Evaluating the Impact of Path Brokenness on TCP Options

Korian Edeline
Université de Liège, Montefiore Institute, Belgium
korian.edeline@uliege.be

Benoit Donnet
Université de Liège, Montefiore Institute, Belgium
benoit.donnet@uliege.be

ABSTRACT

In-path network functions enforcing policies like firewalls, IDSes, NATs, and TCP enhancing proxies are ubiquitous. They are deployed in various types of networks and bring obvious value to the Internet.

Unfortunately, they also break important architectural principles and, consequently, make the Internet less flexible by preventing the use of advanced protocols, features, or options. In some scenarios, feature-disabling middlebox policies can lead to a performance shortfall. Moreover, middleboxes are also prone to enforce policies that disrupt transport control mechanisms, which can also have direct consequences in term of Quality-of-Service (QoS).

In this paper, we investigate the impact of the most prevalent in-path impairments on the TCP protocol and its features. Using network experiments in a controlled environment, we quantify the QoS decreases and shortfall induced by feature-breaking middleboxes, and show that even in the presence of a fallback mechanism, TCP QoS remains affected.

CCS CONCEPTS

• **Networks → Middle boxes / network appliances**; *Network simulations*; Transport protocols.

1 INTRODUCTION

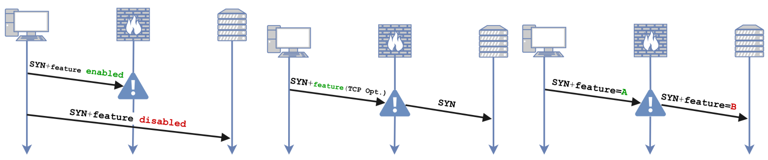
The Internet landscape is constantly evolving. From the original end-to-end TCP/IP architecture, which ensured that all packets exchanged across the Internet would stay untouched in-transit from the transport layer perspective, the last decade has witnessed a progressive introduction of *middleboxes* (i.e., network appliances manipulating traffic for purposes other than packet forwarding [3]). Firewalls and deep packet inspection (DPI) boxes, deployed for security purposes, TCP accelerators for performance enhancement, and network address translation (NATs) boxes have put an end to this paradigm [8].

Today, middleboxes proliferates in large numbers, in various type of networks. In enterprise networks, middleboxes are as numerous as regular network equipment [36]. Tier-1 ASes are deploying more and more middleboxes [7]. Cellular networks are extensively deploying Carrier-Grade NATs (CG-NATs) [40]. Besides, recent progresses in virtualization (i.e., hardware virtualization, containerization) and the introduction of network function virtualization (NFV) are facilitating middlebox deployment [1, 11]. Overall, at least 2% of public network devices are TCP/IP middleboxes, mostly deployed at AS borders, and they affect more than one third of network paths [7, 9].

Although they have made themselves indispensable, by violating the end-to-end semantics, middleboxes have radically changed the transport paradigm. Generic examples of such policies are shown in Fig. 1. As a side effect, they have also introduced a wide variety of impairments to protocols and features, from connectivity, to performance and security issues. Establishing TCP connections with Explicit Congestion Notification (ECN) enabled can lead to connectivity blackouts [26]. Mobile carriers using middleboxes to impose aggressive timeout value for idle TCP connections increase mobile devices battery consumption. Careless TCP middleboxes can facilitate certain network attacks, and even bring new attack vectors [40]. Overall, at least 6.5% of network paths are crossing a middlebox that potentially harms TCP traffic [7, 9].

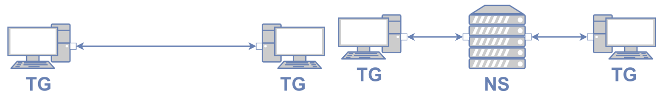
Moreover, middleboxes forbids transport innovation [12]. Often referred to as the ossification of the network infrastructure, this phenomenon consists in middleboxes applying modify or drop policies to packets, and limiting the set of authorized features to a restricted subset. In consequences, alternatives transport protocols that do not rely on TCP nor UDP, such as Datagram Congestion Control Protocol (DCCP) [25], or Stream Control Transmission Protocol (SCTP) [37], despite being standardized, fail to be deployed at large scale. The situation within TCP is similar, with new

ANRW '20, July 27–30, 2020, Online (Meetecho), Spain
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8039-3/20/07…$15.00
https://doi.org/10.1145/3404868.3406662



(a) `feature.blocked` (b) `feature.removed` (c) `feature.changed`

Figure 1: Path Conditions.



(a) Direct **(b) Indirect**

Figure 2: Measurement Setups. TG = Traffic Generator. NS = Network Simulator. Arrows are physical connections.

features being stripped or packets discarded, hampering with TCP innovation [31].

In this paper, we investigate the impact of middlebox-induced TCP feature brokenness that were observed in the wild, in term of Quality-of-Service (QoS). We chose to mimic existing middlebox impairments in a controlled environment, because it requires control on both endpoints and on intermediary devices, with the use of `mmb` [10, 11], a Vector Packet Processing (VPP) [4] plugin that allows to build various stateless and stateful classification and rewriting middlebox policies, and analyze their impact on the TCP traffic. We focus on three basic and widely used features, Explicit Congestion Notification (ECN), Selective ACKnowledgment (SACK), and TCP Window Scaling (WScale), and highlight traffic disrupting policies affecting each feature. Finally, we make all data generated and our Python Notebook freely available to the Research Community[1].

The remainder of this paper is organized as follows: Sec. 2 describes our experimental testbed hardware and configuration; Sec. 3 details the tested features, the chosen experiments and discusses the results; Sec. 4 presents the related works; finally, Sec. 5 concludes this paper by summarizing its main achievements.

2 EXPERIMENTAL SETUP

For quantifying the impact of path brokenness on QoS, we deploy a testbed consisting of three machines with Intel Xeon CPU E5-2620 2.10GHz, 16 Threads, 32GB RAM, running Debian 9.0 with 4.9 kernels. Two of these machines play the role of Traffic Generators (TGs), while one is the Network Simulator (NS). Each machine is equipped with an Intel XL710 2x40GB NIC connected to a Huawei CE6800 switch using one port each for TGs and both for the NS. Traffic exchanged by TGs has to go through the NS first.

The NS relies on *Vector Packet Processing* (VPP) [4], a high-performance userspace packet processing stack, and on the `mmb` and `nsim` plugins to simulate realistic network conditions and middlebox interference. `mmb` is a middlebox plugin for VPP that allows to build various stateless and stateful classification and rewriting middlebox policies [10, 11]. It is used to recreate existing middlebox traffic impairments. `nsim` is a simple network delay simulator implemented as a VPP plugin. It simply adds delay and shapes traffic by processing packet vectors. By tuning `mmb` and `nsim`, the NS can be configured to simulate realistic scenarios of networks with path-breaking middleboxes. The NS runs VPP 18.10, DPDK [20] 18.08 with 10 1-GB huge pages, `mmb` 0.4, and `nsim`, and the TGs run iperf3 [38]. The NS device is configured to maximize its performance, to make sure that it is not the measurements bottleneck.

The TG devices are configured to handle properly Long Fat Networks (LFNs) [21] scenarios (e.g., high bandwidth and high delay), by increasing the TCP receive and send buffers sizes to their maximum value (i.e., 2GB).

We configured our testbed into two different setups: A *direct* client-to-server communication setup, shown in Fig. 2a, is used to evaluate bandwidth baselines and to rule out sender-bounded experiments. An *indirect* setup, Fig. 2b, in which the NS forwards traffic between sender and receiver, and applies the desired network conditions.

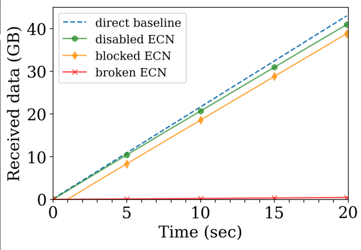
As mentioned above, we generate traffic using iperf. In preliminary, we compute the TGs baseline bandwidth in the direct setup, and the NS overhead in the indirect setup, to ensure that the processing time of the NS is not a bottleneck of the measurements. To this end, we run a single pair of iperf client-server using the direct setup, and we add iperf client-server pairs until the bandwidth reaches the maximum capacity. We found that at least 2 iperf client-server pairs are needed to reach a consistent 37.7 Gbps bandwidth, which is the closest that iperf can get to the maximum capacity of the NICs. The experiment is repeated in the indirect setup, and found a similar bandwidth of 37.4 Gbps.

For the following experiments, we will use a single TCP flow. We will vary different parameters, including the network conditions and the middlebox TCP interference, observe their effect on TCP, and attribute performance deterioration to the parameters. Each experiment lasts for 20 seconds and packets are sized according to Ethernet MTU. All NICs distribute packets to the RX rings by hashing both IP addresses and ports. Each experiment result is averaged

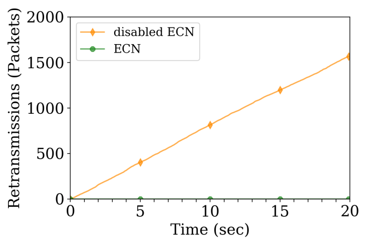
[1]https://github.com/ekorian/vpp-quantify

Path Conditions	Affected Paths [9]	Consequences			
		BT	DF	ND	DT
`tcp.seqnum.changed`	5.5 %	✗	✗	✗	✓
`tcp.opt.sackok.removed`	0.8 %	✗	✓	✓	✗
`tcp.opt.ws.changed`	0.02 %	✗	✗	✓	✓
`tcp.opt.ws.removed`	0.02 %	✗	✓	✗	✗
`tcp.ecn.blocked`	0.01 %	✓	✓	✗	✗
`tcp.ecn.changed`	0.01 %	✗	✓	✗	✗
`ip.ecn.changed`	0.01 %	✗	✓	✗	✓

Table 1: Middlebox Impairments Overview. BT = Blocked Traffic. DF = Disabled Feature. ND = Negotiation Disruption. DT = Disrupted Traffic.



(a) ECN impairments



(b) ECN under Congestion

Figure 3: `tcp.ecn.blocked`, `tcp.ecn.changed`, and `ip.ecn.changed`.

over a thousand runs, except the window scale experiments that are run 50 times.

3 RESULTS

Table 1 lists all middlebox policies we explore in this section. In particular, the column labeled "Affected Paths" provides the proportion of paths affected by those policies, as observed in real traces collected using `tracebox` [6] run between 89 Planet Lab nodes and approximately 600k Alexa top 1M server [9]. We investigate three main policies: the relationships between middleboxes and ECN (Sec. 3.1), between middleboxes and SACK (Sec. 3.2), and, finally, between middleboxes and the window scaling parameter (Sec. 3.3).

First, we measure the direct and indirect baseline bandwidths with a single TCP flow. Given that iperf relies on a single TCP connection, it is bounded to a single CPU and cannot utilize the full capacity of the NICs. We find a throughput value of 34.2 Gbps that we will use as the indirect baseline to compare to the other scenarios.

3.1 ECN

In this section, we investigate interference with the Explicit Congestion Notification (ECN) feature [34]. ECN allows ECN-aware routers with Active Queue Management (AQM) to signal the presence of impending congestion without dropping packets. Instead, it marks packets if they are parts of an ECN-capable TCP connection as experiencing congestion, using a two-bit field in the IP header. The receiver reflects the signaling to the sender, which then reduces its send rate accordingly. The obvious benefit of ECN is the loss rate reduction, but it also improves the TCP throughput, reduces Head-of-Line blocking, and reduces the probability of RTO expiry [13].

We choose to analyze the performances of TCP flows with ECN enabled in the presence of middlebox policies affecting ECN negotiation, or disrupting its proper functioning. In the wild, the latter policies are often legacy IP routers that still consider the IP ECN bits as part of the IP Type of Service (ToS).

We select three realistic scenarios among observations from Table 1: (*i*) `ip.ecn.changed.11`, which consists in the NS rewriting the ECN bits to 11 (i.e., Congestion Experienced) on all packets, (*ii*) `tcp.ecn.blocked`, where the NS blocks ECN-setup SYN packets, forcing the connection to fall back to non-ECN TCP, and (*iii*) fallback-proof `ip.ecn.changed.11`, which consists in the NS rewriting the ECN bits to 11 on all packets already marked 01 or 10 (i.e., ECT(0) and ECT(1), the ECN-capable transport codepoints).

The amount of received data over time of median flows, flows whose total data received is equal to the median, under each of the three aforementioned scenarios is shown in Fig. 3a. The flow crossing the faulty congestion-reporting middlebox of scenario (*i*) (i.e., green curve) displays no substantial QoS deterioration. This is explained by ECN fallback mechanism preliminary ensuring that the path is not ECN-unusable, which is the case when an intermediary device indistinctly marks all packets IP ECN bits to the same value. If ECN is found to be unusable on a given flow, it is disabled. In consequence, the client does not reduce its congestion window, and the flow QoS is unaffected.

The flow that crossed the ECN-setup blocking middlebox of scenario (*ii*) (i.e., orange curve) has an additional second of connection establishment, that is the ECN fallback timeout value. After this delay, a non-ECN connection is established, which displays no sign of QoS deterioration, in comparison to the green curve.

The flow that crossed the second congestion-reported middlebox of scenario (*iii*) (i.e., red curve) shows extreme signs of throughput reduction. In this scenario, the middlebox sets the ECN bits to CE only if the packet is already marked ECT(0) or ECT(1), making it undetectable for the fallback mechanism. Consequently, the faulty middlebox endlessly reports congestion, forcing the sender to reduce its congestion window multiple times.

We also evaluate ECN reduction of packet retransmissions under congestion. To this end, we configure the NS with a token bucket traffic shaping policy, with an average traffic rate of 10Gbps, and a maximum burst size of 10GB. On the token bucket queue, we enable a Random Early Detection (RED) scheduler with a lower threshold of 80%, an upper threshold of 100%, and a drop probability of 10%, to allow high buffer occupancy. We test two scenarios under this setup: (*i*) ECN enabled on the traffic shaping queue, marking the packets as CE instead of randomly discarding them if congestion occurs and, (*ii*), ECN disabled (i.e., `tcp.ecn.changed`, `tcp.ecn.blocked` or `ip.ecn.changed.11`).

Packet retransmission count of each scenario median flow is shown in Fig. 3b. It confirms that ECN allows for reducing congestion window while avoiding unnecessary retransmissions. It should be noted that packets can still be discarded under ECN-marking RED scheduling.

In summary, we showed that ECN fallback successfully deals with ECN-setup SYN blocking and most faulty marking policies. However, we also showed examples of ECN breaking policies that ECN fallback cannot address. Finally, we showed that ECN is valuable to avoid dropping and retransmitting packets that already reached the middle of the path.

3.2 SACK

In this section, we investigate interference with the Selective ACKnowledgment TCP Options (SACK) [16]. SACK consists in two options, SACK-Permitted sent by each endpoints in SYN packets to advertise its support of SACK, and the actual SACK blocks that contains a list of pairs of sequences numbers, each of them acknowledging one or more consecutive packets. Its purpose is to minimize throughput reduction and to avoid unnecessary retransmissions when multiple packets are lost from a single window.

The chosen experiments aim at quantifying the QoS decrease induced by middlebox policies that disable SACK by stripping the SACK-Permitted option from the SYN packets, or by policies breaking SACK. To this end, we introduce a random packet loss rate in the NS.

We select three realistic scenarios (see Table 1): (*i*) loss with SACK, corresponding to the the baseline value, (*ii*) `tcp.opt.sackok.removed` or loss without SACK consisting in the NS stripping the SACK-Permitted option from the TCP SYN packets, and, (*iii*) loss with SACK and `tcp.seqnum.changed` consisting in the NS applying a TCP initial sequence number (ISN) randomizing policy and then rewriting the sequence and acknowledgment number fields of all following packets but not the SACK blocks. This has the effect of making all SACK blocks invalid. The purpose of the latter policies is to fix lacks of ISN randomness, and has been signaled on equipment from major vendors (e.g., CISCO switches).

Fig. 4a and Fig. 4b respectively display received data of median flows, and packet retransmission count of median flows for each scenario, both without artificial loss.

The SACK-enabled flow without broken middlebox of scenario (*i*) (i.e., green curve) has a throughput similar to the baseline, but has more retransmissions events.

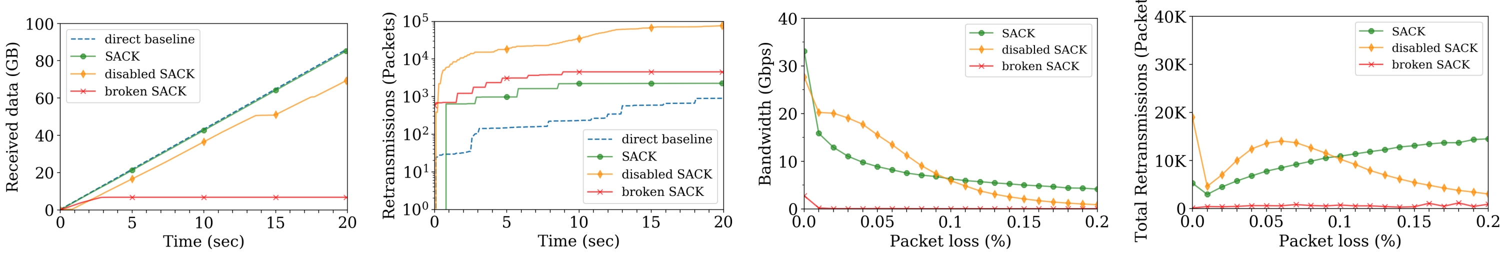
These packet retransmissions are caused by sporadic events of non-artificial loss or reordering, that we observed for 0.001% of packets. The main causes of packet reordering is the inherent parallelism in modern routers [28]. In our setup, the middlebox, the switch, and all NICs involve a multi-queue architecture.

The SACK-disabled flow of scenario (*ii*) (i.e., orange curve) displays a smaller throughput, with three stalling periods, during the first second, between seconds 13 and 15, and during second 17. This stalling periods are caused by multiple consecutive packet reordering at the beginning of the packet window, which, in the absence of SACK, triggers spurious retransmissions from the first Out-of-Order packet to the last packet sent. It is confirmed by Fig. 4b, in which the median orange flow reaches 80,000 retransmissions after 20 seconds.

The SACK-enabled flow with broken middlebox of scenario (*iii*) (i.e., red curve) again shows extreme signs of QoS deterioration. After 3 seconds, it completely stops receiving data. This corresponds to the first loss event, leading the receiver to append SACK blocks to the DUP ACK. Those are then made invalid by the middlebox. The sender implementation (i.e., Linux 4.9) treats packet with invalid SACK blocks by discarding them, without considering its ACK number (i.e., DUP ACK). Then, the receiver indefinitely keeps sending invalid SACK blocks until the sender triggers its retransmission timeout (RTO), sending packets once at a time, without retransmitting the lost packet.

Fig. 4c and Fig. 4d respectively display average throughput and average packet retransmission count in function of artificial packet loss. First, it shows a significant decrease of throughput when introducing artificial loss. Then, we observe that, for packet loss rates between 0.01% and 0.1% the SACK-disabled flow is performing better than the SACK-enabled flow. This is explained by the higher processing time of ACK packets that include SACK blocks. Indeed, due to the linked-last format of SACK blocks, the ACK packets processing time in presence of loss can become too long to keep up with the link speed (i.e., up to 100 times that of a regular ACK packet [24]). For loss rates higher than 0.1%, the SACK-enabled flow has a higher throughput. This is the point where the TCP slowdown induced by the spurious retransmissions of the SACK-disabled flow becomes more important than that of the ACK processing time of the SACK-enabled flow.

In summary, we showed that enabling SACK increases the maximum achievable throughput of flows with low packet loss rates (i.e., < 0.01%), and flows with packet loss rates



(a) No Artificial Loss, Median flow **(b) No Artificial Loss, Median flow** **(c) Average Throughput** **(d) Total Retransmissions**

Figure 4: `tcp.opt.sackok.removed` and `tcp.seqnum.changed`.

higher than 0.1%. However, we discovered that SACK-enabled flows are performing worse than SACK-disabled flows for packet loss rates between 0.01% and 0.1%, due to the extensive processing time of SACK blocks. Finally, we showed that a widespread middlebox policy causes TCP flows to stall after a packet loss event.

3.3 Window Scale Option

Finally, we investigate interference with the Window Scale TCP Options (WScale) [22]. The WScale option is appended by both endpoints of a TCP connection to the SYN packets for advertising the scaling factor of their receive window to the other endpoint. TCP receive window is a 16-bit field with its maximum value limited to 64KB. Wscale introduces a constant left-shift value of up to 14, increasing so the maximum receive window size to 1GB. It is particularly interesting for LFNs that require a high amount of unacknowledged data for optimal performances. Indeed, the window scale should be in line with the bandwidth-delay product (BDP) to fully utilize the available bandwidth, and a 1GB BDP corresponds to a 10Gbps link with a 800ms delay. As the maximum achievable throughput is bounded by $\frac{RWIN}{RTT}$, and the advertised WScale value is determined by the available memory, it should be the highest possible value in order to allow for large BDP when needed.

The chosen experiments aim at quantifying the impact on TCP performances of middlebox policies that unilaterally modify the advertised WScale value. To this end, we simulate LFNs by introducing a delay parameter in the NS. Further, we disable TCP segmentation offload (TSO) and generic segmentation offload (GSO), that seem to bring an upper bound on the BDP, on both TGs for this experiment. Disabling TCP offloading has the effect of pushing more work to the CPU, which reduces the maximum throughput to 12Gbps.

We select three realistic scenarios among observations from Table 1: (*i*) `tcp.opt.wscale.removed` consisting in the NS stripping the WScale option from the TCP SYN packets (this is identical to a WScale option value of 0) and (*ii*) `tcp.opt.wscale.changed` consisting in the NS rewriting the WScale value from both SYN packets with all possible value (from 1 to 14). For both scenarios, we vary the delay parameter from 0 to 800ms, which roughly corresponds to the highest possible BDP for our setup (e.g., 800ms delay at 10Gbps). We also vary the congestion control algorithm, which will influence the TCP window update events. We selected Reno [30], one of the first congestion control algorithms, CUBIC [18], used by default in Linux kernels between versions 2.6.19 and 3.2, H-TCP [27], an algorithm optimized for LFNs introduced in Linux kernel 2.6.13, and, TCP BBR [2] (Bottleneck Bandwidth and Round-trip propagation time), a recent algorithm developed by Google and included in Linux kernels from 4.9.

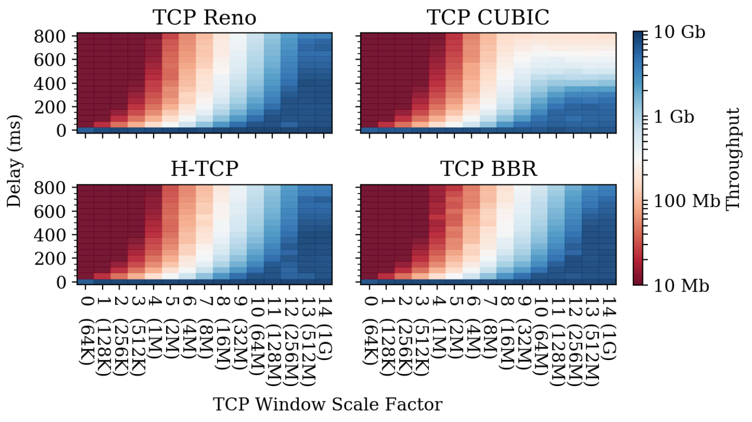


Figure 5: `tcp.opt.ws.removed` and `tcp.opt.ws.changed`.

Fig. 5 displays the median throughput achieved for each tested combination of delay and middlebox-induced WScale value. As defined in RFC 1072, a network is considered as a LFN if its BDP is significantly larger than 10^5 bits (12,500 bytes), which makes all scenarios LFNs but the 0-delay scenario. The latter is affected by a window scaling factor of 0 (up to 3 with TCP CUBIC), and cannot reach a 10 Gbps throughput. Results of LFN scenarios show that in-path modification of the window scaling parameter has a direct impact on the throughput: (*i*) in order to reach a 100Mbps throughput, the minimal window scaling parameter is 6, and it increases to 8 for flows with very high RTTs (i.e., > 400 ms), (*ii*) to reach a 1 Gbps throughput the minimal window scaling parameter is 8 (11 for high-RTT flows), and (*iii*) to reach a 10Gbps throughput the minimal window scaling parameter is 11 (13 for high-RTT flows).

We note that TCP CUBIC performs poorly in extreme BDP scenarios (i.e., WScale higher than 10 and delay higher than 400ms), lowering so the impact of WScale clipping. We observe no significant differences between TCP Reno, H-TCP, and TCP BBR.

In summary, we showed that in-path modifications of TCP window scaling parameters have a direct impact on the maximum achievable throughput, and if middleboxes do not consider the flow RTT in their `tcp.opt.wscale.changed` policies, they are unable to choose a non-impairing value. Moreover, as bandwidth keeps increasing over time, there is a risk that this issue will become more important in the future [5, 17].

4 RELATED WORK

Detal et al. [6] presented an algorithm that is able to reveal in-path middleboxes while requiring control on a single endpoint. Edeline and Donnet [7, 9] extended it and showed that at least 2% of deployed network devices are TCP/IP middleboxes, and that they affect more than one third of network paths. Moreover, 6.5% of paths are potentially impaired by a middlebox, while 1% of paths are highly impaired.

The benefits of using Explicit Congestion Notification (ECN) is a well-studied topic. Floyd et al. [14, 15] showed that a standard TCP flow with a throughput of 1Mbps has an approximate 2% throughput gain with ECN enabled. Salim et al. [35] studied the relative throughput gain of ECN flows versus non-ECN flows in a controlled environment, and evaluated it to up to 60% in high loss high congestion scenarios. Fairhurst et al. [13] listed the benefits of enabling ECN for TCP flows as improving throughput, reducing Head-of-Line blocking, and the probability of RTO expiry. Trammell et al. [39] investigated the deployment status of ECN, alongside connectivity and negotiation issues. They found that TCP connections to 0.42% of hosts experience ECN-dependent connectivity.

Wang et al. [40] revealed the deployment of TCP window checking middleboxes in cellular networks. Qian et al. [32, 33] showed that such middleboxes enable TCP sequence number inference attacks. Hesmans et al. [19] investigated the problem of using Selective ACKnowledgment in presence of TCP sequence number randomizer. They showed that, in such scenario, TCP with SACK enabled performs worse than with SACK disabled. However, their setup has a small maximum goodput (i.e., 10Mbps), which makes it harder to observe the SACK tradeoff.

Jain et al. [23] investigated the impact of the link buffer size on TCP buffer size and throughput. In a controlled environment, they showed that the maximum feasible throughput is bounded by the network buffers. In this paper, we show that a similar phenomenon happens when TCP receive buffers are shrunk by middleboxes. Lukaseder et al. [29] analyzed six TCP congestion control algorithm, including Reno, CUBIC, and H-TCP, in real networks, with relatively small BDPs. They concluded that in absence of loss, the TCP variant has no strong influence on throughput, but as soon as losses are experienced, H-TCP and CUBIC perform better.

5 CONCLUSION

In this paper, we investigated the impact of existing in-path middlebox modifications to TCP ECN, SACK, and WScale. We showed that most ECN impairments are addressed successfully by the ECN fallback mechanism, and that, in the absence of similar mechanisms, SACK and WScale are more vulnerable to path brokenness. Moreover, we showed that all three features are valuable for improving TCP QoS, and therefore even if fallback is able to reduce the impact on QoS, by transforming a traffic disruption into a feature disabling policy, this remains a shortfall for TCP.

In light of the above information, we also recommend operators to: (*i*) not disable ECN, and to make sure in-band congestion signaling is possible, (*ii*) not deploy TCP sequence number re-shuffling policies, because it fixes a vulnerability, but enables another [32], and in case they do, to make sure to include SACK blocks to the mapping to avoid severe TCP impairments, and (*iii*) leave TCP window scaling parameter untouched, as long as there is no information on the flow RTT nor BDP, and therefore on the minimum window scaling parameter that does not reduce the throughput.

We also recommend for host configuration to: (*i*) have ECN enabled, because ECN fallback properly handles the most prevalent path conditions, while unhandled conditions are very rare, (*ii*) have SACK enabled to guarantee a decent throughput in the presence of packet loss, and in the unfortunate presence of a SACK-breaking middlebox, as they tend to be located close to edge networks [7], to consider replacing the offending device if located in their local network, and to switch hosting solution if located in destination network, and (*iii*) have TCP window scaling enabled.

REFERENCES

[1] T. Barbette, C. Soldani, and L. Mathy. 2015. Fast Userspace Packet Processing. In *Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*.

[2] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue – Network Congestion* 14, 5 (September/October 2016), 20–53.

[3] B. Carpenter and S. Brim. 2002. *Middleboxes: Taxonomy and Issues.* RFC 3234. Internet Engineering Task Force.

[4] Cisco. 2002. Vector Packet Processing (VPP). See https://fd.io.

[5] Cisco. 2018. Cisco Visual Networking Index: Forecast and Trends 2017–2022. Cisco White Paper.

[6] G. Detal, b. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet. 2013. Revealing Middlebox Interference with Tracebox. In *Proc. ACM Internet Measurement Conference (IMC)*.

[7] K. Edeline and B. Donnet. 2017. A First Look at the Prevalence and Persistence of Middleboxes in the Wild. In *Proc. International Teletraffic Congress (ITC)*.

[8] K. Edeline and B. Donnet. 2017. An Observation-Based Middlebox Policy Taxonomy. In *Proc. ACM CoNEXT Student Workshop*.

[9] K. Edeline and B. Donnet. 2019. A Bottom-Up Investigation of the Transport-Layer Ossification. In *Proc. IFIP Network Traffic Measurement and Analysis Conference (TMA)*.

[10] K. Edeline, J. Iurman, C. Soldani, and B. Donnet. 2019. *mmb: Flexible High-Speed Userspace Middleboxes.* cs.NI 1904.11277. arXiv.

[11] K. Edeline, J. Iurman, J. Soldani, and B. Donnet. 2019. mmb: Flexible High-Speed Userspace Middleboxes. In *Proc. ACM Applied Networking Research Workshop (ANRW)*.

[12] K. Edeline, M. Kühlewind, B. Trammell, and B. Donnet. 2017. copycat: Testing Differential Treatment of New Transport Protocols in the Wild. In *Proc. ACM/IRTF/ISOC Applied Networking Research Workshop (ANRW)*.

[13] G. Fairhurst and M. Welzl. 2017. *The Benefits of Using Explicit Congestion Notification (ECN).* RFC 8087. Internet Engineering Task Force.

[14] S. Floyd. 1994. TCP and Explicit Congestion Notification. *ACM SIGCOMM Computer Communication Review* 24, 5 (October 1994), 8–23.

[15] S. Floyd. 2003. *HighSpeed TCP for Large Congestion Windows.* RFC 3649. Internet Engineering Task Force.

[16] S. Floyd, H. Mahdavi, M. Mathis, and M. Podolsky. 2000. *An Extension to the Selective Acknowledgement (SACK Option for TCP).* RFC 2883. Internet Engineering Task Force.

[17] A. Gerber and R. Doverspike. 2011. Traffic Type and Growth in Backbone Networks. In *Proc. Optical Fiber Communication Conference*.

[18] S. Ha, I. Rhee, and L. Xu. 2008. CUBIC: a New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operation Systems Review* 42, 5 (July 2008), 64–74.

[19] B. Hesmans, F. Duchene, C. Paasch, G. Detal, and O. Bonaventure. 2013. Are TCP Extensions Middlebox-Proof?. In *Proc. Workshop on Hot Topics in Middleboxes and Network Function Virtualization (HotMiddlebox)*.

[20] DPDK Intel. 2014. Data Plane Development Kit. https://dpdk.org.

[21] V. Jacobson and R. Braden. 1988. *TCP Extensions for Long-Delay Paths.* RFC 1072. Internet Engineering Task Force.

[22] V. Jacobson, R. Braden, and D. Borman. 1992. *TCP Extensions for High Performance.* RFC 1323. Internet Engineering Task Force.

[23] M. Jain, R. S. Prasad, and C. Dovrolis. 2003. *The TCP Bandwidth-Delay Product Revisited: Network Buffering, Cross Traffic, and Socket Buffer Auto-Sizing.* CERCS Technical Reports 193. Georgia Institute of Technology.

[24] I. Järvinen and M. Kojo. 2009. Improving Processing Performance of Linux TCP SACK Implementation. In *Proc. Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*.

[25] E. Kohler, M. Handley, and S. Floyd. 2006. *Datagram Congestion Control Protocol (DCCP).* RFC 4340. Internet Engineering Task Force.

[26] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. 2010. Netalyzr: Illuminating the Edge Network. In *In Proc. ACM Internet Measurement Conference (IMC)*.

[27] D. Leith and R. Shorten. 2004. H-TCP: TCP for High-Speed and Long-Distance Networks. In *Proc. International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet)*.

[28] K.-C. Leung, V. Li, and D. Yang. 2007. An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges. *IEEE Transactions on Parallel and Distributed Systems* 18, 4 (April 2007), 522–535.

[29] T. Lukaseder, L. Bradatsch, B. Erb, R. W. Van Der Heijden, and F. Kargl. 2016. A Comparison of TCP Congestion Control Algorithms in 10G Networks. In *Proc. IEEE Conference on Local Computer Networks (LCN)*.

[30] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. 2000. Modeling TCP Reno Performance: a Simple Model and Its Empirical Validation. *IEEE/ACM Transactions on Networking* 8, 2 (April 2000), 133–145.

[31] G. Papastergiou, G. Fairhurst, D. Ros, A. Brunstrom, K.-J. Grinnemo, P. Hurtig, N. Khademi, M. Tüxen, M. Welzl, D. Damjanovic, and S. Mangiante. 2017. De-ossifying the Internet Transport Layer: A Survey and Future Perspectives. *IEEE Communications Surveys & Tutorials* 19, 1 (2017), 619–639.

[32] Z. Qian and Z M. Mao. 2012. Off-Path TCP Sequence Number Inference Attack – How Firewall Middleboxes Reduce Security. In *Proc. IEEE Symposium on Security and Privacy*.

[33] Z. Qian, Z. M. Mao, and Y. Xie. 2012. Collaborative TCP Sequence Number Inference Attack: How to Crack Sequence Number Under a Second. In *Proc. ACM Conference on Computer and Communications Security*.

[34] K. Ramakrishnan, S. Floyd, and D. Black. 2001. *The Addition of Explicit Congestion Notification (ECN) to IP.* RFC 3168. Internet Engineering Task Force.

[35] J. H. Salim and U. Ahmed. 2000. *Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks.* RFC 2884. Internet Engineering Task Force.

[36] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. 2012. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. In *Proc. ACM SIGCOMM*.

[37] R. Stewart. 2007. *Stream Control Transmission Protocol.* RFC 4960. Internet Engineering Task Force.

[38] A. Tirumala, F. Qin, J. Duagn, J. Ferguson, and K. Gibbs. 2005. Iperf, the TCP/UDP Bandwidth Measurement Tool. See http://iperf.sourceforge.net/.

[39] B. Trammell, M. Kühlewind, D. Boppart, Learmonth I., G. Fairhust, and R. Scheffenegger. 2015. Enabling Internet-wide Deployment of Explicit Congestion Notification. In *Proc. Passive and Active Measurement Conference (PAM)*.

[40] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang. 2011. An Untold Story of Middleboxes in Cellular Networks. In *Proc. ACM SIGCOMM*.