Summary:

Objective:

1. Measurement of network disruptions and significant signal strength fluctuations on 2000 smartphones.
2. How popular mobile apps today handle network instabilities?
3. Janus an interface management framework that
   1. exploits the multiple interfaces to handle network disruptions;
   2. reduces video stalls by 2.9 times and increase 31% of the time of good voice quality;
   3. reduces traffic size by 26.4% and energy consumption by 16.3% compared to naive solutions.

Hypothesis:

1. Network Instabilities: User mobility inevitably results in significant signal strength changes and network disconnection and reconnection (mere high bandwidth and low latency are not enough).
2. Signal strength fluctuation is normal in daily usage, 10 dBm is a threshold that
3. denotes significant strength changes that can impact upper-level user experiences.

Limitation:

Most apps are unaware of the actual network performance and passively follow the OS WiFi-if-available.

Detailed Summary:

Methodology:

* Why few of today’s popular apps handle network disruptions or exploit multiple interfaces to improve the user experience.
  + Before Android 5.0, the interface selection was completely controlled by the OS.
  + Inheritance From the PC Era: The network stack and APIs used in current mobile systems are mostly inherited from the ones designed in the wired environment.
  + Android maintains an active network inside the operating system that is set to WiFi-if-available and offers no support for developers to change the network interface to fit their needs.
  + An Outdated Assumption
  + Current Solutions Are Ill-Suited for Handling Network Disruptions:
    - insufficiency of current network API due to potential instability of IPv6 routes.
    - Misuse or absence of available libraries due to complexity.
* This paper proposes to address the issue by providing direct support inside the OS.
* Design Options of Interim Solution:
  + Avoid the inefficiency problem caused by misusing power control APIs.
  + Reduces the overhead of handling network disruptions.
  + Allows data transfers from different processes to be clustered, improving the opportunity of the device turning off the radio.
  + OS layer solution is compatible with existing apps and does not require modifying the app source code.
* User-Dependent vs. User-Independent: We design Janus as an user-dependent solution, i.e., allowing user customization, out of two considerations.
* Principles of Handling Disruptions: Janus transparently handles:
  + transient network disruptions for both foreground and background tasks.
  + permanent disruptions for background tasks.

Experiment:

* A: Deployed a utility app (Anonymous App) in Google Play store that has been downloaded on over 100,000 handsets.
* The utility app performs periodic logging of the network usage of all apps running on each phone every 5 seconds during screen-off when CPU is on and every 1 second during screen-on.
* Our trace data contains logs from 2000 Android devices.
* B: Examine how existing mobile apps react to dynamic network conditions and network disruptions, by studying the behavior of 30 popular apps selected from Google Play on a Nexus 6 smartphone running Android 6.0.
* C: We examine whether the apps are able to use the best network to obtain good performance under three scenarios:
  + Low WiFi
  + Diminishing WiFi.
  + Congested network load
* D: Examine how existing apps react to network disruptions.
* E: Evaluate how the Request Manager helps reduce energy through flexible request scheduling.
* F: Microbenchmark to evaluate the basic performance of Janus as an integrated system.

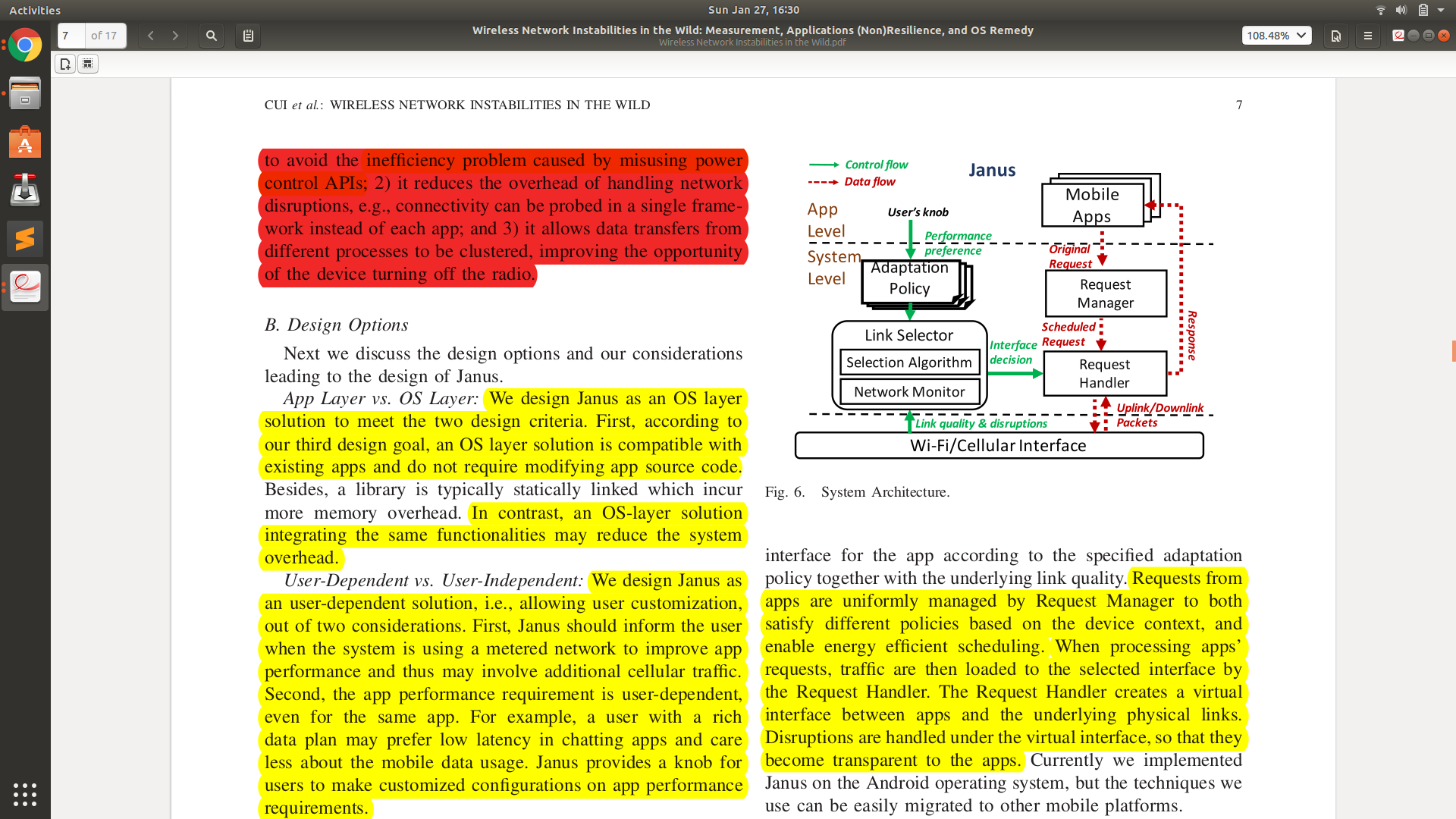
Results:

* A: On 30% of the devices, the user experiences more than 25 WiFi disconnections and reconnections and more than 24 cellular data network disconnections and reconnections per day on average.
* A: On over 10% of the devices, on average the user experiences more than 6.8 and 42 severe signal strength drop per day using WiFi and cellular data, respectively.
* A: On over 10% of the devices, over 3.9 and 3.0 foreground network sessions in a day experience at least one network disconnection in WiFi and cellular data respectively.
* A: On average, the devices spent 47.8% of the time connected to WiFi, 29.3% of the time when WiFi is disconnected but mobile data is connected, and the remaining 23.0% of the time when neither of the interfaces is connected.
* B & C: Most recorded “Not support” means that the app does not actively select an interface and passively uses the interface selected by the OS,
* B & C: “WiFi-only” indicates that the app can be optionally configured to only use WiFi. Overall, few apps deploy their own selection mechanisms and the majority of them are forced to use the interface selected by the Android

OS.

* D: First, we find that many mobile apps, e.g., OperaMini and BBC News, do not handle network disruptions at all.
* D: Second, some apps are only able to tolerate very short disruptions but are not robust against longer disruptions.
* D: Finally, interestingly we observe that different tasks performed by the same app may tolerate disruptions differently.
* D: Moreover, although some apps are able to resume connections from network disruptions, their reaction to connectivity recovery of the underlying network is too slow, which also causes poor user experience.
* F: Janus reduces the page load time (including both data transfers and disruption recovery) by up to 37.18/34.93/30.56% for a 3/5/10s transient network disruption as compared to other approaches
* E: Janus helps to reduce more energy consumption than other three libraries for a 60s disruption (up to 47.3%) than a 3s disruption (up to 30.9%).
* F: Janus reduces the number of stalls by 85.7% as it switches to LTE when the WiFi throughput is insufficient.
* F: Fine-Grained Flow Control
* F: Reducing Traffic Size and Energy Consumption

JANUS Implementation:

* 
* We implement Janus framework as a system level service on LG Nexus 6 running Android 6.0 with a 2.0 GHz octa-core 64-bit CPU and 3GB RAM. Janus has implemented in around 3500 lines of Java code.

Positive Points:

1. Released Source Code.
2. OS level Solution (maintaining versatility)
3. Verified implementation lacking among different naive solution and approaches not just experiment but at the level of comparing source code.

Negative Points:

1. Did not explain the energy consumption and overhead due to introducing JANUS itself.
2. Did not actually explain where and how they introduced JANUS to Android.
3. Did not explain interface interaction and switching at the implementation level.
4. Huge dependency on User Adaption Policy for good results.