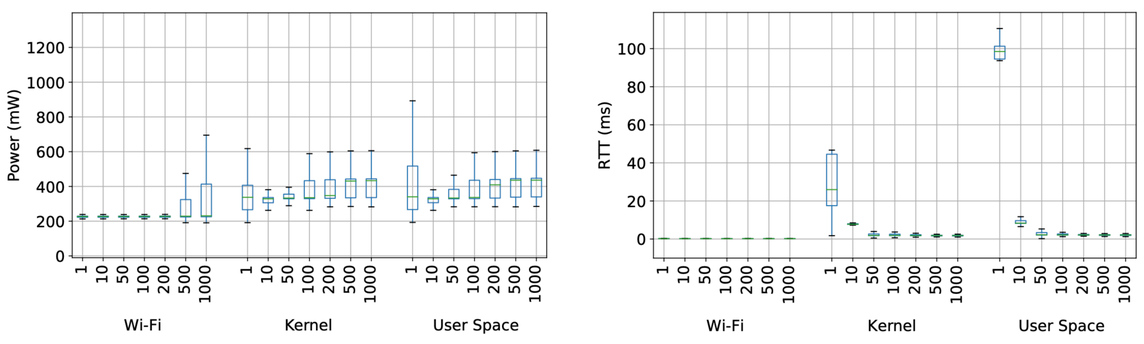
ReactiFi: Reactive Programming of Wi-Fi Firmware on Mobile Devices

Artur Sterz, Matthias Eichholz, Ragnar Mogk, Lars Baumgärtner, Pablo Graubner, Matthias Hollick, Mira Mezini and Bernd Freisleben

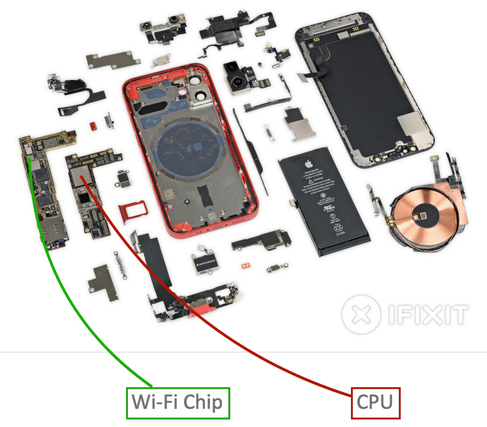
Why Programming Wi-Fi?

Many possible applications (e.g., switch Wi-Fi channel, switch from AP to P2P mode) for improving user's network experience like reduce power consumption or boost latency significantly when implemented in the Wi-Fi firmware and not in kernel or user-space. But as of today this is way too difficult to achieve, because developers have to be platform- and Wi-Fi experts.



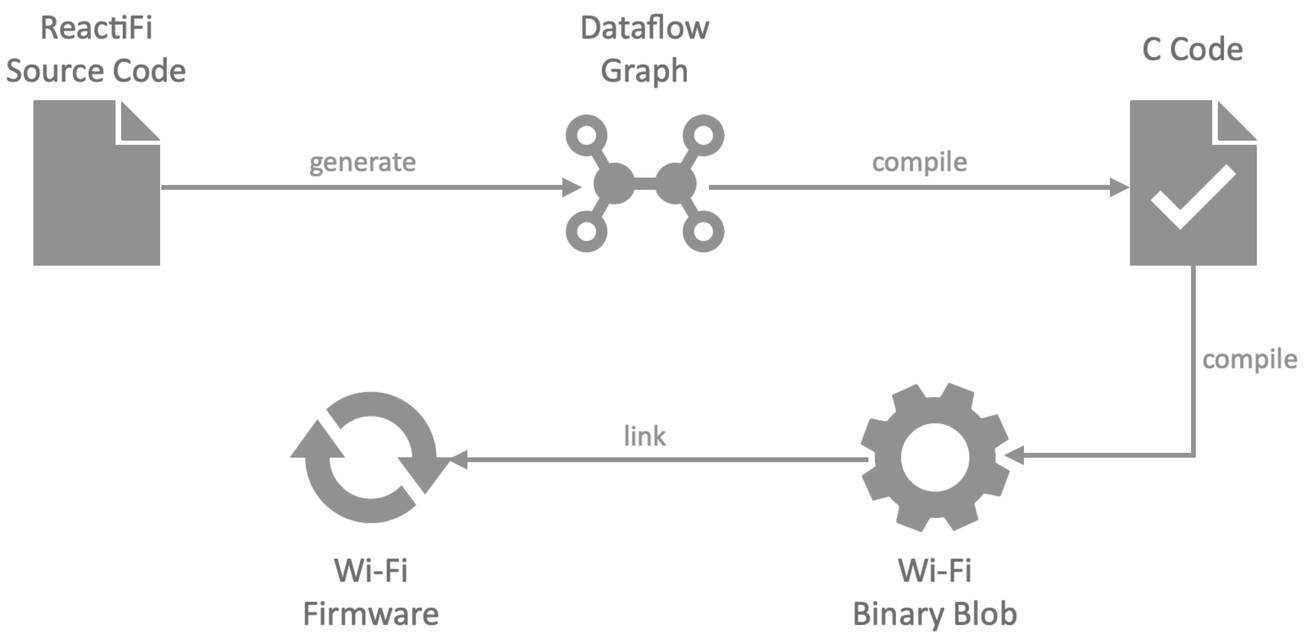
Design

Wi-Fi chips are constrained co-processors with limited memory and no concurrency or parallelization. Wi-Fi chips also have strengths like little power consumption. ReactiFi makes strengths easy to use and handles weaknesses transparently. Also, basic Wi-Fi functionality is preserved, e.g., acknowledging every frame. The language should not mess with mandatory functionality. To achieve these goals a domain specific, data-flow driven language is necessary.



Language and Compiler

```
1 Source(Timer(10ms))
2   .fold({ 0 })((channel, time) => { (channel % 20) + 1 })
3   .observe(SwitchChannel)
4
5 val addrs = Source(Monitor)
6   .filter(frame => { frame.type == MANAGEMENT })
7   .map(frame => { frame.src })
8
9 val timer = Source(Timer(200ms))
10
11 val count = fold({ hashset_new() })(
12       timer -> (acc, time) => { hashset_new() },
13       addrs -> (acc, addr) => { hashset_add(acc, addr) })
14   .map(p => { sizeof(p) })
15
16 timer.snapshot(count.change(0))
17   .map(tuple => { tuple.snd })
18   .observe(SendToOS)
```

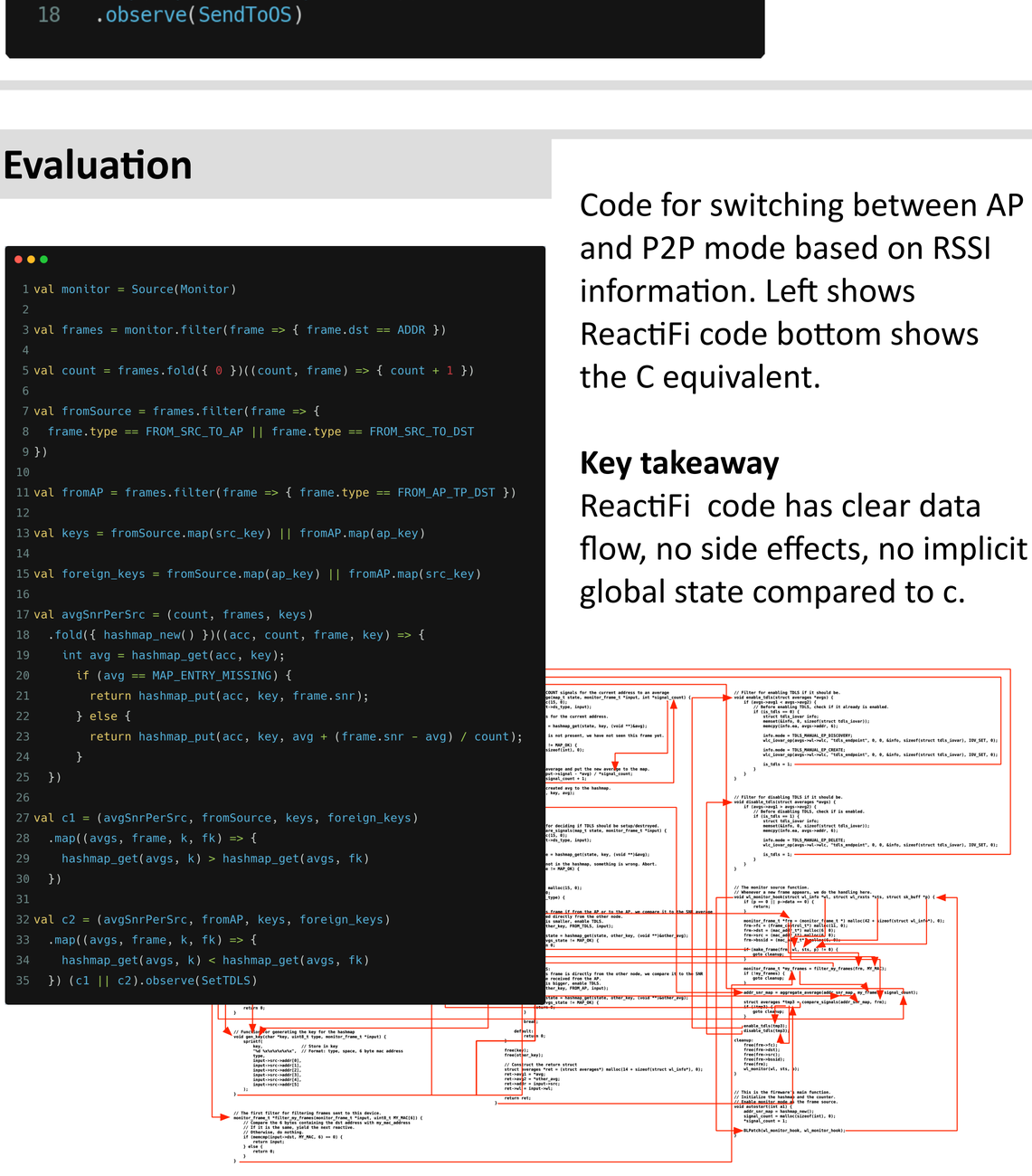


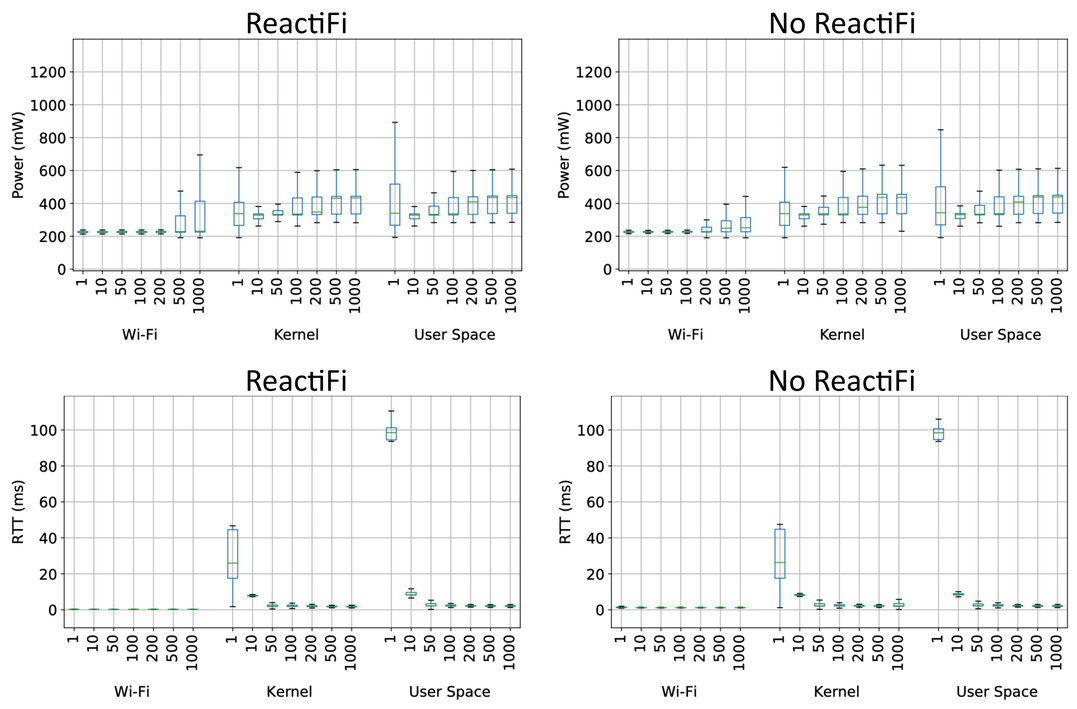
Evaluation

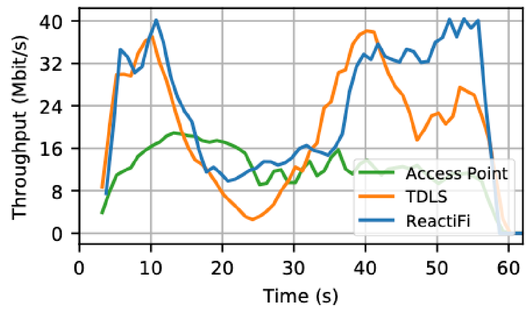
```
1 val monitor = Source(Monitor)
2
3 val frames = monitor.filter(frame => { frame.dst == ADDR })
4
5 val count = frames.fold({ 0 })((count, frame) => { count + 1 })
6
7 val fromSource = frames.filter(frame => {
8   frame.type == FROM_SRC_TO_AP || frame.type == FROM_SRC_TO_DST
9 })
10
11 val fromAP = frames.filter(frame => { frame.type == FROM_AP_TP_DST })
12
13 val keys = fromSource.map(src_key) || fromAP.map(ap_key)
14
15 val foreign_keys = fromSource.map(ap_key) || fromAP.map(src_key)
16
17 val avgSnrPerSrc = (count, frames, keys)
18   .fold({ hashmap_new() })((acc, count, frame, key) => {
19     int avg = hashmap_get(acc, key);
20       if (avg == MAP_ENTRY_MISSING) {
21         return hashmap_put(acc, key, frame.snr);
22       } else {
23         return hashmap_put(acc, key, avg + (frame.snr - avg) / count);
24       }
25   })
26
27 val c1 = (avgSnrPerSrc, fromSource, keys, foreign_keys)
28   .map((avgs, frame, k, fk) => {
29     hashmap_get(avgs, k) > hashmap_get(avgs, fk)
30   })
31
32 val c2 = (avgSnrPerSrc, fromAP, keys, foreign_keys)
33   .map((avgs, frame, k, fk) => {
34     hashmap_get(avgs, k) < hashmap_get(avgs, fk)
35   }) (c1 || c2).observe(SetTDLS)
```

Code for switching between AP and P2P mode based on RSSI information. Left shows ReactiFi code bottom shows the C equivalent.

Key takeaway
ReactiFi code has clear data flow, no side effects, no implicit global state compared to c.







Key takeaway
Above: ReactiFi does not add any overhead in terms of power consumption or latency.

Left: Case-Study for changing from AP to P2P mode uses best available mode so that always highest possible throughput is achieved.