June 2020

On-Device WiFi Health Monitor

Kai Shi

Ning Zhang

Kumar Anand

Etan Cohen

ON-DEVICE WIFI HEALTH MONITOR

Abstract

A population of user devices employs on-device WiFi health connectivity monitors to allow for efficient and accurate detection of WiFi connectivity issues caused by the device software itself, thereby facilitating rapid software version rollback or other corrective software revisions to mitigate WiFi connectivity issues caused by the device software.

Background

WiFi connection failures and disconnections at a user device typically are a result of poor signal quality between the user device and a corresponding WiFi access point (AP). However, issues with the AP itself or with the software (including firmware) of the user device can lead to WiFi connectivity issues even when the device-AP signal quality is sufficient. Software-based connectivity issues can be classified into the following three categories:

- Regression – connectivity issues that only manifest after a new software release;
- Improvement – connectivity issues are present only in older software releases and disappear after a new software release; and
- Non-specific – connectivity issues manifest in both previous and new software releases

Some triage and analytic approaches for addressing WiFi connectivity issues rely on raw data (such as raw connection event logs) uploaded by user devices so as to monitor the overall device population health state and to detect any potential software issues. However, these approaches are susceptible to incomplete or inaccurate analysis. For example, regression-type connectivity issues sometimes only manifest in a relatively small device sub-population and thus may not be apparent from metrics aggregated over the entire device population. Similarly, performance degradation due to regression-type connectivity issues may only appear with certain models of

APs, which makes ready detection by cloud data analysis more difficult. Moreover, some user devices may already be experiencing WiFi connectivity issues independent of a software update, which can lead to such instances incorrectly being identified as a regression-type connectivity issue.

Description

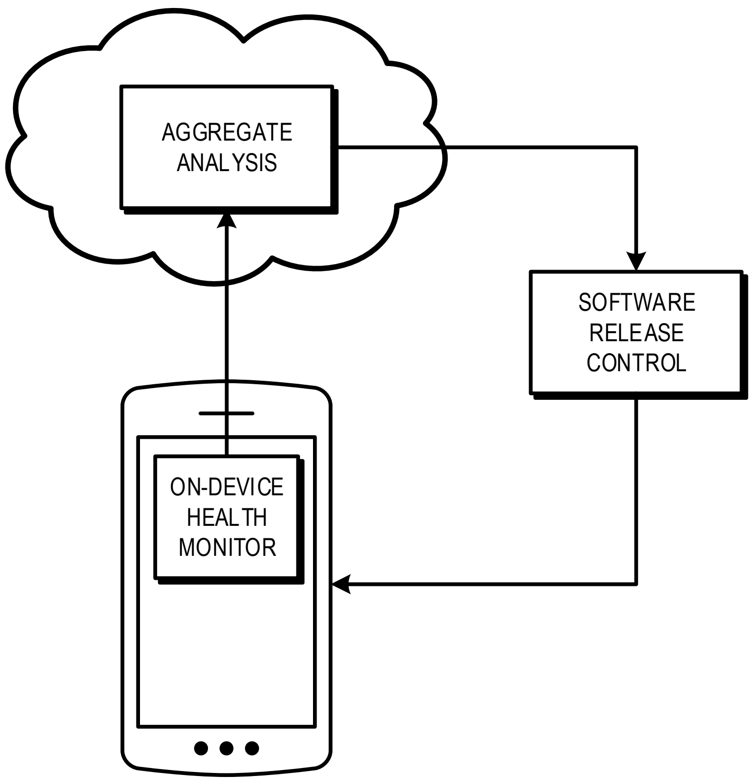

Figure 1 – WiFi Health Monitor Workflow

Figure 1 above illustrates the workflow for a technique for monitoring WiFi health and detecting and classifying software issues with only a relatively small data set size and with high reliability while maintaining user privacy. The foundation for this approach is the implementation of a connectivity health monitor module on the user device (henceforth referred to as the “on-device health monitor”) that operates to detect WiFi connectivity health-related issues at the user device, which could be either specific to the software release implemented at the user device or independent of the software release. For example, one output that can be provided by the on-device health monitor can include a count of WiFi networks with a significant change in connection statistics, which indicates a software-dependent connectivity issue. Another example output can include a count of WiFi networks with a high connection failure rate, which indicates a software-independent connectivity issue. The outputs of the on-device health monitor are transmitted to a network destination of a developer of the software, a network operator, or other entity, whereupon the outputs from the on-board health monitors of a population of user devices are aggregated and the resulting aggregated data is subjected to analysis, which can include use of a deep neural net (DNN) or other machine-learning-based technique. If the result of the analysis indicates that a relatively significant number of WiFi networks are experiencing connectivity issues, it is likely that the cause is related to a software regression issue with the user devices. If so, the software provider can trigger a rollback of the software to an earlier version or the development of a new release intended to address the connectivity issue caused by the current software version.

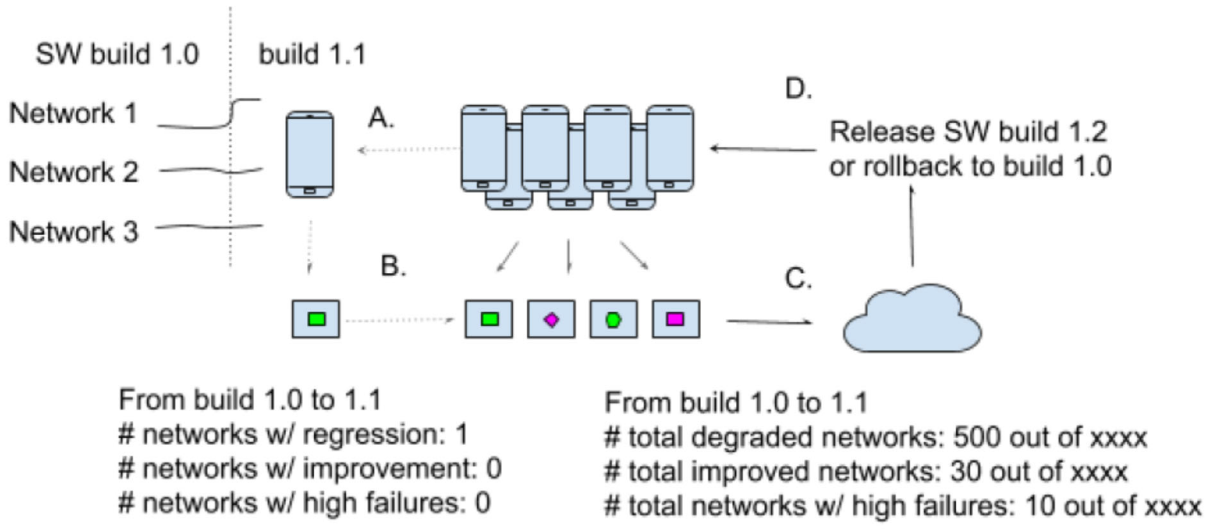
SW build 1.0
build 1.1
Network 1
Network 2
Network 3
A.
B.
C.
D.
Release SW build 1.2
or rollback to build 1.0
From build 1.0 to 1.1
# networks w/ regression: 1
# networks w/ improvement: 0
# networks w/ high failures: 0
From build 1.0 to 1.1
# total degraded networks: 500 out of xxxx
# total improved networks: 30 out of xxxx
# total networks w/ high failures: 10 out of xxxx

Figure 2 – Example Regression Detection

Figure 2 above illustrates an example of software regression from software build 1.0 to build 1.1 for a population of user devices. In this example, a subset of user devices detects WiFi networks experiencing regression-type connectivity issues (step A) and thus their on-board health monitors report this connectivity issue (step B). Although this connectivity issue is not manifesting at all user devices in the population, a significant number of impacted user devices in the aggregated data (step C) indicates regression-type connectivity issues caused by build 1.1 with a high confidence. Accordingly, the build control team decides to release a software build 1.2 or rollback to old build 1.0 for the user devices in this population (step D).

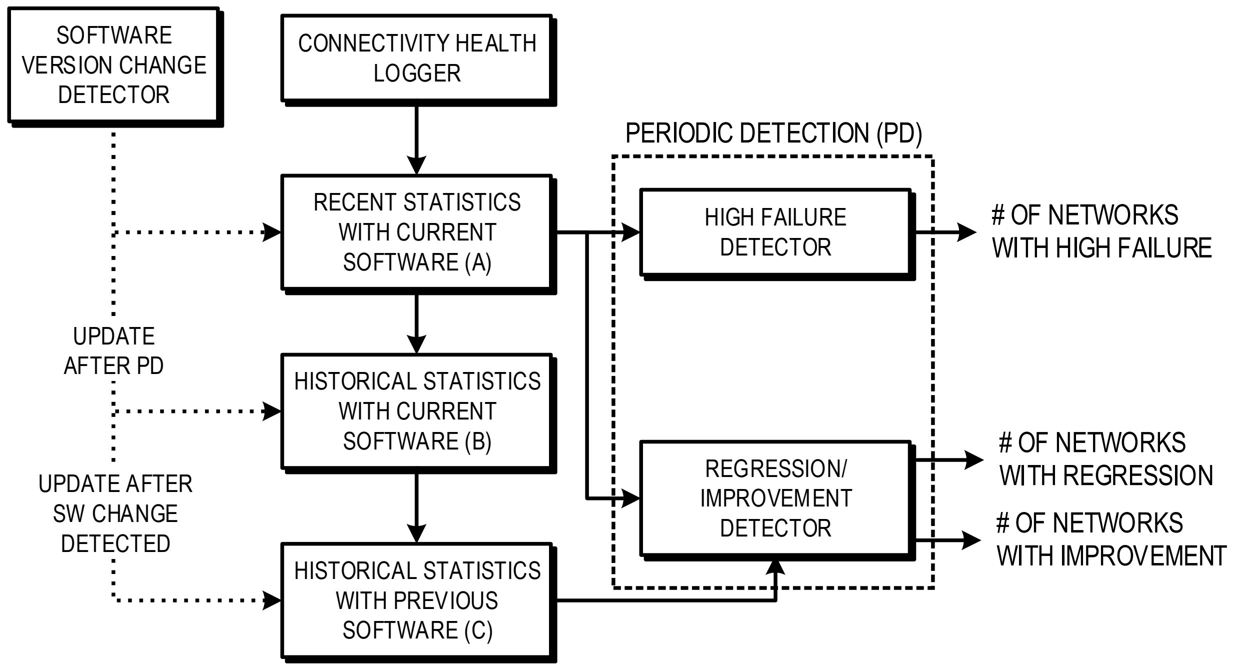
SOFTWARE VERSION CHANGE DETECTOR
CONNECTIVITY HEALTH LOGGER
PERIODIC DETECTION (PD)
RECENT STATISTICS WITH CURRENT SOFTWARE (A)
HIGH FAILURE DETECTOR
# OF NETWORKS WITH HIGH FAILURE
UPDATE AFTER PD
HISTORICAL STATISTICS WITH CURRENT SOFTWARE (B)
REGRESSION/ IMPROVEMENT DETECTOR
# OF NETWORKS WITH REGRESSION
# OF NETWORKS WITH IMPROVEMENT
UPDATE AFTER SW CHANGE DETECTED
HISTORICAL STATISTICS WITH PREVIOUS SOFTWARE (C)

Figure 3 – On-Device Health Monitor

Figure 3 above illustrates an example implementation of the On-Device Health Monitor. The solid lines represent the data path while dotted lines represent the control path. During normal operation, a connectivity health logger collects connectivity-related statistics such as, for example, a count of connection attempts and normal/abnormal failures, and a count of normal/abnormal disconnections. The failure count may break down to each unique failure cause. The abnormal connection failures are typically defined as failures at high receive signal levels. Similarly, the abnormal disconnection failures may be defined as disconnections at high receive signal levels or high transmit/receive link speeds. The abnormal connection failure or disconnection can be further limited to, for example, connection failure with high received signal strength indicator (RSSI) with a scan done a few seconds before the connection attempt, or a not-host-initiated disconnection with a high RSSI or a high transmit speed measured a few seconds before the disconnection. The data collected by the logger is typically saved in a persistent memory of the user device so that the data is still available after device reboot. The data is initially saved in a bucket representing the recent statistics with current SW (bucket A). The collected data then may be moved to other buckets after periodic detection (PD) or a version change is detected. Bucket B stores historical statistics for the current software version. Therefore, after a PD is completed, the data in bucket A is moved to bucket B. Bucket C stores historical statistics of the previous software version. If a new software version change is detected, data in bucket B is moved to bucket C while data in bucket A and B are cleared. Note that since the connectivity issues are mostly network specific, the data buckets and data collection are implemented on a per-network basis.

A software version change detector can use the following information for detecting software changes: on-device operating system (OS), driver, or firmware version; and network-side software/firmware version (collected based on network vendor specific information announced in beacon or other management frames). During periodic detection, the data in bucket A and C are used for analysis. Thus, for each network, if there is sufficient data in bucket A but insufficient data in bucket C, this implies that the corresponding WiFi network does not have sufficient usage with the previous software version, and thus only data in bucket A should be used to detect any abnormally high connection or disconnection issues. The determination of whether there is sufficient data can be made by, for example, checking if there is a sufficiently large number of connection attempts at high RSSI or disconnection at high RSSI/transmit speed. One possible implementation could define the following abnormal connection failure and disconnection rates: abnormal connection failure rate = the ratio between the number of connection failures at high RSSI and the total number of connection attempts at high RSSI; abnormal disconnection rate = the ratio between not-host-initiated disconnections (or all connections) at high RSSI or high transmit speed and the number of all disconnections at high RSSI or high transmit speed. The on-device health monitor then reports detection if the abnormal connection failure ratio or abnormal disconnection ratio is above a predefined threshold.

However, if there is enough data in both bucket A and C, this means that the corresponding network has sufficient usage with the current and previous software versions. As such, the detection of network regression can be performed by checking whether failureRate > thrHigh(bucket A) and failureRate < thrLow (bucket C), where thrHigh and thrLow are pre-defined thresholds. Alternatively, the regression detection can be performed by checking whether failureRate(bucket A) / failureRate (bucket C) > threshold. A more sophisticated detection could use a sliding window for both bucket A and C. In either case, only the recent K (a fixed number) connection events are counted. In this approach, detection of network regression is triggered when meanFailure (bucket A) > meanFailure (bucket C) + std (bucket C)

* thrNumSigma, whereby “mean” and “std” refers to the mean and standard deviation, respectively, of abnormal failure counts, and thrNumSigma is a detection parameter representing the number of sigma (a.k.a., std). Similar to the simple counter approach, the sliding window of bucket A is flushed after a software version change is detected or periodic detection is done. Note that while the example above utilizes two buckets, other implementations may use more buckets and maintain statistics data for older software versions. Moreover, some implementations may collect and use data at a service set identifier (SSID) level (that is, per network name) while other implementation may choose to collect and use data at a basic service set identifier (BSSID) level (that is, on a per-AP basis).

References

1. U.S. Patent Application Publication No. 20160189441, entitled “Apparatus and Method of Error Monitoring with a Diagnostic Module” and filed on March 3, 2016, the entirety of which is incorporated by reference herein.
2. U.S. Patent Application Publication No. 20060233114, entitled “Method and apparatus for performing wireless diagnostics and troubleshooting” and filed on April 15, 2005, the entirety of which is incorporated by reference herein.
3. U.S. Patent Application Publication No. 2014028092, entitled “Device and method for detecting and visualizing network health” and filed on March 17, 2014, the entirety of which is incorporated by reference herein.