Understanding the Networking Performance of Wear OS

Xiao Zhu[1] Yihua Ethan Guo[2] Ashkan Nikravesh[1]
Feng Qian[3] Z. Morley Mao[1]

[1]University of Michigan [2]Uber Techologies Inc. [3]Univeristy of Minnesota

UNIVERSITY OF MICHIGAN
UBER
UNIVERSITY OF MINNESOTA

Wearable Networking Is Important

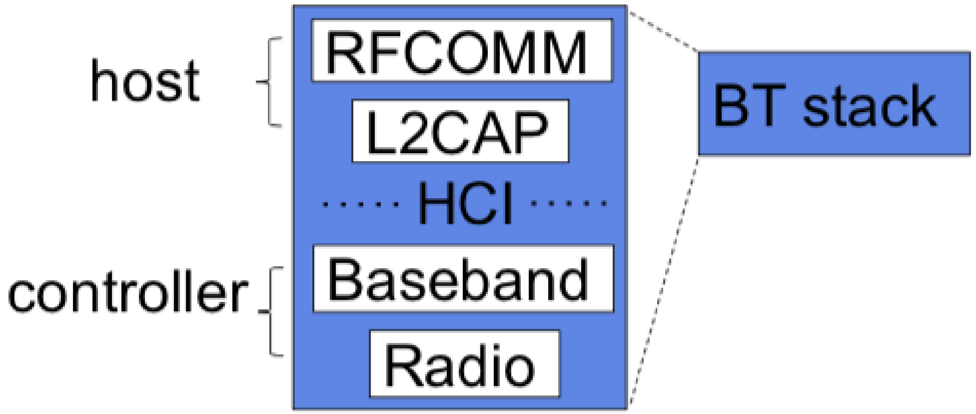
Increased popularity

Third-party apps

Multiple network interfaces

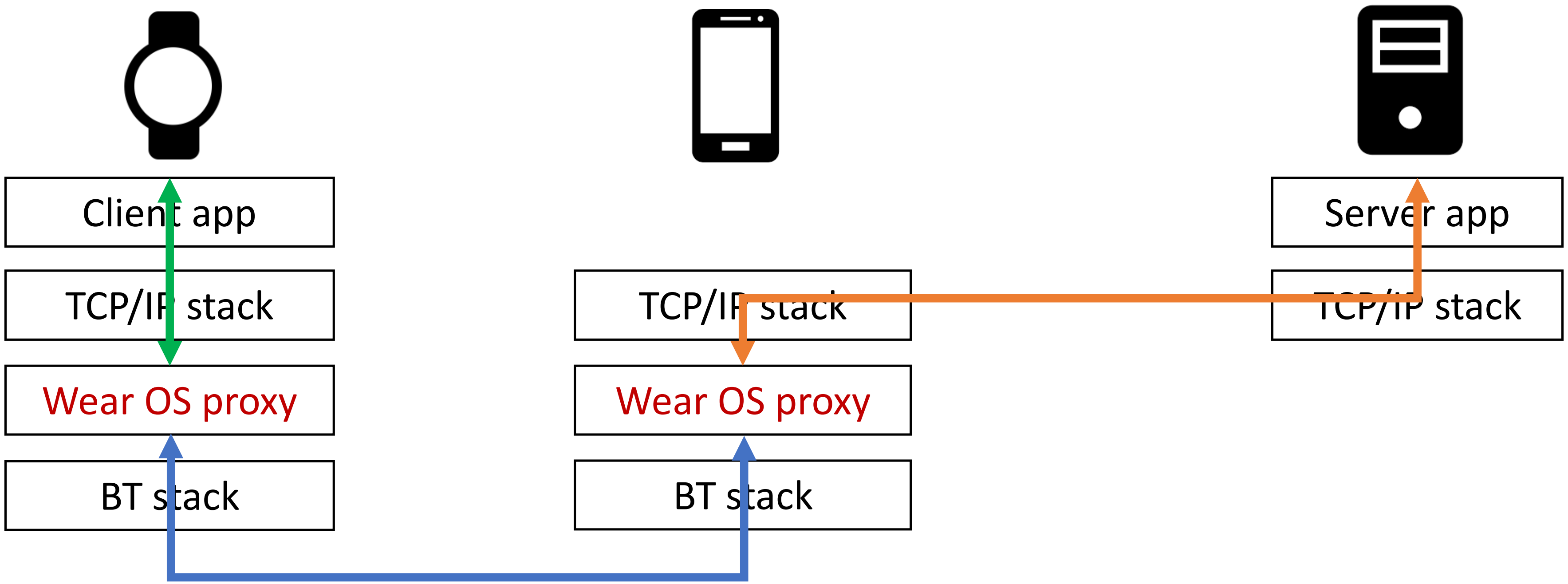
It Is Different from Smartphone Networking

- Bluetooth (BT) communication
  - Different protocol stack and radio state machine



It Is Different from Smartphone Networking

- Bluetooth (BT) communication
  - Different protocol stack and radio state machine
- Smartphone as a “gateway”
  - A pair of proxies in Wear OS



It Is Different from Smartphone Networking

- Bluetooth (BT) communication
  - Different protocol stack and radio state machine
- Smartphone as a “gateway”
  - A pair of proxies in Wear OS
- Network interface switching
  - BT has a much shorter range
  - Vertical handover under mobility

Wearable Networking Stack Is Under-explored

Application [Nirjon MobiSys 15][Shen MobiSys 16]

OS
[Liu APSys 15]
[Liu Mobisys 16]

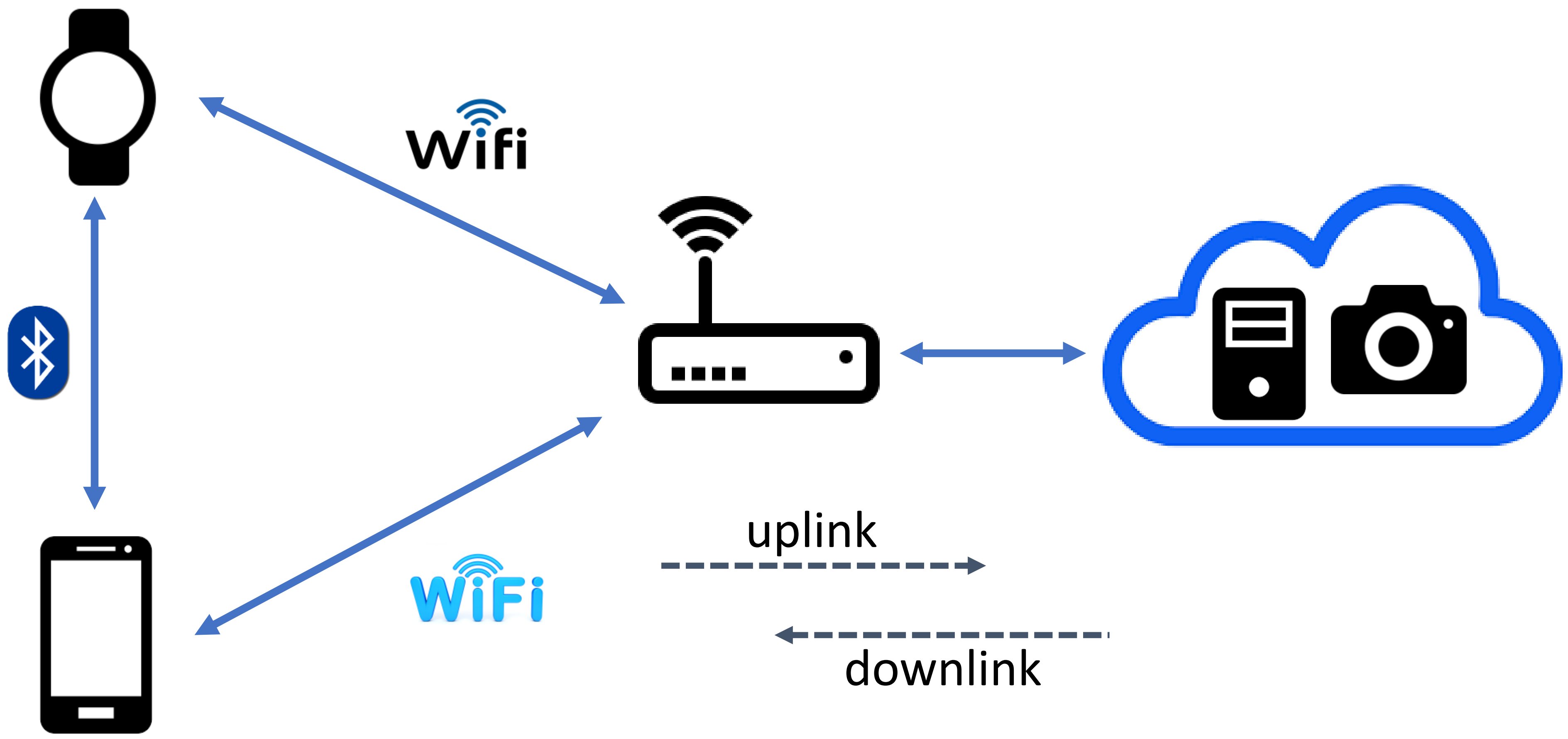
Networking
Traffic characterization: [Kolamunna IMC 18]
Core networking stack: ?

UI
[Chen CHI 14]
[Xu MobiCom 17]

Power [Liu Mobisys 17][Yang ICNP 17]

The networking performance and application QoE on commercial wearables is not well-studied.

Wearable Networking Testbed



The Wearable Network Measurement Toolkit

- Active Measurements
  - Bulk data transfer and constant bitrate traffic
  - Automatic reconnection upon network failure
- Passive Measurements
  - Collect WiFi and BT traces from multiple entities and layers
  - Packet transmission/reception pipeline Instrumentation
  - Signal strength and network states
- Open-source
  - 3K lines of C++, Java, and Python code
  - https://github.com/XiaoShawnZhu/WearMan.

Overview of Measurement Findings

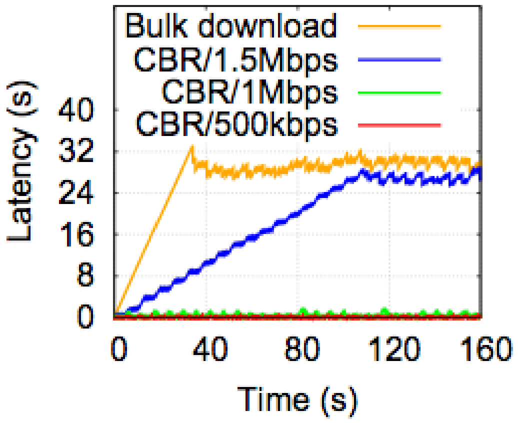
1. Proxy at paired smartphone
   - End-to-end latency is inflated to tens of seconds
   - Phone’s TCP receive buffer causes bufferbloat
2. Network handover
   - Handovers are performed reactively
   - BT-WiFi handovers may take 60+ seconds
3. Bluetooth radio resource management
   - Different state machine models on phone and wearable
   - BT download experiences frequent “blackout” periods
4. Network interface selection
   - Wear OS’s default interface selection policy is often suboptimal
   - Multipath on wearables faces obstacles

Overview of Measurement Findings

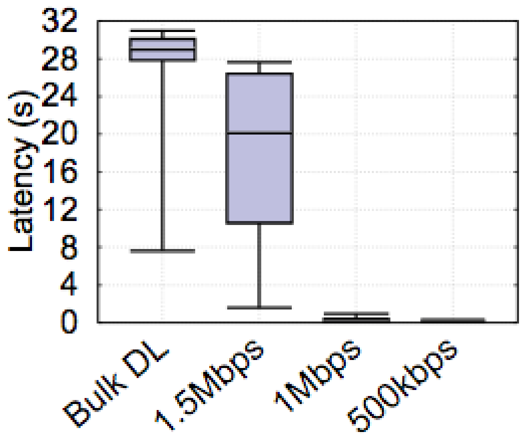
1. Proxy at paired smartphone
   - End-to-end latency is inflated to tens of seconds
   - Phone’s TCP receive buffer causes bufferbloat
2. Network handover
   - Handovers are performed reactively
   - BT-WiFi handovers may take 60+ seconds
3. Bluetooth radio resource management
   - Different state machine models on phone and wearable
   - BT download experiences frequent “blackout” periods
4. Network interface selection
   - Wear OS’s default interface selection policy is often suboptimal
   - Multipath on wearables faces obstacles

Impact of Smartphone Proxying

- End-to-end (E2E) latency characterization
  - Constant bitrate (CBR) traffic and bulk transfer



(a) E2E delay over time.

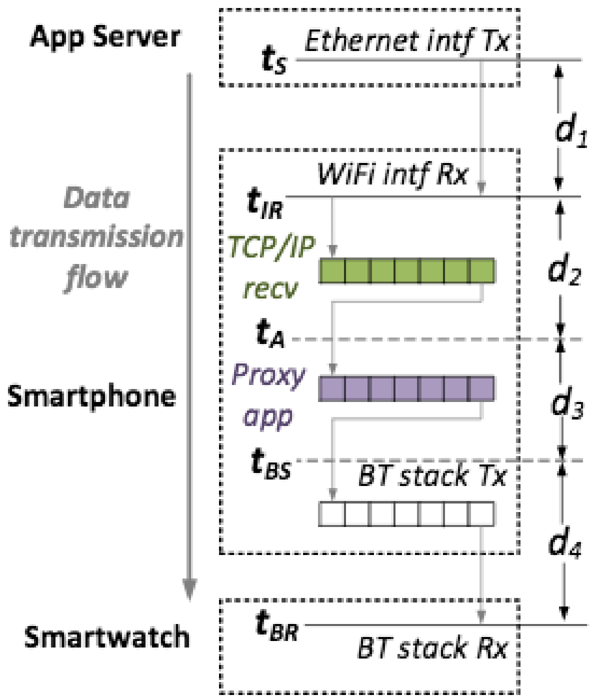


(b) E2E delay distribution.

E2E latency is dramatically inflated to 30+ seconds for high bitrate traffic.

Impact of Smartphone Proxying

- Root cause analysis
  - Breaking down the E2E latency



data is transmitted out

data is received in the phone OS kernel

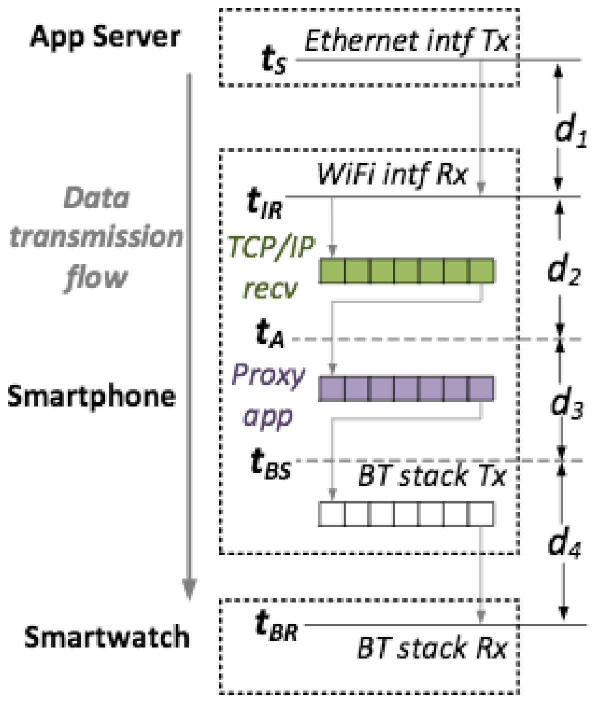
data is copied to the proxy’s userspace

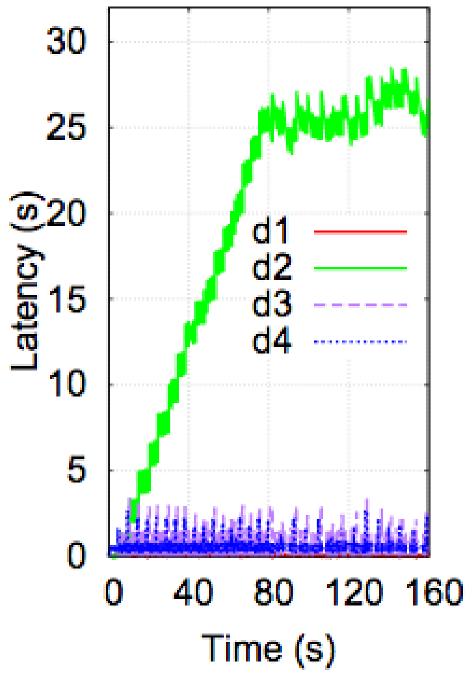
data is sent to the BT stack

data is delivered to the wearable OS

Impact of Smartphone Proxying

- d2 dominates the E2E latency
  - Delay incurred by TCP receive buffer





Impact of Smartphone Proxying

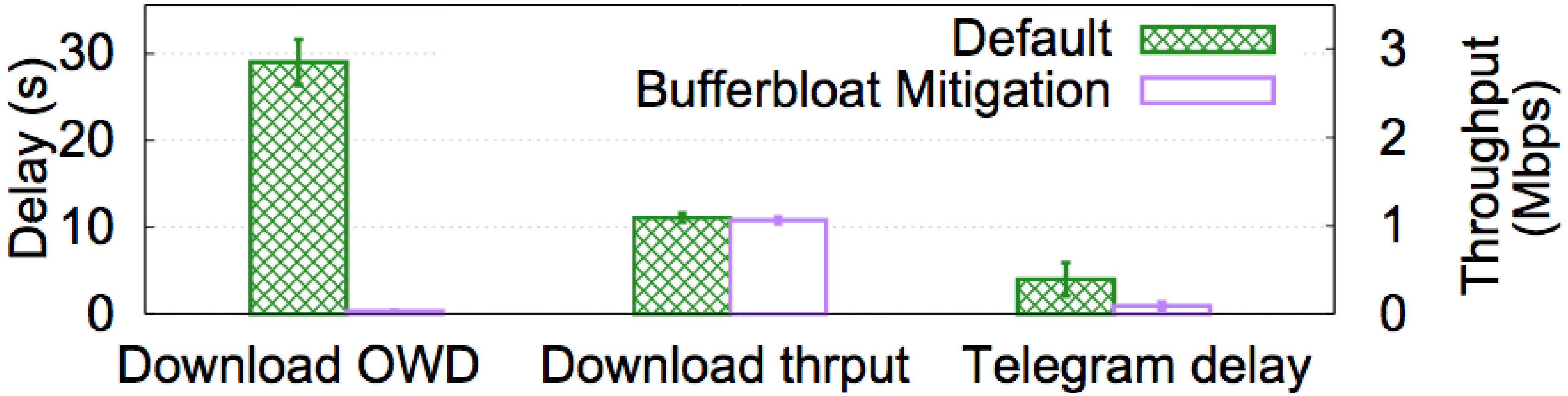
- Phone's TCP receive buffer size impact on E2E latency

	Nexus 5X	SGS5	Nexus 5
tcp_rmem_max	8,291,456	4,525,824	2,097,152
rmem_max	8,388,608	2,097,152	2,097,152
d_2: TCP/IP recv (s)	26.1 ~ 28.6	4.0 ~ 5.5	4.1 ~ 5.7
Total E2E OWD (s)	27.9 ~ 30.1	5.7 ~ 6.7	5.9 ~ 7.0

Smaller TCP receiver buffer size reduces the E2E latency, but setting it to be too small may throttle the server-phone connection throughput.

Impact of Smartphone Proxying

- Mitigating the bufferbloat
  - Examine the queue length (Q) on the phone and phone-wearable bandwidth (BW)
  - Throttle the server-phone connection when $\frac{Q}{BW}$ becomes high



Dynamic server-phone flow control that considers the phone-wearable network condition reduces the E2E delay.

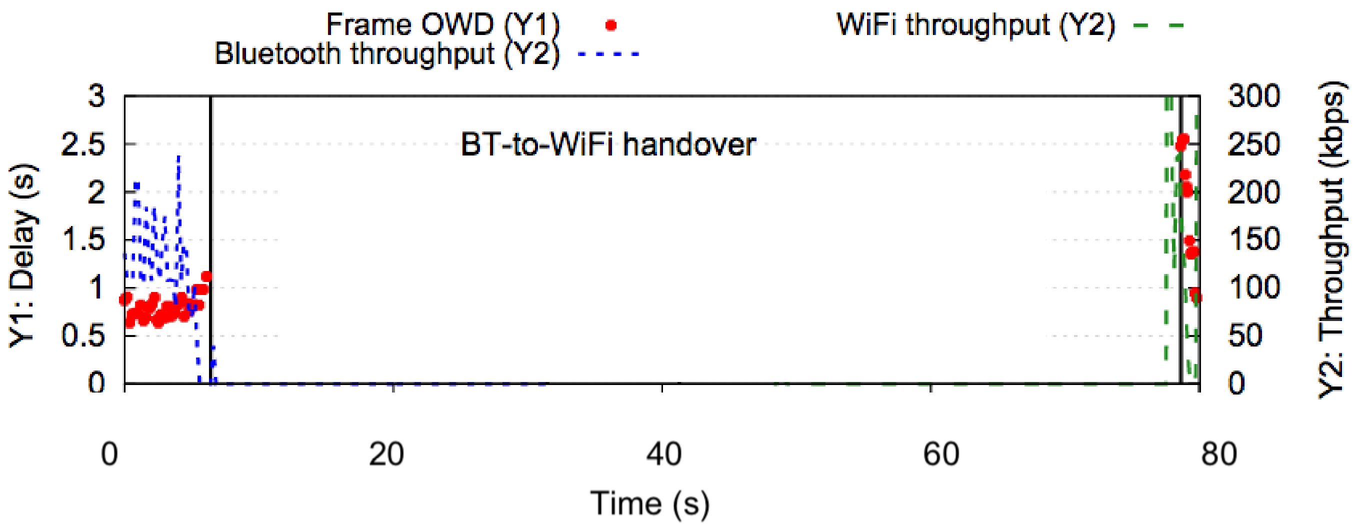
Overview of Measurement Findings

1. Proxy at paired smartphone
    - End-to-end latency is inflated to tens of seconds
    - Phone’s TCP receive buffer causes bufferbloat
2. Network handover
    - Handovers are performed reactively
    - BT-WiFi handovers may take 60+ seconds
3. Bluetooth radio resource management
    - Different state machine models on phone and wearable
    - BT download experiences frequent “blackout” periods
4. Network interface selection
    - Wear OS’s default interface selection policy is often suboptimal
    - Multipath on wearables faces obstacles

BT-WiFi Handover Performance

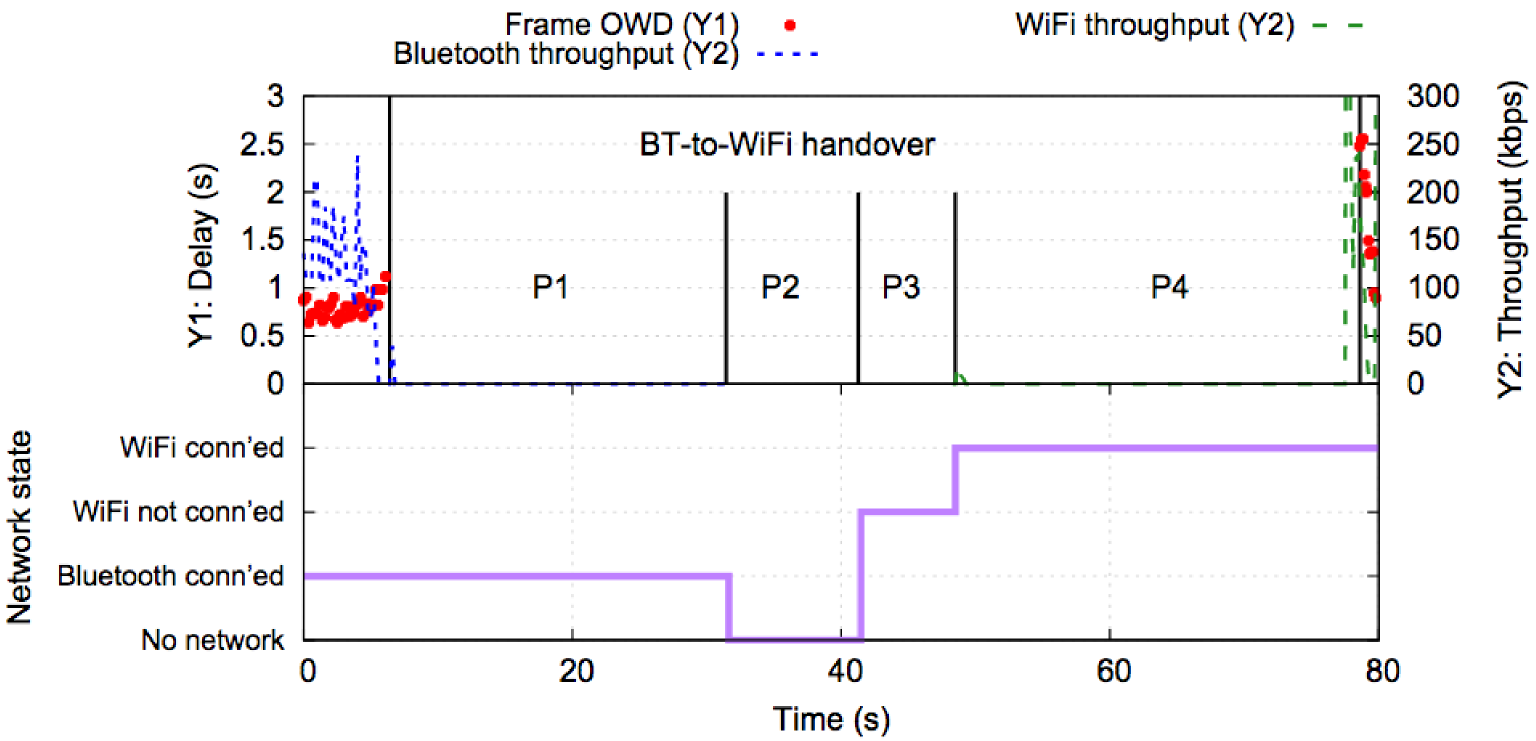
- Monitoring the network state
  - *ConnectivityManager* in Wear OS
    - *Avaliable* or not: whether the network interface is up
    - *Connected* or not: whether the interface provides actual network connectivity
- Experiment setup
  - Both BT and WiFi are enabled
  - Real-time streaming traffic
    - tinyCam app: stream real-time videos captured from an IP camera
  - A user wearing a smartwatch moves away from the paired smartphone

BT-WiFi Handover Performance



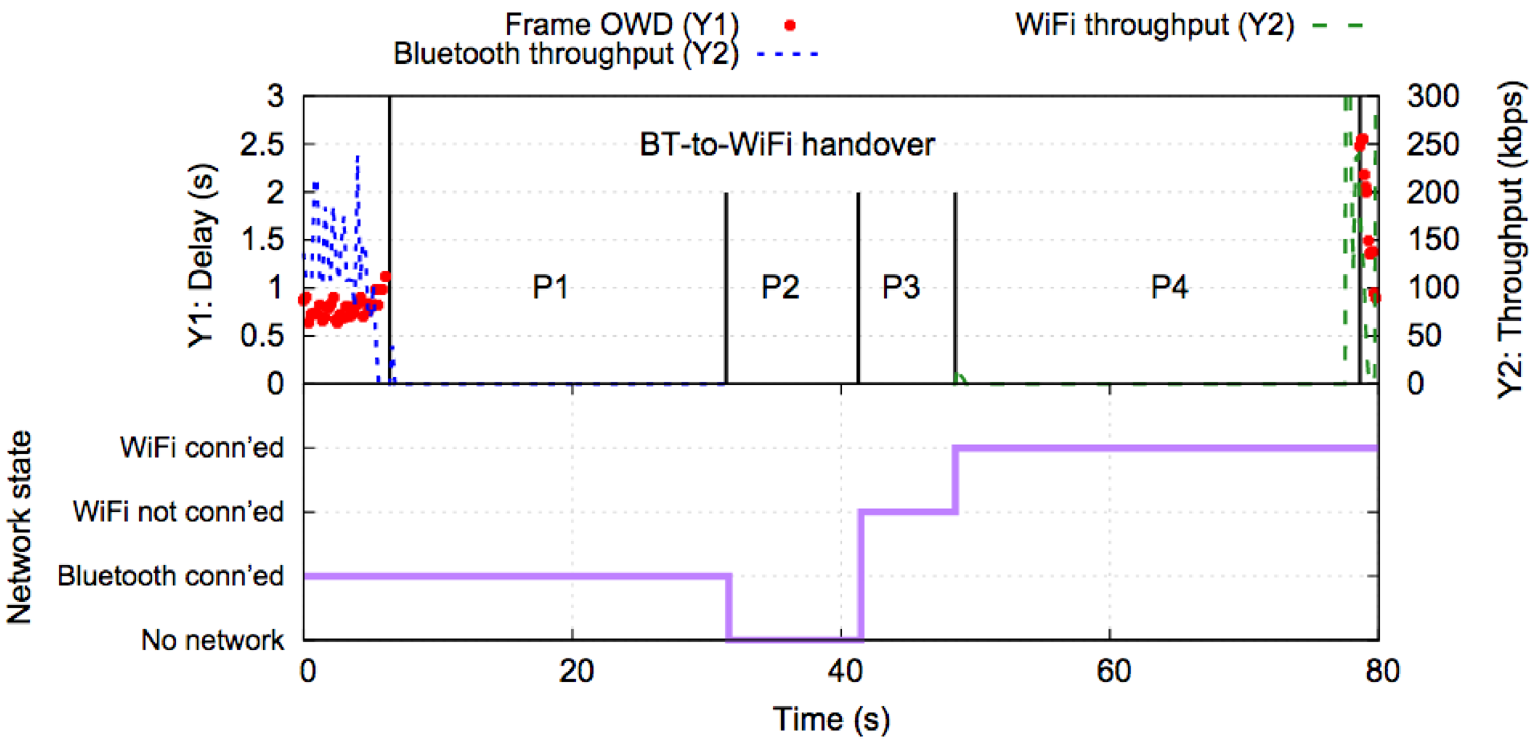
- Throughput and frame delay severely degrade during the handover
- 60+ seconds of interruption time when no video data is received

Root Cause Analysis: Delay Breakdown



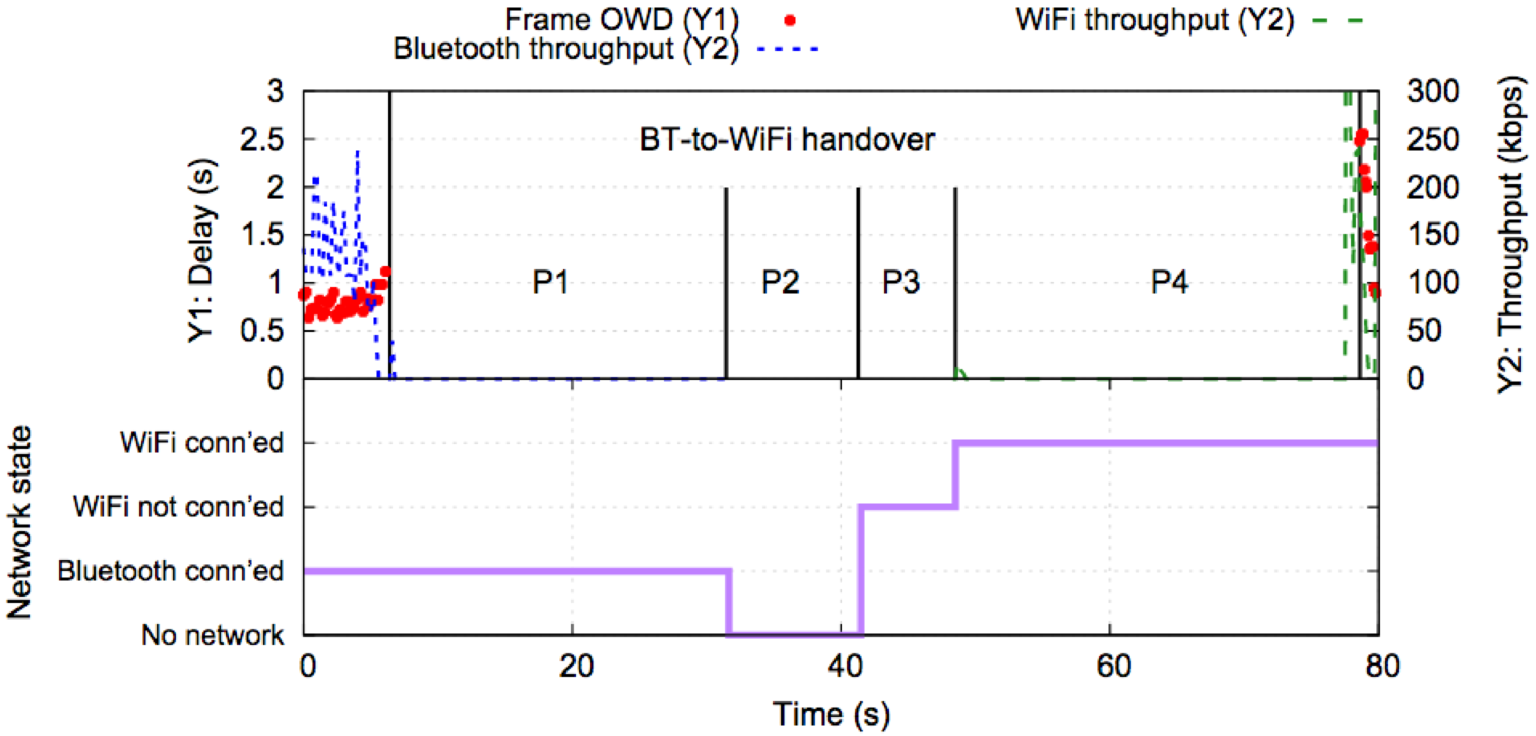
- P1: BT is connected but data cannot be actually transmitted
- P2: no network available
- P3: WiFi available (interface up) but not connected
- P4: WiFi connected but no application data transmission

Root Cause Analysis: Delay from the Wear OS (P1, P2, and P3)



Reactive in nature: Only after BT connection gets lost completely (P1), the Wear OS turn on (P2) and then connect to (P3) WiFi.

Root Cause Analysis: Delay Incurred by the Wearable App (P4)



Insufficient protocol support for applications: wearable apps need to implement their own data migration logic.

Root Cause Analysis: Delay Incurred by the Wearable App (P4)

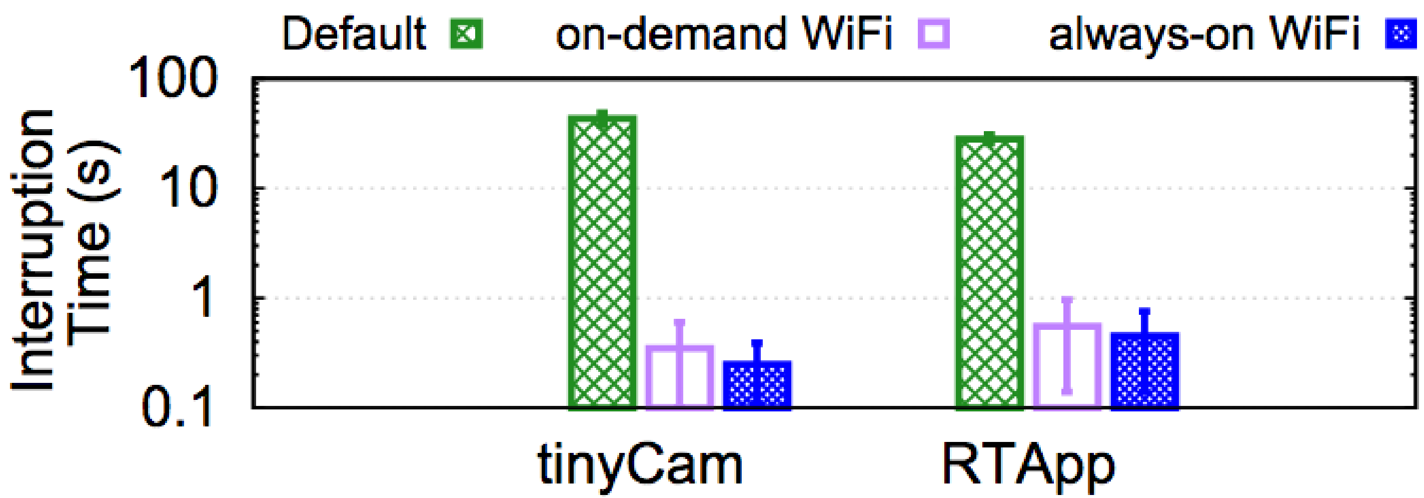
- P4: 33.3s (tinyCam) v.s. 5.6s (RTApp)
  - RTApp: downloading a 3KB data chunk every 160ms, establish new connection once a new network interface is connected after a handover.
- Overall handover interruption time

	LG Urbane	LG Urbane 2nd	HUAWEI Watch
tinyCam	43.1 ± 5.7 s	52.9 ± 8.2 s	70.2 ± 9.7 s
RTApp	28.3 ± 2.6 s	14.3 ± 1.3 s	38.6 ± 5.3 s

Improved application data migration logic (in RTApp) reduces P4 as well as the overall interruption time.

Reducing the Handover Delay

- Proactively performing a handover to WiFi when BT quality degrades
  - Variant 1: establish WiFi when performing handovers (on-demand WiFi)
  - Variant 2: pre-established WiFi (always-on WiFi)



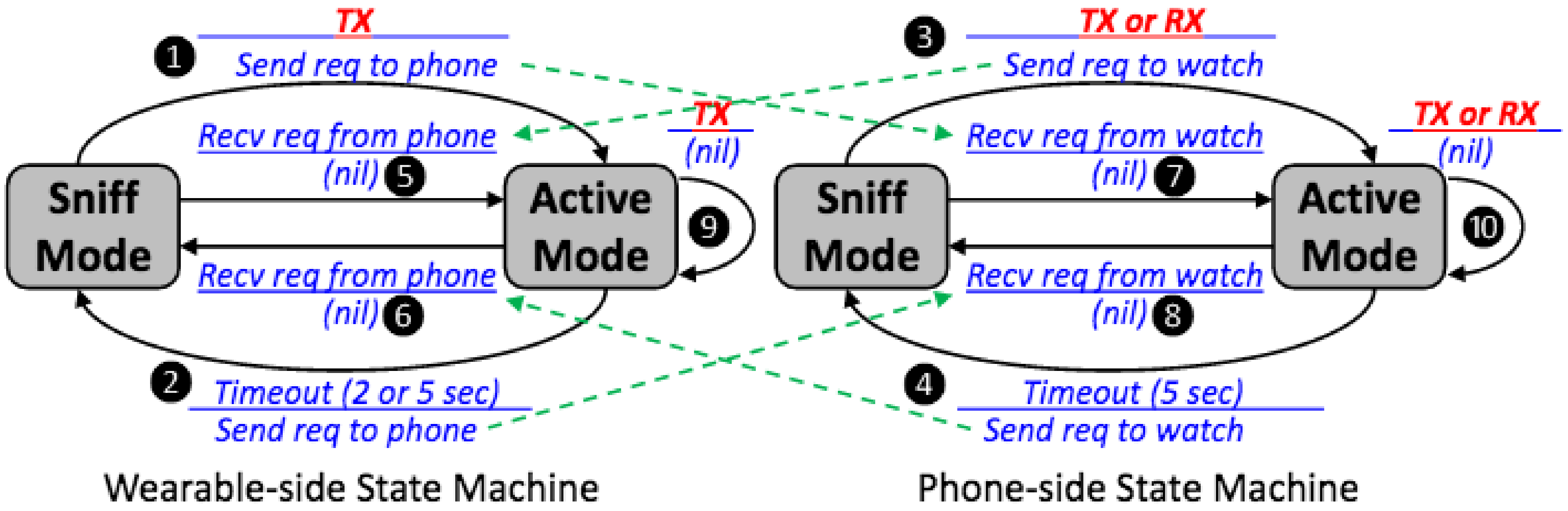
Summary

- First in-depth study on the networking performance of Wear OS.
- Developed a toolkit for wearable networking measurement and analysis.
- Identified performance issues regarding key aspects of wearable networking.
- Analyzed the root causes and proposed practical solutions.

Thank you!

Thank you!

BT State Machines on Wearable and Phone



QoE-energy Tradeoffs of Different Networks