these software failures was in the 1990s, when AT&T upgraded the software of their entire national long distance call system in the United States and erroneously turning off the entire network for nine hours as of a software bug in the new code. This resulted in a $60 million lost as of 75 million missed phone by AT&T customers calls and 200 000 airline and hotel reservations and other businesses that relied on the telephone network [1].

The details of why this happened are as follows. AT&T had 114 transceiver centers all over the USA that had the job of processing and routing calls and communicating with all the other transceiver centers. One of these transceiver centers or 'switches', let's say Switch 1, had a minor mechanical problem on the 15th of January 1990. Protocol says that this switch should message a congestion signal (telling other switches Switch 1 was too busy and shouldn't be messaged until it was less busy) to all the switches it was connected to if there was some problem with the switch. Therefore, this is what Switch 1 did.

Before the upgrade, when a switch had come back online after being fixed, a switch would send out a message to switches it was connected to telling other switches it was back online, which would cause all the switches Switch 1 was connected to reset themselves. This would allow these other switches to now interact with Switch 1. However, after the upgrade, Switch 1 would not have to send a separate message saying it was in service, but simply start messaging the other switches as usual, and the other switches would detect this from Switch 1 and reset themselves to start interacting with Switch 1 again [1]. This upgrade allowed for a faster reset time and therefore faster routing speeds and calls for customers.

What went wrong and why

The problem occurred when Switch 1 reinitialized itself after resetting from the upgrade and messaged another switch, say Switch 2. Switch 2 reset itself to acknowledge Switch 1's return and while it was resetting, Switch 1 sent a second message to Switch 2. Switch 1 did this as the upgraded code meant messaged were also sent with a shorter break in-between than before the upgrade. This second message caused Switch 2 to think it needed to reset itself again, causing Switch 2 to sent out the congestion signal. Once Switch 2 was back, it sent a message to Switch 3, and then a second to Switch 3, causing Switch 3 to send out the congestion signal. This cascaded through all 114 switches, making them constantly reset and send signals during others resets for 9 hours [3]. This only stopped when engineers at AT&T found out what was going on and network loads were low enough to allow the system to stabilize. The engineers could then shut down the entire network and fix the problem. By this stage, a fortune had been lost due to a single buggy line of code.

```
1   while (ring receive buffer not empty
           and side buffer not empty) DO

2     Initialize pointer to first message in side buffer
      or ring receive buffer

3     get copy of buffer

4     switch (message)

5        case (incoming_message):

6              if (sending switch is out of service) DO

7                  if (ring write buffer is empty) DO

8                      send "in service" to status map

9                  else

10                     break

                   END IF

11            process incoming message, set up pointers to
              optional parameters

12            break
        END SWITCH


13    do optional parameter work
```

Figure 1: Pseudocode of the bug that crippled the AT&T network [1]

This buggy line of code was a break statement found in an if clause nested in a switch clause in the upgraded recovery software of all 114 switches. When the destination switch received the second of the two closely sent messages while it was still busy with the first message (buffer not empty, line 7), the program running should have exited out of the if clause (line 7), handled the incoming message, and set up the pointers to the database (line 11). However, because of the break statement in the else clause (line 10), the program exited out of the case statement completely and began doing the optional parameter work which overwrote the data (line 13). Error correction software saw the overwrite and shut the switch down while it could reset [1]. Because every switch contained identical software, the resets cascaded down the network, crippling the entire AT&T call system.

The AT&T Long Distance call system had been designed so that a single switch could not crash the entire system. The software also had features that would "heal" defective switches, so they could not shut down other switches. Each switch would also monitor other switches it was connected to.

This would allow switches to determine if their neighbor switches were reliable or not. However, in this case all the switches became unreliable at the same time, meaning no switch could mark others as defective or heal any of the other switches, resulting in a failure of the entire system.

What could realistically have been done to avoid the problem

This software bug was unfortunately easy to remain undetected as of the fact it was coded and compiled in C. This meant that this particular problem was not as obvious as it would be in a structured programming language with a stricter compiler such as C++. The other factor that contributed was the fact that the switches reset themselves when a change in the system occurred, such as a new switch coming online. This is a vulnerable practice as it is an easy way to cripple switches. A more fault-tolerant hardware and software system that could handle minor problems without resetting would have significantly reduced the effects of a software bug [1].

Furthermore, the software update loaded in the AT&T network had already passed through a multitude of tests and had remained undetected through the busy Christmas season. AT&T is fanatical about its reliability of its systems and so sloppy testing and debugging cannot be to blame for this bug [1].

Fortunately, this software bug is only famous as of its rarity, many similar bugs have been detected before release, or deterred by the fault-tolerance system in place in the AT&T software. This was also at a time that using large scale software solutions was fairly new, therefore much better standards for these types of systems and stricter compilers now exist. This network collapse acted as a case study for how bugs in self-healing software can cripple healthy systems, and the hardships of detecting obscure load- and time-dependent defects in software [1]. Therefore, these types of software bugs will hopefully never happen again.

Conclusion

The AT&T long distance call system failure of 15 January 1990 cost millions of dollars due to a software bug in the software update to increase the speed of the phone network. This software bug caused all the switches in the network to constantly reset themselves, meaning 50% of the calls placed on the AT&T network failed to go through. This happened for 9 hours until traffic was low, the system stabilized and engineers could fix the problem. This problem is famous, thankfully, as of its rarity - the AT&T network is known for its reliability to this day. The failure gave new insight into self-healing software bugs, and the difficulty in detecting obscure load- and time-dependent defects in software.

References

[1] Dennis Burke. All circuits are busy now: The 1990 at&t long distance network collapse, 1995.

[2] Robert N. Charette. Why software fails, 2005.

[3] Nicole Gawron. Infamous software bugs: At&t switches, 2014.