Virtual Wires: Rethinking WiFi networks

Yudong Yang[1], Yuming Jiang[2], Vishal Misra[1], Dan Rubenstein[1]
[1]*Dept. of Computer Science, Columbia University, United States*
[2]*Dept. of Information Security and Communication Technology, Norwegian University of Science and Technology, Norway*
[1]{yyd, misra, danr}@columbia.edu, [2]jiang@ntnu.no

***Abstract*—WiFi is the dominant means for home Internet access, yet is frequently a performance bottleneck. Without reliable, satisfactory performance at the last hop, end-to-end quality of service (QoS) efforts will fail. Three major reasons for WiFi bottlenecking performance are its: 1) inherent wireless channel characteristics, 2) approach to access control of the shared broadcast channel, and 3) impact on transport layer protocols, such as TCP, that operate end-to-end, and over-react to the loss or delay caused by the single WiFi link.**

In this paper, we leverage the philosophy of centralization in modern networking and present our cross layer design to address the problem. Specifically, we introduce centralized control at the point of entry/egress into the WiFi network. Based on network conditions measured from buffer sizes, airtime and throughput, flows are scheduled to the optimal utility. Unlike most existing WiFi QoS approaches, *our design only relies on transparent modifications, requiring no changes to the network (including link layer) protocols, applications, or user intervention*. Through extensive experimental investigation, we show that our design significantly enhances the reliability and predictability of WiFi performance, providing a "virtual wire"-like link to the targeted application.

I. Introduction

Protocol layering, decentralization and end-to-end congestion control have been the cornerstones of the Internet architecture, enabling continual growth of backbone infrastructure and rapid development of novel applications. However, the traditional end-to-end, TCP-based approach that determines an individual flow's bandwidth allocation at bottleneck links assumes that competing flows have the same priority and will benefit equally from an equal share of bandwidth. Such assumptions are no longer realistic, failing to effectively capture the bandwidth and delay needs of the corresponding applications [1].

In this paper we apply the philosophy of centralization at the end of the network spectrum: the modern home network, where access is dominated by WiFi, and where a diverse and ever-growing mix of device types and applications must be supported. We highlight the limitations of the current protocol layering and distributed control schemes in the context of WiFi, and present, implement and experiment with a cross layer design that uses a similar philosophy of centralization. Our design consists of three components:

1) *A flow classifier* that automatically classifies flows into utility-based application categories, which feeds into:
2) *A novel optimizer* that maximizes the sum of the application utilities, accounting for the mix of application types, device types, and wireless link rates;
3) *A work conserving scheduler* which, sitting at the center of the design, takes the bandwidth allocated by the optimizer as input and manages airtime use by flows to meet the optimizer's allocation.

Together, the components work to find and deliver the most application-appropriate QoS to the flows under the available bandwidth constraints. This is accomplished with *complete transparency*, requiring no user intervention. For this reason, we call our system **Virtual Wires** (or **VW** for short), as it brings switched Ethernet-like performance characteristics to WiFi. As our experiments show, in a wide variety of scenarios, VW enables automatic provisioning of a "virtual wire"-link for each application that supplies it with a QoS-appropriate amount of bandwidth.

In particular, while traditional WiFi bandwidth management mechanisms are implemented within the link layer, we implement ours within the transport layer via a transparent splitting proxy, hence applications benefit without needing any modification. While the use of session splitting is not new [2], [3], our use of splitting is novel in that it not only enhances a flow's throughput and delay but also permits better coordination of allocations among competing flows, which improves WiFi performance given its limitations due to its *physical layer characteristics* and *link layer shared channel access control*. In addition, splitting compensates for transport layer artifacts caused by *end-to-end TCP congestion control*, which might influence bandwidth and delay allocated to sessions, i.e., VW overrides the rates assigned by TCP's congestion control mechanism that is oblivious to application requirements.

Our work shows that reliability and predictability of WiFi can be enhanced by centralized control and transparent modifications at the point of entry/egress into the WiFi network, which can also be viewed as complementary to any efforts in the link or physical layers to improve WiFi performance reliability. Our contributions can be summarized as follows:

- We present a 3-component architecture which, from an application's standpoint, vastly improves the utilization of sessions traversing a WiFi link.
- We demonstrate a novel benefit of session splitting, using it

This work was funded in part by the NSF through awards CNS-1717867 and CNS-1618911. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.
978-1-7281-1434-7/19/$31.00 ©2019 IEEE

as a means to both a) control the allocation across different flows at the WiFi bottleneck, and b) apply appropriate congestion control to different connectivity regions based on their unique characteristics (i.e., backbone vs WiFi link)
- We present extensive experimental validation of our system, using testbeds that include commodity home WiFi routers built by Google and Linksys under real Internet applications.
- Experimental results show that the performance reliability is improved in terms of bandwidth, delay and system utility. In a wide variety of settings, Virtual Wires is able to meet higher QoS for tagged applications in the presence of greedy, background flows, providing much higher and more stable throughput rates for flows whose applications depend on stable throughputs, thereby enabling higher quality (e.g., 4K) streaming. Delays are reduced by 59.9%-89.5% depending on the congestion level. By utilizing differing TCP variants on opposite sides of the split, our system improves link utilization by 30.7% for long distance WiFi.

The remainder of the paper first introduces the rationale of our design, and then presents the overall design architecture. In addition, we present experimental results. Finally, we discuss related work, and conclude.

II. Design Rationale

In this section, we present the rationale for our design, going over both the limitations of WiFi as well as prior art that does not fully address these limitations.

A. WiFi Limitations

The unpredictability in WiFi performance is due to many wireless channel factors such as multipath, fading, and interference. However, WiFi suffers from additional challenges that are unique from other wireless technologies:

Poor isolation: A shared WiFi radio channel experiences interference due to activity of other devices in the WiFi network. Such interference is known to cause large variations in WiFi network bandwidth and delays [4].

Rate anomaly: In a WiFi network, the link rate is adjusted per device. The packet-by-packet arbitration of WiFi scheduling induces a phenomenon termed *rate anomaly* [5], where low-rate users[1], who require longer times to transmit their frames, consume a disproportionate share of the airtime needed by other connected devices.

Air time allocation policy: To address the above challenges, some commercial implementations apply a technique called *Air Time Fairness*, where the air time is simply partitioned equally among the competing devices. However, this mechanism lacks the expressivity and flexibility to deal with differing QoS requirements across different applications. For instance, flows that can afford being slowed down should yield airtime to e.g., a video streaming app that needs additional airtime to meet its video bitrate.

[1]whose rates are often low due to either the use of legacy WiFi technologies or weak (low SNR) signals.

B. Existing Approaches to QoS

There exist prior attempts to develop non-obtrusive means to provide QoS within a WiFi environment. WiFi Multimedia (WMM) [6], introduced by the WiFi Alliance, classifies all traffic into four classes via the setting of the differentiated services code point (DSCP) bits in the IP header. Each class is mapped to a queue where the four queues are served with differing priorities. However, predefined priorities are insufficient at meeting the optimal QoS of the whole system. For example, a video streaming application won't have higher QoS when its goodput becomes greater than its video bitrate. In this case, assigning a higher priority to the video will offer no improvement to video quality, but may unnecessarily reduce throughputs of other (lower priority) applications. Moreover, DSCP bits are often altered by network operators along the Internet path before packets reach the WiFi router [7], and hence may not even be tuned specifically for the needs of the WiFi link. We will show that in contrast, VW requires no setting of header bits, as we do automatic classification at the WiFi access point. In addition, unlike WMM, we tune *at the wiFi boundary* specifically to the needs of *current* applications.

An alternate approach to achieving QoS at the WiFi boundary is via implementation of an *artificial bottleneck*. For instance, the Linksys, D-Link, Netgear routers and firmware OpenWRT (with qos-scripts module) and DD-WRT we tested use this approach where the router imposes an aggregate throughput that is less than the available bandwidth of the interface. Since available bandwidth in WiFi is a function of the set of devices and their current physical rates which continually fluctuate, an artificial bottleneck will at times waste available bandwidth when actual bandwidth is greater than the imposed artificial bottleneck, and at other times be insufficient because actual bandwidth falls below the imposed artificial bottleneck. In contrast, our scheduler is *work conserving*, and makes use of all available bandwidth while also directing that available bandwidth to best meet the QoS needs for existing application (requiring *no user intervention*).

C. End-to-End Effects

WiFi's unreliability in performance is even more pronounced end-to-end, as WiFi is increasingly the only shared broadcast link on an end-to-end path, whereas almost everything else is switched. The inconsistent and unpredictable behavior of the one-hop WiFi link not only causes challenges for transmissions crossing that link, but in fact has a substantial effect on transport layer protocols that operate end-to-end. For example, the recently developed BBR TCP congestion control algorithm [8], designed to reduce delay and increase throughput, is aimed at *wired* links. WiFi links not only hide losses from the transport layer to a great extent by link layer retransmissions, but the varying physical rates on a WiFi link can cause BBR to underperform. We present a detailed analysis of this effect in Section III-B and experimental validation in Section IV.

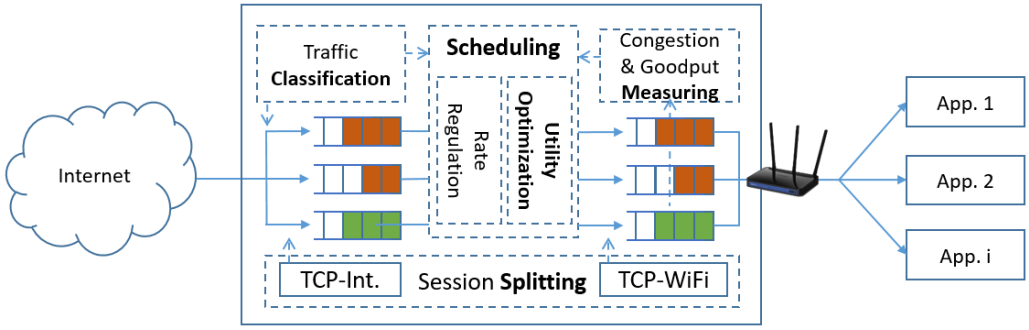


Fig. 1. System overview. Solid arrows are data flows (uplinks are omitted), doted arrows are control flow

III. Architecture

In this section, a design overview is first presented, followed by introduction to the detailed ideas of the design.

A. Design Overview

Our system resides within a WiFi router, as illustrated by Fig.1. For simplicity of presentation, we only show the downlink portion of the design (where the WiFi link is the final hop). It contains a similar mechanism to handle the uplink portion. The modules with solid borders already exist in traditional Linux implementations, while the modules with dashed line borders are our design. Solid arrow lines indicate the flow of network traffic, and the dashed arrow lines indicate the relations between the various modules.

The following provides a summary of our design ideas, which are further explained in the subsequent subsections.

(i) Session Splitting. Virtual Wires uses a proxy approach to split network sessions into two parts, and for each part of a split session, we select TCP implementation that is more suitable for the characteristics of that part of the session.

(ii) Traffic Classification. Each network session is mapped into application classes, where a QoS profile can be assigned per application class. The QoS profile can also be made dependent on the capabilities of the flow receiver, e.g. a 4K TV vs a smartphone. Each application class session maps to its own queue that can be used to schedule its packets.

(iii) Utility Optimization. Based on the QoS profile, a utility function is mapped to each application. The optimizer solves the utility maximization problem, and assigns the optimal *airtime allocation* to the scheduler.

(iv) Scheduling. When the network is not congested, the packets are forwarded as soon as they arrive. Otherwise, our system schedules packets to meet the airtime assigned by the optimizer.

(v) Feedback. Virtual Wires tracks the achieved transport-layer rates of the split portion of the connection traversing the WiFi network, and feeds this information back to the optimizer and scheduler to select traffic for forwarding that ensures the optimal use of WiFi airtime when congested.

Together, these design ideas result in a transparent proxy solution that requires no changes to existing applications or end-systems.

B. Session Splitting

The system uses a transparent proxy as a means to intercept and split network sessions. The transparent proxy is similar to Network Address Translation (NAT). Instead of translation, in our system, the proxy splits the network sessions and forwards data between the WiFi portion and Internet portion of the split.

Specifically, a connection is split into two parts: a client's session that ends at our proxy, and a remote session that connects the proxy with the remote server. The proxy sits in the middle and listens on both sessions. Whenever a data packet arrives on one side, the proxy receives and forwards the data packet to the other side. The proxy is similar in transparency to a NAT, and does not require any setup on the user's smart phone or laptop.

For TCP sessions, the system obtains two benefits by splitting: One is related to throughput gains due to separate and shorter feedback loops running at two ends of the split, and this effect is well known in the TCP-splitting literature [3].

Another gain with session splitting is that it allows us to select different congestion control algorithms on the two sides of the split to match with the diverse characteristics of the respective paths they traverse, namely a likely wired Internet path and a WiFi link. To understand its necessity, consider the BBR case as an example: BBR [8] is a recently proposed TCP congestion control algorithm being deployed on Google's servers. However, our tests in Section IV show that BBR throughput is significantly suboptimal across a WiFi link. In part, this is because BBR underestimates the congestion window when the last hop WiFi link is bottlenecked. BBR sets congestion window to $B \cdot r$, where B is the bandwidth of the bottleneck and r is the minimum end-to-end RTT. With WiFi link in the path, unlike wired link, the average RTT R is greater than r, not only because of queuing but also factors like channel waiting, packets aggregation, retransmissions on the wireless link. As a consequence, the achieved throughput becomes $B\frac{r}{R}$ that can be low when the propagation time is small relative to the average delay.

Our system maximally make use of the full bandwidth while also take advantages of BBR, by using BBR on the wired link but switching the congestion control on the WiFi link.

C. Automatic Traffic Classification

In our system, network connections are grouped into application classes, with each class maps to a utility function.

To classify the traffic, we use the random forest algorithm. The advantage of using a machine learning-based algorithm is that user or application intervention is not required, i.e., users need not select or specify applications classes, and applications require no modification. Since the topic of traffic classification has been well-studied [9], [10] and the classifier itself is not the focus of this paper, we use the same statistic feature as [9] to implement the classifier. In addition, we separate connections by device's IP address and use MAC address to identify the type of devices [11] which is added in the feature. To enable run-time classification, we update statistics and predict every 2 seconds.

D. Utility Optimization

In the WiFi system that has N concurrent applications, and each application has utility function $u_1(b_1), \cdots, u_N(b_N)$, where $u_i(b_i)$ is the utility generated by application i when its goodput is b_i. In our system, we use $u_i(b_i)$ to characterize the QoS requirements for applications. For example, a YouTube video application can map to a simple step function that increases at where b_i equals to the bitrates of each quality levels (720p, 1080p). For web, or other bandwidth-greedy applications, we can use $w_i \log(b_i)$ to achieve proportional fairness weighted by w_i [12].

The goal is to maximize the summation of all the utility functions within the WiFi system. The optimization problem is

$$\max \sum_{i=1}^{N} u_i(b_i)$$

$$\text{s.t. } \; b_i = a_i L_i, \; 0 \le a_i, \; \text{and } \sum_{i=1}^{N} a_i \le U$$

where a_i is the airtime fraction assigned to application i, and L_i is the physical link rate of the device. $0 < U \le 1$ is the measured WiFi network utilization, the total fraction of airtime that can be used to transfer data. Depending on the radio interference, fading and transmission overhead, U is estimated by $U = \sum_{i=1}^{N} \frac{\hat{b}_i}{L_i}$, where $\hat{b}_i$ is the measured goodput of application i.

The optimizer is triggered whenever input U, L_i or $u_i(\cdot)$ is changed. If all the utility functions in the system are concave functions, e.g. $log(\cdot)$, then an L-BFGS-B algorithm from scipy package [13] is used. If there exist non-concave functions in utilites, e.g. step function for videos, then we divide $[0, U]$ into 20 grids, and evaluate on all feasible combinations of grids. Then L-BFGS-B is applied on the best grid point to find the local maximum near the grid. To speed up the process, we add the condition $\sum_{i=1}^{N} a_i = U$ when the utility functions are non-decreasing. After the optimal airtime allocation solved, the bandwidth is computed by $b_i = a_i L_i$ and sent to the scheduler.

E. Scheduling

In our system, a scheduling module controls the order in which packets are forwarded. When the system determines that the WiFi link is not the bottleneck, the system forwards packets in an unmanaged (FIFO) manner: packets are simply forwarded across the session sockets.

When the WiFi link is identified as the bottleneck, the scheduler enforces the rates assigned by the optimizer in a work conserving manner. Each flow's rate is regulated according to its application class, and the scheduler selects the next packet to transmit as a function of both a flow's need and its required airtime usage to forward its next packet. More specifically, each application session's rate is regulated to prevent it from possibly consuming more than its allocated share of bandwidth b_i. The regulation is achieved via a token bucket mechanism.

Algorithm 1: Scheduler

```
1  while p = recv_packet() do
2      if WiFi link is bottleneck then
3          a = application of p
4          if token_a > 0 then
5              forward packet p;
6              token_a = token_a − p_size;
7              update airtime measurement of a
8          else
9              store p, schedule upon token_a > 0
10     else
11         forward packet p;
```

Algorithm 1 details the scheduling module's implementation. In addition, there are other threads that increase tokens at rate b_i, forward stored packets when $token_a > 0$, and set scheduler back to unmanaged FIFO when WiFi link becomes not congested.

F. Measurements and Feedback

To detect whether the WiFi network becomes bottleneck, we monitor the send buffers on the WiFi side using Unix syscall ioctl(SIOCOUTQ). When the scheduler releases a packet to the WiFi link, the packet is enqueued to the send buffer and waits for the underlying AP's link layer to forward the packet onward. When the WiFi network is not a bottleneck, the WiFi rate is greater than the enqueue rate so the packets in the buffers will be cleared after a short time. In contrast, when the WiFi becomes the bottleneck, packets in the buffer are backlogged. Using this fact, we monitor the size-sum of the buffers of all sessions as a measure to indicate the condition of WiFi network.

In addition, the system measures the actual goodput $\hat{b}_i$ of each queue on the WiFi side. And L_i, the physical link rate, is measured directly from driver using `iwinfo`. These values are updated every 2 seconds and fed back to the optimizer.

IV. Performance Evaluation

In this section, we experimentally demonstrate how Virtual Wires performs in enhancing the reliability and predictability of WiFi performance.

The experiments were conducted in an indoor environment. As shown in Fig.2, we placed the client receivers in two areas with distance at 3 and 75 feet to the router. We ran our tests on WiFi on 2.4Ghz. In our test environment, there

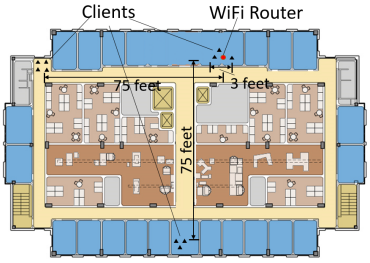
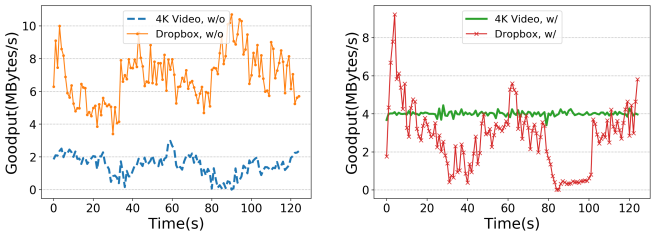


Fig. 2. Positions of WiFi router and receivers



(a) without Virtual Wires (b) with Virtual Wires

Fig. 3. 4K video on mobile v.s. Dropbox at 3 feet

were approximately 50 APs visible to our WiFi router, causing interference that one might expect in a typical setting in cities. Virtual Wires was implemented on OpenWRT [14] operating systems using a Linksys WRT1200AC WiFi router.

1) Reliable bandwidth allocation: In this set of experiments we demonstrate 1) our ability to maintain reliable QoS for high bandwidth video flows, e.g. for 4K video at home; 2) the impact of varying physical rates (through distance change) on the performance, with and without our system.

We stream 4K video at bitrate 30Mbps and we walk with the device back and forth in the following way: the target is at 3 feet from the router when t=0, 60 and 120, and at 75 feet when t=30 and 90. The utility function for this flow is set to a step function: $u(b) = 10$ if bandwidth $b \geq 32$Mbps, otherwise $u(b) = 0$. There is a background receiver, downloading large files from Dropbox.com and staying at 3 feet from the router throughout the experiment, and that flow is tagged as a greedy flow with utility function $0.1\log(b)$.

Shown in Fig 3, Virtual Wires is able to sustain the throughput of the 4K flow, whereas without our system there are wild fluctuations and dips in the download rate. Another thing to note is that with Virtual Wires, when the signal of the video receiver is weak (t$\in [10, 50] \cup [70, 110]$), our utility based optimizer ensures that the air time allocation and targeted QoS are feasible, thus we are able to maintain its throughput demand all the way. As expected we sacrifice the throughput of the background flow.

2) Improved delay performance: We demonstrate the advantage of our system on isolating the delay of applications. We stream a 1Mbps flow and some greedy background flows. We use a step function for the utility of the fixed-rate flow to make optimizer assign bandwidth slightly more than its send rate. Background flows use $\log(b)$ as utility. The delays of the fixed-rate flow is measured by the router to device TCP round trip time. We test the scenarios on two routers "Linksys" and "Onhub", both with(**w/**) and without(**w/o**) our system. Each experiment was repeated 100 times.

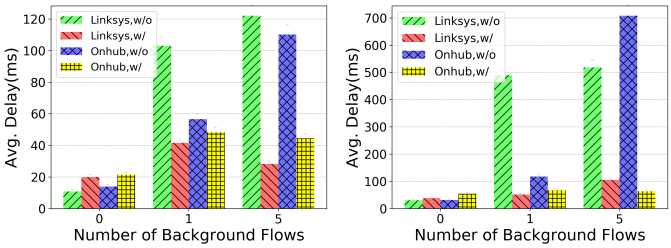


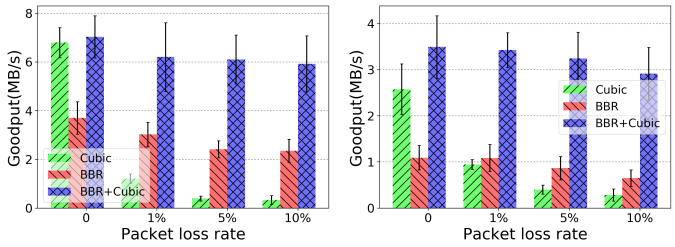
Fig. 4. Delay comparison, receiver at 3 feet(L) and 75 feet(R)

In Fig 4, we show the delay performance of the receiver at two distances. When there is no background flow, the delay overhead introduced by our system is caused by session splitting, kernel/userspace memory copying. When there is even just one background flow, our system already decreases delay by 60% to the Linksys, and such improvement is more dramatic when there are more background flows. In addition, when the receiver is at 75 feet with weaker channel quality, the savings are even more pronounced.

3) Impact of congestion control: We evaluate the impact of congestion control mechanisms on throughput, and demonstrate the advantage of our ability to not only split, but *switch* TCP's congestion control mechanism as discussed in Sec.III-B. We send a flow using BBR and Cubic through an emulated wire path with different loss rate and delay. We show the results in Fig 5 where the wire delay is 10ms, the typical network delay from our WiFi router to Google CDN. We also performed experiments for other wired link delays with the same observation, so the results are omitted here. In this test, we use Cubic TCP algorithm on the wireless link because Cubic outperforms other algorithms (Reno, Vegas, BBR, Bic) on wireless link in our tests. We test when the receiver is at 3 feet and 75 feet to show the impact of distance.

As can be observed in Fig 5, our system, which is "BBR+Cubic", increases link utilization. Though the increase is less observable at 3 feet, it achieve 30.7% at 75 feet compared to "Cubic" without our system when no packet loss on wire. This is because Cubic is frequently affected by packet losses on WiFi when the link is unreliable. However, our system has isolated TCPs for each side so that packet losses on WiFi does not shrink the sending window of the sender. As soon as the WiFi becomes available, our WiFi side TCP adapts faster than "cubic" because our system shorten the feedback loop, and hence improves link utilization.

Cubic outperforms BBR when there is no loss, BBR maintains its performance at losses (validating the design goal) while Cubic falls off. With our system, the link utilization is improved by taking the advantages of both BBR on wired link and Cubic on WiFi. The improvement is more significant when more packet loss introduced on the wire part and longer WiFi distance. We can also observe that BBR performs better at 3 feet compared to 75 feet, since the average delay at 3 feet is more stable than at 75 feet. In other experiments, as the wired link delay increases the gap between BBR and BBR+Cubic reduces as expected.



(a) receiver at 3 feet (b) receiver at 75 feet

Fig. 5. BBR+Cubic v.s. BBR v.s. Cubic at different loss rate

V. Related Work and Discussion

In the literature, session splitting can be traced back two decades and a recent review can be found [15]. Specifically, in [3], the authors propose to split TCP connection by a wired path and a wireless link to handle performance degradation caused by packet loss on the wireless link. In [16] and [17], the authors compare TCP splitting against end-to-end protocols and reliable link-layer protocols. In [18], the authors propose removing TCP congestion control on the last hop when in the presence of packet loss. However, these works focus on improving the performance of each TCP flows, while our work also focuses on resource allocation among multiple competing TCP flows. In [19], the authors mainly focus on the asymmetry property of WiFi and provide priority and fairness control. Their work relies on implementation on the AP's MAC and IP layer which differs from ours.

In this paper, we mainly focus on the home WiFi environment where there is only one bottleneck under the control of Virtual Wires. In the case of multiple APs or channels, the problem becomes more challenging. As the bottlenecks in the system are no longer unique, we need to identify the competing groups among the channels and ranges of the APs. A sophisticated QoS scheduler should also consider the interference amongst APs and clients. Roaming is another problem in such an environment. Using session splitting, we can hand over TCPs between APs to make the application layer session transfer seamlessly [20]. However, the QoS guarantee needs to adjust as the client move to another network with different WiFi conditions.

VI. Conclusion

Our work addresses the challenges of improving utility in a WiFi environment through a cross layer design. Our design supports automatic classification of flows into utility-based categories, based on which, utility-optimizing bandwidth / airtime allocation is made and implemented by a scheduling mechanism. Together, the classifier, the optimizer and the scheduler work to create and deliver, under the available bandwidth constraints, the most appropriate utility to the applications, *completely transparently* to network (including link layer) protocols and applications, requiring no user intervention. As our experiments show, in a wide variety of scenarios, we are able to automatically provision a "virtual wire" for the application, with the right amount of bandwidth, on top of WiFi.

References

[1] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat, "Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing," in *Proceedings of ACM SIGCOMM*, 2015, pp. 1–14.

[2] S. Kopparty, S. V. Krishnamurthy, M. Faloutsos, and S. K. Tripathi, "Split tcp for mobile ad hoc networks," in *IEEE GLOBECOM*, vol. 1, Nov 2002, pp. 138–142 vol.1.

[3] R. Yavatkar and N. Bhagawat, "Improving end-to-end performance of tcp over mobile internetworks," in *First Workshop on Mobile Computing Systems and Applications (WMCSA)*. IEEE, 1994, pp. 146–152.

[4] S. Gollakota, F. Adib, D. Katabi, and S. Seshan, "Clearing the rf smog: Making 802.11n robust to cross-technology interference," in *Proceedings of ACM SIGCOMM*, 2011, pp. 170–181.

[5] V. Bahl, R. Chandra, P. P. C. Lee, V. Misra, J. Padhye, D. Rubenstein, and Y. Yu, "Opportunistic use of client repeaters to improve performance of wlans," in *Proceedings of 2008 ACM Conference on Emerging network experiment and technology (CoNEXT 2008)*, Madrid, Spain, December 2008.

[6] Wi-Fi Alliance, "Wi-fi certified for wmm-support for multimedia applications with quality of service in wi-fi® networks," *Austin, Wi-Fi Alliance*, 2004.

[7] A. Custura, R. Secchi, and G. Fairhurst, "Exploring dscp modification pathologies in the internet," *Computer Communications*, vol. 127, pp. 86–94, 2018.

[8] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[9] W. M. Shbair, T. Cholez, J. Francois, and I. Chrisment, "A multi-level framework to identify https services," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, 2016, pp. 240–248.

[10] M. Shafiq, X. Yu, A. A. Laghari, L. Yao, N. K. Karn, and F. Abdessamia, "Network traffic classification techniques and comparative analysis using machine learning algorithms," in *Computer and Communications (ICCC), 2016 2nd IEEE International Conference on*. IEEE, 2016, pp. 2451–2455.

[11] J. Martin, E. Rye, and R. Beverly, "Decomposition of mac address structure for granular device inference," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 2016, pp. 78–88.

[12] R. J. La and V. Anantharam, "Utility-based rate control in the internet for elastic traffic," *IEEE/ACM Transactions on Networking (TON)*, vol. 10, no. 2, pp. 272–286, 2002.

[13] E. Jones, T. Oliphant, P. Peterson *et al.*, "Optimization and root finding (scipy.optimize)." [Online]. Available: https://docs.scipy.org/doc/scipy/reference/optimize.minimize-lbfgsb.html

[14] "Openwrt: a linux distribution for embedded devices." http://openwrt.org, 2018.

[15] B. H. Kim, D. Calin, and I. Lee, "Enhanced split tcp with end-to-end protocol semantics over wireless networks," in *Wireless Communications and Networking Conference (WCNC), 2017 IEEE*. IEEE, 2017, pp. 1–6.

[16] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving tcp performance over wireless links," *IEEE/ACM transactions on networking*, vol. 5, no. 6, pp. 756–769, 1997.

[17] H. Elaarag, "Improving tcp performance over mobile networks," *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, pp. 357–374, 2002.

[18] F. Le, S. H. Wong, R. Raghavendra, V. Pappas, and E. Nahum, "Removing tcp congestion control on the last hop in split tcp environments," in *Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2016 14th International Symposium on*. IEEE, 2016, pp. 1–8.

[19] A. Gupta, J. Min, and I. Rhee, "Wifox: Scaling wifi performance for large audience environments," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 217–228.

[20] G. Hampel, A. Rana, and T. Klein, "Seamless tcp mobility using lightweight mptcp proxy," in *Proceedings of the 11th ACM international symposium on Mobility management and wireless access*. ACM, 2013, pp. 139–146.