Design Considerations for RINA Congestion Control over WiFi Links

Kristian A. Hiorth, Michael Welzl
University of Oslo, Norway
kristahi@ifi.uio.no, michawe@ifi.uio.no

***Abstract*—Suboptimal TCP congestion control performance over wireless 802.11 (WiFi) networks is a well known issue. We will argue that some of these problems are difficult to overcome when employing end-to-end congestion control such as in TCP. Using measurements, we show that control could better be applied locally, using knowledge about the local WiFi connection and adapting to its characteristics. Under the Recursive InterNetworking Architecture (RINA), using such schemes is a natural architectural consequence, making deployment simpler and more efficient than in the "normal" Internet.**

***Index Terms*—congestion control, wifi, wireless networks, RINA, TCP**

I. Introduction

Internet congestion control traditionally operates end-to-end, using implicit feedback such as packet loss (TCP Reno-like congestion control), delay (Vegas [1], LEDBAT [2] and many others) and attained throughput (TCP Westwood [3], BBR [4]). An end-to-end Internet path may traverse many diverse network links, and can contain several bottlenecks, the position of which can fluctuate over time. This is further exacerbated by the increasingly common—and increasingly drastic—changes of the physical (PHY) capacity, e.g. with recent WiFi links and 5G technology.

Changing the PHY rate breaks TCP's assumption of a static bottleneck capacity, causing poor performance. In case of 3G and 4G networks as well as satellite links, it has become common to counteract this performance degradation by installing "Performance Enhancing Proxies" (PEPs): devices which typically shorten the control loop in one way or another, often by simply splitting TCP connections. Since these devices operate at layer 4 while being located inside the network, they work against common Internet design principles, making them a "persona non grata" for the Internet standardisation body, the IETF. This has led to a discrepancy between research (which mostly adheres "end-to-end" design) and commercial practice (where PEPs are bought and installed because they do improve performance).

In the Recursive InterNetworking Architecture (RINA) [5], PEPs are a first-class citizen—their functionality naturally appears when congestion control is applied [6]. RINA is therefore the right framework for our considerations on applying congestion control not end-to-end, but locally, to better adapt to the peculiarities of a WiFi link. Because RINA is not meant for IETF standardization, the resistance against PEP design is irrelevant in this scope; this of course impairs RINA deployment, but there are possibilities, e.g. using gateways or overlay / underlay solutions. A possible RINA deployment scenario at the network edge is discussed further in [7].

The authors were part-funded by the Research Council of Norway under its "Toppforsk" programme through the "OCARINA" project (http://www.mn.uio.no/ifi/english/research/projects/ocarina/). The views expressed are solely those of the authors.

Using measurements of a WiFi testbed, we show that there is an obvious relationship between easily measurable factors such as the number of actively sending hosts and their total throughput on the one hand, and the expected rate of a newly joining host on the other; this lets us derive the feasibility of designing a mechanism that will work much better in a WiFi setting than its end-to-end counterparts. Even in operational practice, in a WiFi context, PEPs are far less common than in cellular networks—as we will see, this is poor network design, greatly limiting the attainable performance.

We elaborate on our idea of local WiFi congestion control in the next section. In Section III, we present some measurement results from our local testbed that illustrate how our envisioned mechanism could work. We discuss the potential of our mechanism in comparison with today's approaches in Section IV, and conclude after a discussion of related work in Section V.

II. Basic rate estimation concept

The goal of congestion control is to identify and maintain the ideal data transmission rate (or window, in case of a window-based protocol): a rate that utilises the network well, while minimising delay and packet loss, and attaining fairness among competing sources.

In 802.11 WiFi networks, the limiting factor that determines how high throughput can become is the 802.11 Medium Access Control (MAC) protocol's Distributed Coordination Function (DCF). Since the radio medium WiFi operates over is a shared resource, there must be a mechanism that coordinates access. To achieve this, the DCF implements a mechanism called Carrier Sense Medium Access with Collision Avoidance (CSMA/CA). CSMA/CA strives to attain good utilisation and fairness for all actively sending hosts.

For an application, transmitting data with the ideal sending rate for WiFi means to send at the rate that the WiFi DCF allows: sending less leads to under-utilisation (the DCF sees an opportunity to send data, but has no data available in the buffer), but sending more causes a buffer to build inside the sending host, producing undesirable latency.[1] By monitoring the buffer, a congestion control mechanism could determine the buffer drain rate, which is the rate at which the buffer should be filled to keep it constant. This could allow to keep just enough data in the buffer to always allow the DCF to send frames when it needs to, but not more.

There is, however, a caveat with this method: using the buffer drain rate is *reactive*, not *proactive*—to measure it, the sender must first transmit data. How quickly should an application send from the outset, or after a transmission break? Because this question is very hard to answer for end-to-end TCP congestion control, Google's proposal to increase the Initial Window from 3 to 10 packets was only published by the IETF after approximately four years of intense discussion [8]. For local congestion control over WiFi, we may be in a position to give a better answer—an *informed* one, instead of having to define a constant.

Since the wireless medium is shared, and hosts can listen to traffic between any other hosts within their reach, a host can listen to the medium to obtain the necessary inputs and feed them into a model of DCF behaviour in order to obtain an average transmission rate. The performance of the 802.11 DCF has been thoroughly studied and modelled, see for instance [9] and references therein. However, these models impose a number of restrictions and simplifications in order to make the models tractable. The original model by Bianchi [10, 11], for instance, does not take varying physical rates into account. Extended models are often complex, making them hard to use despite their limitations.

We therefore opted for a simpler, more general approach: because a host can measure the throughput obtained by other hosts and see how many there are (via their MAC addresses), and because it is known that the 802.11 DCF aims to attain fairness between hosts, we can use Machine Learning (ML) to predict the attainable sending rate, based on a learned relationship between these factors and the throughput of a host. Using ML, it is also relatively straightforward to consider other parameters that may play a role—the station's transmission power settings, antenna configuration, signal-to-interference-plus-noise ratio etc. We expect an ML approach to be more robust and future-proof than applying a traditional mathematical model because the proprietary nature of several MAC mechanisms (e.g. rate adaptation), deployment-specific configurations (e.g. different TXOP settings, MAC parameters advertised by the AP, etc) and implementation artifacts (e.g. MAC customizations, or more recently even choice of multiple-input and multiple-output (MIMO) modes) can make modelling assumptions fail. In fact, even seemingly basic ones such as seeing stable Channel State Information (CSI) under stable environment conditions, in an empty room, can be wrong with off-the-shelf equipment [12].

We now turn to some local testbed measurements that help us better understand how well our idea may work. The details of the testbed that we used for the following measurements as well as all other measurements in this paper are provided in Appendix A.

[1]For TCP senders on the same host where the DCF operates, this problem has largely been solved in Linux; we will elaborate in Section IV.

III. Experiments

We devised the following experiment: three traffic flows are generated using *iperf 2*, with 10 second staggered starting times. The flows originate on separate wireless testbed nodes and all terminate at the single uplink wired testbed node, emulating an upload scenario. In order to saturate the medium we configured iperf to generate UDP traffic with a throughput target an order of magnitude higher than the physical capacity of the wireless network. In practice this causes iperf to maintain a filled WNIC transmission queue, so that the DCF becomes the bottleneck in the network.

Although physical rate adaptation is an intrinsic part of the operation of real wireless networks, this mechanism can severely obscure the basic functioning of the DCF. Therefore we completely disabled the rate control mechanism in this first experiment and forced all wireless nodes to operate at the 54Mbps PHY rate. We repeated the same experiment at other PHY rates and saw the same trends, so for brevity's sake we do not reproduce these results here.

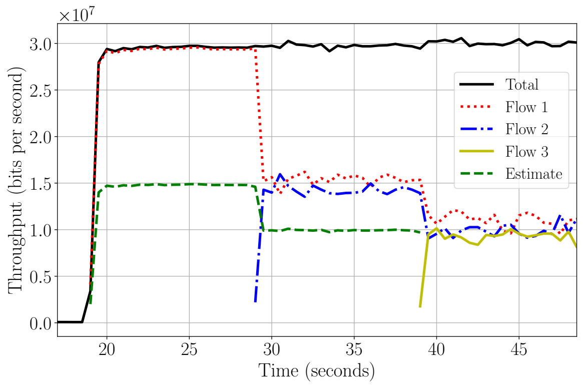


Fig. 1: Throughput achieved by 1 to 3 staggered, overlapping and saturating UDP flows. PHY rate fixed at 54Mbps on all nodes.

Figure 1 shows the attained throughput per host, along with a simplistic estimate that is obtained by dividing the total throughput by $N + 1$ to predict the throughput that a host will attain when it sends, where N is the number of active hosts that were recently seen. Clearly, when there is only one flow active, our estimate is very close to the throughput (which translates into the sender-side buffer draining rate) that is indeed attained by flow 2 when it joins. Similarly, when the two flows are active, the estimate is very close to the later throughput of flow 3. It appears that this simple logic is indeed a feasible approach to obtain an estimate of the attainable sending rate when PHY rates are fixed and equal.

As mentioned above, modern wireless networks are heavily reliant on physical rate adaption mechanisms. This allows them to accommodate nodes that are physically spread around in the environment relative to the access point (AP) and thus subject to significantly different radio transmission conditions in communicating with the AP.

To examine whether our rate estimation method can also work with different PHY rates, we repeated the previous experiment, but now assigned each wireless node a different rate. The node originating the first flow is fixed at a low rate (12Mbps), the source of the second flow is fixed at a medium rate (24Mbps) while the last source node is fixed at a high rate (54Mbps). In all other regards the experiment setup is identical to the previous one.

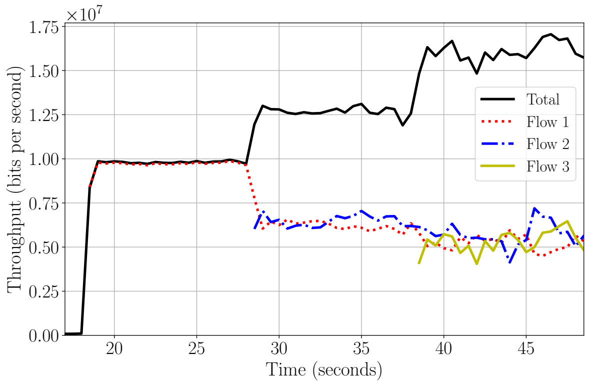


Fig. 2: Throughput achieved by 1 to 3 staggered, overlapping and saturating UDP flows. PHY rates fixed at 12Mbps, 24Mbps and 54Mbps for flows 1, 2 and 3, respectively.

Figure 2 shows the results of the experiment; clearly, while our previous simple division based estimate would not directly work anymore, the relationship between the per-host throughput and the measurable parameters (number of hosts, throughput so far, PHY rates) is not much more complex than in the homogeneous-PHY-rate case.

The behaviour in Figure 2 may in fact seem surprising, as it does not expose the well known WiFi “performance anomaly” [13], where a low-rate flow unfairly reduces the chance for high-rate flows to transmit data because it needs more time than a high-rate flow to transmit its frames. The reason for the unexpectedly fair behaviour in this test is the *ath9k* airtime fairness scheduler [14], which has been included in Linux and enabled by default since version 4.11. We confirmed this by disabling the airtime fairness scheduler; this resulted in the behaviour shown in Figure 3. Clearly, determining the attainable sending rate is much harder in this scenario, and we therefore conclude that the *ath9k* airtime fairness scheduler should be enabled in support of our mechanism whenever possible.

Now we drop the last simplification of our tests, and allow the WiFi MAC to automatically adjust the PHY rates. In our case, with Linux, this is done by the Minstrel algorithm [15]. Figure 4 shows that, although the sending behaviour becomes more dynamic, there is still a clear relationship between the number of flows and the throughput attained for each

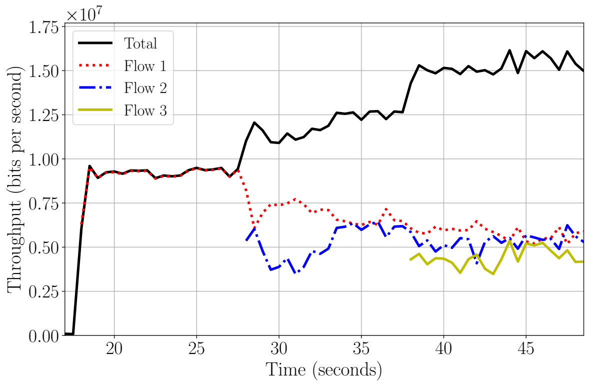


Fig. 3: Throughput achieved by 1 to 3 staggered, overlapping and saturating UDP flows. PHY rates fixed at 12Mbps, 24Mbps and 54Mbps for flows 1, 2 and 3, respectively. *Airtime fairness is disabled.*

flow. While fairness appears to be slightly delayed by the convergence of Minstrel, eventually flows 1 and 2, and later flows 1-3, attain the same sending rate. It seems obvious that an ML algorithm could also be trained to incorporate the behaviour of Minstrel—if not the immediate rates of all flows, then probably the convergence time and a safe lower bound for the rate to be used in the meantime.

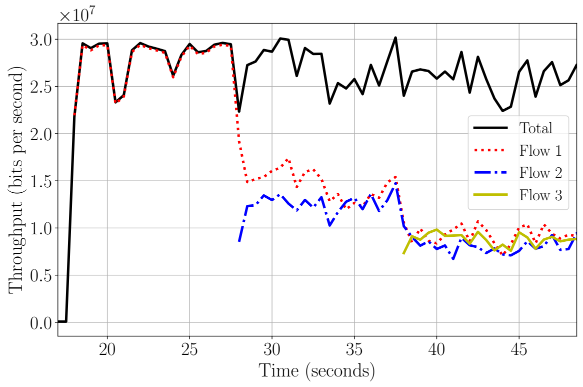


Fig. 4: Throughput achieved by 1 to 3 staggered, overlapping and saturating UDP flows. PHY rates dynamically adapted by the Minstrel rate adaption algorithm.

IV. Discussion of attainable benefits

We started with the presumption that the envisioned mechanism can outperform end-to-end congestion controls such as TCP. As a simple test to check if this is indeed the case, we ran the same scenario as before (1 to 3 staggered flows with a 10 second gap in between), but with end-to-end TCP (three flavours: Reno, Cubic and BBR) across our wireless link and the following wired 300 Mbps connection, with a minimum round-trip time of 100 ms in the wired network. As Figure 5 shows, because the TCP control loop operates at RTT time scales, the total throughput was significantly lower than with the UDP baseline as shown in Figure 1 in all cases: around 25-26 instead of almost 30 Mbps. From these diagrams, it also appears that the 802.11 DCF does a better job at giving a fair share to each flow than TCP in all three cases.

Having established that a mechanism to calculate a sending rate from measuring the wireless medium could work, and that such a mechanism has the potential to outperform even a modern TCP flavour like BBR, we now discuss the possible use cases for local WiFi congestion control. First, for RINA, WiFi is just another link; communicating over it requires the creation of a "shim DIF" (Distributed InterProcess Facility—the RINA equivalent of a layer). To interconnect this WiFi shim DIF with congestion control mechanisms in other DIFs, it is necessary to know the attainable sending rate of the WiFi-DIF. However, in practice, such interconnection of WiFi is often limited to special scenarios such as Wireless Mesh Networks. Often, in a practical WiFi setting, the sender will be located at the wireless host that would also run our rate estimation algorithm. There, a sending application could simply try to send as fast as possible, and leave it to the NIC to find an appropriate upper limit for the sending rate.

Such aggressive transmission necessarily builds up queue space inside the sending host (and makes the sender stop when a queue is full). In case of Linux, this intra-host flow control contains a number of mechanisms that work together to keep the queue sizes as small as possible and avoid unnecessary delay ("bufferbloat"). Directly above the NIC, a mechanism called Byte Queue Limits (BQL) limits the amount of data to be enqueued by the NIC. Above this, a queuing model takes network packets from several sources and, by default, treats them with an algorithm called FQ_CoDel: a combination of a slightly altered form of fair queueing and per-flow Active Queue Management. Above this, TCP's own send buffer is dynamically limited by a mechanism called TCP Small Queues (TSQ). A good overview of this structure is provided in [16].

So, what benefits can a calculated rate yield over this kind of intra-host flow control? Small or not, queues first need to grow and an application must learn to stop sending before the application itself can begin to understand the sending rate. This means that, e.g., an adaptive multimedia application cannot know the ideal rate for a video codec from the beginning, but must "experiment" with quality levels in order to find the one that is closest to the attainable sending rate. Here, having knowledge of a rate from the beginning could make the startup faster and improve it in quality. Pure pushback is a late signal which may have more issues—e.g., generally, in recursive networks, it was found to be a poorly performing feedback method [17]. Rate feedback is inherently faster: an application could be told its rate before it even begins to transmit.

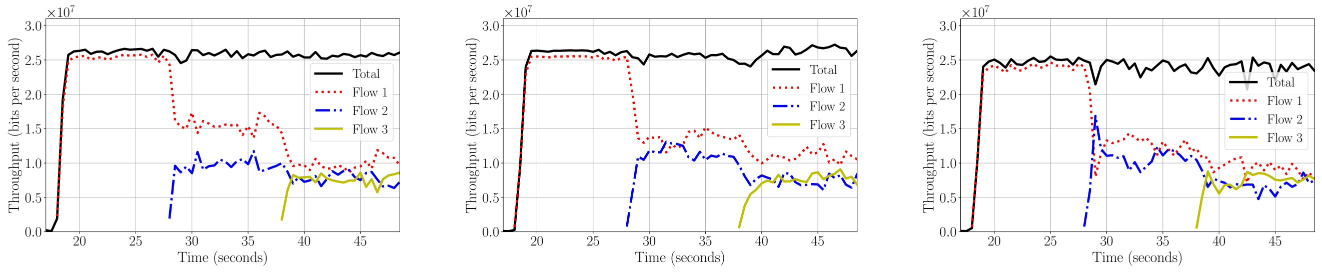
V. Related Work

WiFi, just like many other wireless or otherwise peculiar links, has been considered in many "cross-layer" research ideas. Cross-layer, in the context of transport protocols, usually means that TCP is augmented with information about a particular link [18]. Such approaches hardly see deployment, however, as TCP implementations in end hosts are based on an IETF standard which is designed to operate over any kinds of networks, not just a specific link layer. On several occasions, when cross-layer ideas were brought to standardization, it has turned out that certain open issues have not been addressed by the proposers [19]: how will TCP operate with this new idea when the wireless link in question is not the bottleneck? Can the method scale, does it guarantee security? Sometimes, addressing all of these issues of a wireless cross-layer proposal is not even possible. These difficulties are due to the mismatch between trying to operate efficiently over a specific wireless link on one hand, and requiring congestion control to operate end-to-end on the other.

As explained in Section I, this mismatch is commonly addressed in the Internet by installing PEPs—and PEP-like behaviour is an inherent property of RINA [6]. For a general overview and taxonomy of PEPs, see [20]. The "naturally appearing" PEP behaviour in RINA is connection splitting. Connection splitters "break" a TCP connection (and, hence, its congestion control loop) by acting like a receiver towards the sender and acting like a sender towards the receiver with spoofed IP addresses. Snoop [21] is one of the most well known and studied splitters, but there are other well studied examples such as PEPSAL [22], which is targeted primarily at satellite communication (SATCOM) systems and is still in active development and use. There is ample evidence that TCP PEPs are common in SATCOM systems [23, 24]. PEPs are also commonly deployed in mobile cellular networks (4G/LTE), see for example [25, 26]. There is also continued work on PEP solutions better tailored to operate specifically within 802.11 wireless networks [27, 28].

Generally, not all congestion control approaches use end-to-end control loops. For example, the IEEE P802.1Qcz standard includes a "congestion isolation" mechanism which uses back-pressure to identify congestion-causing flows and to inform upstream switches of congestion such that traffic towards other destinations can be placed in different queues. End-to-end congestion control is also a particularly poor match for Information-Centric Networking (ICN), which decouples data from the location where they are stored. In ICN, hosts emit "Interest" packets, which prompt sources to transmit the requested data back to the host. Since a series of Interest packets may take various paths to different data sources, one common approach to carry out congestion control is to limit the number of interest packets, thereby limiting the returning data. Such "interest shaping" is commonly carried out inside the network, in a hop-by-hop (HBH) fashion [29].

The authors of [30] adapt one particular HBH ICN congestion control scheme to operate across a link with changing capacity, which they derive using a packet train based available capacity estimation technique. The location of the control and the relationship to wireless network characteristics puts this work in close relationship to ours, but its goals are slightly different: we do not target HBH interest shaping, and, instead of a packet train based estimation, we use passive



(a) TCP Reno (b) TCP Cubic (c) TCP BBR

Fig. 5: Throughput achieved by 1 to 3 staggered, overlapping and greedy TCP flows. PHY rate fixed at 54Mbps on all nodes.

listening to directly obtain measurements from the wireless network, allowing us to obtain different information than just the available capacity (e.g., the number of active stations and their physical sending rates).

VI. Conclusion

In this paper, we have discussed the suitability of a local loop solution for congestion control over wireless networks, as opposed to traditional end-to-end mechanisms such as TCP.

Our calculations to derive a rate estimate worked for simple cases with only 3 hosts, but it seems obvious that they will work fine for a larger number of hosts too. The outcome of the decisions by such an algorithm is deterministic; hence, while the relationship between the measurable inputs and the attainable rate is certainly not as simple in a real system as in the cases that we have highlighted in this paper, there is reason to hope that this relationship could be generally modelled via regression with Machine Learning (ML)—e.g., the authors of [31] show that even a full MAC protocol can be created with ML.

In future work, we will develop the algorithm and carry out tests with it. We intend to investigate various RINA use cases: mesh networks as well as intra-host feedback to sending applications, to contrast with the intra-host flow control mechanisms of Linux that we discussed in the previous section. Different from the Internet, where PEPs have limited abilities to communicate with TCP endpoints, in RINA, the rate can be signalled from one DIF to the next, such that an end-to-end path is composed of multiple locally congestion-controlled segments (DIFs). As with the WiFi DIF envisioned in this paper, these locally controlled DIFs can be expected to operate much more efficiently than in the Internet case; this gives RINA the potential to render use cases such as wireless mesh networks much more attractive than they are today.

Appendix

Testbed configuration

All the experiments described in this paper were run on a hardware testbed. Measuring on real hardware rather than relying on simulations ensures the practical relevance of our results. We will now describe the hardware and software configuration of the testbed.

A. Hardware configuration

The testbed consists of 6 identical *Hewlett-Packard HP Compaq 8100 Elite CMT* desktop computers equipped with *Intel Core i7 870* 2.93GHz quad-core CPUs and 16GB of main memory. One of these computers acts as a dedicated control node which orchestrates the operation of the other nodes, but does not directly participate in measurements. Of the remaining five machines, one node acts as router and wireless access point, physically interconnecting two subnetworks to which the remaining test nodes are attached. All test nodes as well as the router have a dedicated, switched wired connection to the control node.

The wireless nodes and the router are all equipped with *D-Link DWA-547* wireless network interface cards (WNICs) based on the *Qualcomm Atheros AR9280* chipset. In all experiments, the wireless nodes are associated to an infrastructure-mode basic service set (BSS) hosted by the router node. The BSS operates in 802.11g mode. This choice enables us to contain the scope of the present work by not needing to consider all of the advanced features of the operating modes present in 802.11n and 802.11ac. One of the wireless nodes is equipped with an additional WNIC that is used to capture raw 802.11 link layer traffic in monitor mode.

B. Software configuration

The testbed is managed using the *TEACUP* network experimentation framework.[2] [32] The testbed nodes run *Ubuntu 16.04.1 LTS* with a *Linux* 4.19 kernel. The kernel is compiled with the default build configuration, with the exception that we have enabled kernel parameters related to debugfs control and inspection of the 802.11 rate control mechanism and of the *ath9k* driver which powers the WNICs.

TEACUP configures emulated network parameters (delay, bandwidth limitations, etc.) on the router using the standard Linux *tc* and *netem* mechanisms. The uplink wired link is configured with a roundtrip delay of 20 ms and a 300 Mbps capacity, which is much higher than the wireless network capacity; this ensures that this link is never the bottleneck in our tests.

[2]Project website: https://sourceforge.net/projects/teacup. Please note that we maintain an enhanced version available from: https://github.com/screw/teacup

References

[1] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," in *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, ser. SIGCOMM '94. New York, NY, USA: ACM, 1994, pp. 24–35.

[2] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)," RFC 6817 (Experimental), RFC Editor, Fremont, CA, USA, Dec. 2012.

[3] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP Westwood: bandwidth estimation for enhanced transport over wireless links," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '01. New York, NY, USA: ACM, 2001.

[4] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 50:20–50:53, Oct. 2016.

[5] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, 2008.

[6] P. Teymoori, M. Welzl, S. Gjessing, E. Grasa, R. Riggio, K. Rausch, and D. Siracusa, "Congestion control in the recursive internetworking architecture (RINA)," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–7.

[7] K. Ciko and M. Welzl, "First contact: Can switching to RINA save the internet?" in *ICIN 2019 Workshop (RINA 2019)*, Paris, France, Feb. 2019.

[8] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, "Increasing TCP's Initial Window," RFC 6928 (Experimental), RFC Editor, Fremont, CA, USA, pp. 1–24, Apr. 2013.

[9] M. Laddomada, F. Mesiti, M. Mondin, and F. Daneshgaran, "On the throughput performance of multirate IEEE 802.11 networks with variable-loaded stations: analysis, modeling, and a novel proportional fairness criterion," *IEEE Transactions on Wireless Communications*, vol. 9, no. 5, pp. 1594–1607, May 2010.

[10] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE Journal on selected areas in communications*, vol. 18, no. 3, pp. 535–547, 2000.

[11] G. Bianchi and I. Tinnirello, "Remarks on IEEE 802.11 DCF performance analysis," *IEEE Communications Letters*, vol. 9, no. 8, pp. 765–767, 2005.

[12] G. Bianchi, S. D. Domenico, M. D. Sanctis, L. Liberati, V. Perrotta, and E. Cianca, "Unveiling access point signal instability in wifi-based passive sensing," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, July 2017, pp. 1–9.

[13] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance anomaly of 802.11b," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, vol. 2, March 2003, pp. 836–843 vol.2.

[14] T. Høiland-Jørgensen, M. Kazior, D. Täht, P. Hurtig, and A. Brunstrom, "Ending the anomaly: Achieving low latency and airtime fairness in WiFi," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, 2017, pp. 139–151.

[15] D. Xia, J. Hart, and Q. Fu, "Evaluation of the Minstrel rate adaptation algorithm in IEEE 802.11g WLANs," in *2013 IEEE International Conference on Communications (ICC)*, June 2013.

[16] C. A. Grazia, N. Patriciello, T. Høiland-Jørgensen, M. Klapez, M. Casoni, and J. Mangues, "Adapting TCP Small Queues for IEEE 802.11 Networks," in *Proceedings of IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Sep. 2018.

[17] D. A. Hayes, P. Teymoori, and M. Welzl, "Feedback in recursive congestion control," in *Computer Performance Engineering*, D. Fiems, M. Paolieri, and A. N. Platis, Eds. Cham: Springer International Publishing, 2016, pp. 109–125.

[18] B. Fu, Y. Xiao, H. J. Deng, and H. Zeng, "A survey of cross-layer designs in wireless networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 110–126, First 2014.

[19] S. Dawkins, "Path Aware Networking: Obstacles to Deployment (A Bestiary of Roads Not Taken)," Internet Engineering Task Force, Internet-Draft draft-irtf-panrg-what-not-to-do-00, Oct. 2018, work in Progress.

[20] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135 (Informational), RFC Editor, Fremont, CA, USA, pp. 1–45, Jun. 2001. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3135.txt

[21] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," *Wireless Networks*, vol. 1, no. 4, pp. 469–481, 1995.

[22] C. Caini, R. Firrincieli, and D. Lacamera, "Pepsal: a performance enhancing proxy designed for tcp satellite connections," in *2006 IEEE 63rd Vehicular Technology Conference*, vol. 6, 2006, pp. 2607–2611.

[23] T. Ahmed, E. Dubois, J.-B. Dupé, R. Ferrús, P. Gélard, and N. Kuhn, "Software-defined satellite cloud ran," *International Journal of Satellite Communications and Networking*, vol. 36, no. 1, pp. 108–133, 2018.

[24] L. Thomas, E. Dubois, N. Kuhn, and E. Lochin, "Quic and satcom," 2018. [Online]. Available: https://arxiv.org/abs/1810.04970

[25] F. Li, J. W. Chung, and X. Jiang, "Driving tcp congestion control algorithms on highway," in *NetDev 2.1*, 2017. [Online]. Available: https://netdevconf.org/2.1/papers/compare-tcp-highway-netdev-4-final.pdf

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

[27] W. Hui, Y. Fukushima, and T. Yokohira, "Throughput improvement of tcp proxies in network environment with wireless lans," in *2014 IEEE REGION 10 SYMPOSIUM*, April 2014, pp. 82–87.

[28] A. Bhartia, B. Chen, F. Wang, D. Pallas, R. Musaloiu-E, T. T.-T. Lai, and H. Ma, "Measurement-based, practical techniques to improve 802.11ac performance," in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 205–219.

[29] Y. Ren, J. Li, S. Shi, L. Li, G. Wang, and B. Zhang, "Congestion control in named data networking – a survey," *Computer Communications*, vol. 86, pp. 1 – 11, 2016.

[30] B. Ahlgren, P. Hurtig, H. Abrahamsson, K. Grinnemo, and A. Brunstrom, "ICN congestion control for wireless links," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2018, pp. 1–6.

[31] Y. Yu, T. Wang, and S. C. Liew, "Deep-reinforcement learning multiple access for heterogeneous wireless networks," in *2018 IEEE ICC*, May 2018, pp. 1–7.

[32] S. Zander and G. Armitage, "CAIA testbed for TEACUP experiments version 2," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150210C, 2015.