Adaptive Transport Layer Protocols using In-band Network Telemetry and eBPF

Ramyashree Venkatesh Bhat, Jetmir Haxhibeqiri, Ingrid Moerman, Jeroen Hoebeke
IDLab, Ghent University – imec, Ghent, Belgium
Email: [name.surname]@ugent.be

***Abstract*—Many applications use Transmission Control Protocol (TCP) to achieve end-to-end reliable, ordered, and error-free data transfer in the network. The decisions are entirely based on the partial end-to-end information obtained from the acknowledgment packets. With several applications moving towards the wireless domain or wired-wireless domain, there has been advancements in the field of application-network interaction and innovations to obtain real-time network monitoring information on a per-hop basis. This paves a way for extensibility and customization of transport protocols. In this paper, we use detailed real-time in-band network telemetry information to adjust the data transfer at the sender side by modifying the congestion control algorithms in real time. This new technique is tested for different network scenarios and the obtained results indicate that a more network-aware TCP design can greatly increase performance under lossy conditions. The implemented technique illustrates how tighter interactions between higher-layer protocols and the network, in combination with real-time telemetry, can facilitate the way for novel, more adaptive protocol designs.**

***Index Terms*—INT, eBPF, wireless networks**

I. Introduction

The transport layer of the OSI model provides a logical host-to-host communication service. TCP and UDP are the two popularly used transport layer protocols, with TCP differing from UDP because of the ordered, reliable, error-free data transfer, flow control, and congestion control mechanisms. These mechanisms operate based on partial end-to-end information available from the acknowledgment (ACK) packets received from the other end. Existing congestion control algorithms have been particularly designed for wired networks. With several applications moving towards wireless media, it is crucial to design a suitable congestion control algorithm that takes into account the wireless network conditions such as lossy medium, mobility, thick walls, etc. There have been techniques such as split connection approach [1], protocol conversion, link-level approach [2], [3], TCP protocol boosters [4], and several other explicit congestion control mechanisms that address the issue of congestion in wireless media, but with the disadvantage of requiring additional functionality in the intermediate nodes.

Recently there has been progress in the area of application-network interactions [5] for easy extensibility and on-the-fly customization of higher-layer protocols, as well as in the area of network monitoring and verification. In-band Network Telemetry (INT), initially implemented for wired networks, has been extended to wireless networks by designing an architecture and logic to generate and process INT-enabled packets for WiFi-based networks [6]. This design has a low overhead and is capable of collecting real-time information on end-to-end, per-hop and per-flow basis. Considering this, there is a great opportunity for optimizing the behavior of transport protocols by exploiting the in-depth insights that can be acquired from INT in their decision-making. To validate this opportunity, this paper focuses on the TCP congestion control mechanism behaviour, by adjusting the congestion window size, a key variable limiting the data transfer, based on the real-time network information obtained from INT.

To achieve this, it is necessary to first understand the relation between congestion control algorithm behavior and different monitored INT parameters for wireless networks. The recent technology, extended Berkeley Packet Filter (eBPF), provides an option to run mini-programs in a safe virtual machine in the Linux kernel with low overhead and can be used to monitor TCP sockets in real-time. As such, we integrate eBPF with INT to collect real-time data for both TCP behavior and network parameters. From this, we can derive a relationship between INT parameters and TCP parameter changes, e.g. TCP congestion window size. Such derived relation is used to modify in real-time the congestion window size for optimizing the transport layer behavior for different network scenarios.

The rest of the paper is structured as follows, section II provides information about existing works in the area of INT and eBPF. A brief introduction on congestion control and a few algorithms along with the problems in wireless networks is provided in section III. Section IV gives a detailed description of the problem statement. Section V and VI present the methodologies and implementations used to address the problem statements along with the obtained results and its discussion. Finally, the paper is concluded in section VII.

II. Related work

This section discusses about different research works in the area of In-band Network Telemetry and eBPF.

A. In-band Network Telemetry

Continuous network monitoring is one of the key features of future private professional networks. Active and passive monitoring methods are the often used monitoring techniques where probe packets [7] and network device polling [8] are used respectively for collecting network information. Cisco Net-Flow [9], IP Flow Information Export (IPFIX) [10], Simple Network

978-1-6654-2854-5/21/$31.00 ©2021 IEEE

Management Protocol (SNMP) [11] are few examples of protocols that use active and passive monitoring techniques. Since these monitoring methods introduce additional traffic in the network or collect information at fewer network nodes, we opt for a new technique called INT. Even though INT was initially introduced for wired networks using P4 [12], lately it has gained wide attention for monitoring Software-defined networks (SDN) applications. Several researchers have implemented and tested INT for SDN networks and industrial wireless sensor networks [13], [14]. INT-enabled node architecture for SDN-based networks is presented in [6] and the accuracy of the monitoring technique has been validated in terms of monitoring overhead, and network (re)configuration. In this work, we use the INT technique mentioned in [6] as a base implementation and further integrate our techniques to achieve the goal.

B. extended Berkeley Packet Filter

extended Berkeley Packet Filter is a pioneering innovation that can run programs in the Linux kernel without changing the kernel source code or loading the kernel modules or affecting other applications and systems [15]. It has been mainly used for system monitoring purposes, but it can also be used for tracing, security, and networking applications. In-network programmability based on IPv6 segment routing using eBPF is presented in [16]. An extensible Linux TCP stack, with new eBPF callbacks to support user-defined TCP options is implemented in [17]. In combination with P4, eBPF has been used to implement INT in OpenVSwitch (OVS) [13].

The new congestion control designs introduced in this paper are based on INT and eBPF, both with very low overheads, whereas the existing explicit congestion control techniques require additional bits in each packet, are moderately expensive in routers, and require modification of all the intermediate routers and switches. Also, INT gives real-time network updates, hence faster adaption to the network changes.

III. Congestion control algorithms

TCP flows include a series of data packets sent from a sender to a receiver, along with corresponding series of acknowledgments flowing in the reverse direction. At any given time, a sender may send a certain number of packets (known as the congestion window, or *cwnd*) before an acknowledgment. Thus, the size of the *cwnd* controls the rate of data sent for a flow. Using TCP congestion control procedures, a source increases or decreases a flow's *cwnd* based on the network events. The triggering factor for congestion control algorithms is the packet loss event, which can occur due to timeout or 3 duplicate acknowledgements (DUP-ACKs). Every congestion control mechanism has four phases (figure 1), slow start, congestion avoidance, fast retransmit, and fast recovery. Each algorithm is in the slow start phase after a connection is initialized or after a timeout. In this phase, for every packet acknowledged the *cwnd* increases by one MSS (Maximum Segment Size), hence the rate of increase is very rapid and *cwnd* doubles for every round trip time (RTT). When the *cwnd* is more than the slow start threshold (*ssthresh*), the algorithm enters the congestion avoidance phase. Every time a packet loss event is detected by 3 DUP-ACKs, the TCP performs fast retransmit, where it retransmits the missing packet and enters the fast recovery phase. Congestion control mechanisms are different by their behavior or actions taken during each congestion phase.

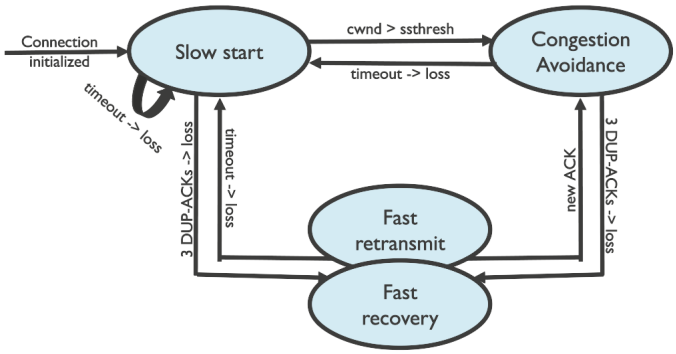

Fig. 1. Different phases of congestion control mechanisms

There are several congestion control algorithms of which TCP Reno and TCP CUBIC are widely used. TCP Reno was initially introduced in 1990 BSD release, after which several versions have been introduced. Currently, New Reno [18] is the recent modification which is available in the Linux kernel. The predecessors of Reno had only a slow start, congestion avoidance and fast retransmit, and fast recovery was introduced in Reno. The New Reno improves retransmission during the fast recovery phase. It introduces a specific algorithm to use partial acknowledgments and uses TCP Selective Acknowledgement (SACK) option. TCP CUBIC was first implemented in 2006 and since then it has been used as a default algorithm in Linux machines [19]. The CUBIC algorithm is real-time dependent and modifies the *cwnd* based on the elapsed time from the last congestion event. Due to the window adjustment mechanism (concave and then convex), the algorithm stability is improved while maintaining high network utilization [20].

Problems in wireless networks: In general wireless networks face several problems with connectivity, security, network expansion, interference, etc., and network congestion being one of them. Network congestion occurs when a buffer is overflown in an intermediate node due to multiple devices using the same network path, or misconfiguration, or a large amount of data transfer. Network congestion causes queuing delay, packet loss, or blocking of new connections, thus reducing the network quality. Apart from this, wireless networks also face packet losses due to wireless medium characteristics (interference, long distance between device, mobility, thick walls etc). The TCP protocol is designed to tackle packet losses due to network congestion only and is thus relevant to wired networks. Since the same mechanisms are used in

wireless networks, these mechanisms generalize each packet loss as a consequence of network congestion, thus reducing the packet transfer rate for every packet loss and resulting in poorer data transfer in wireless networks.

IV. Problem Statement

The advancement in the field of application-network interaction [21] and the innovations to obtain real-time network information on a per-hop basis paves a way for extensibility and customization of transport layer network protocols. Also, as discussed in section III, the existing congestion control algorithms are more suitable for wired networks, and there is a requirement for a better congestion control mechanism for wireless scenarios. We focus on adapting the existing congestion control mechanisms of TCP to the wireless network conditions based on the real-time network information obtained from the INT. To achieve this, we try to answer the following questions,

1) Is there a relationship between the parameters obtained from INT and the congestion window size of TCP?
2) If so, can we use the INT information to modify the congestion window size in real-time?
3) Does the new technique improve the performance of the TCP for wireless network conditions?

V. TCP and INT Parameters Relation

In this section, we derive the relation between the INT and TCP parameters related to congestion control algorithm.

A. Collecting the data points for analysis

To find the relation between the INT data and the congestion window size, it is essential to measure both INT and congestion window size for each packet. For wireless networks, the INT header is encapsulated as an IPv6 option [6]. To collect the INT data and congestion window size at the same time, we integrate the real-time TCP data traced by eBPF in the INT end-to-end option, as shown in Figure 2. Other network parameters, such as queue filling, timestamping, packet losses and wireless link parameters (RSSI, data rate, MCS) are still included as hop-by-hop option in INT data, but are not shown in Figure 2. We also add a TCP flag (which is set to 1 when the data traced using eBPF is included in the extension header) to indicate the presence of real-time TCP data to the sink. All the alignment in the INT extension is done as 4 bytes as required.

B. Implementation

As it is difficult to directly code in eBPF, a tool called BPF Compiler Collection (BCC) provides front-ends in Python and Lua to write BPF programs with kernel instrumentation in C, including a C wrapper around LLVM [1] The INT and INT-enabled node architecture is implemented in the Click modular router framework in [6]. We integrate the INT-enabled node architecture with the BCC library to trace the TCP data in real-time using eBPF. The INT source node adds the real-time congestion window size as an end-to-end parameter inside

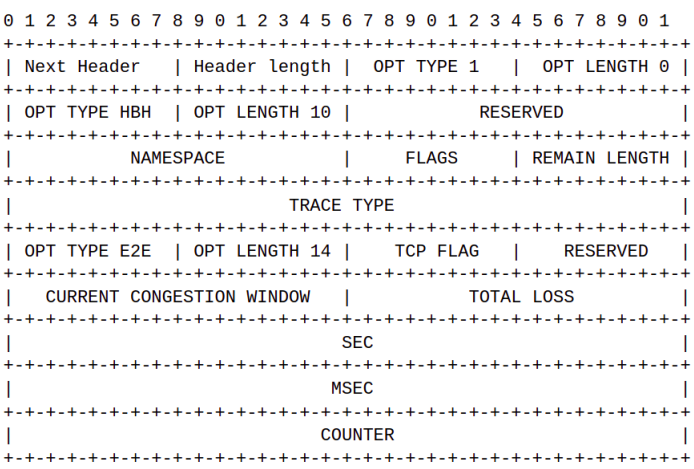

Fig. 2. Integration of eBPF traced data with INT as IPv6 extension header

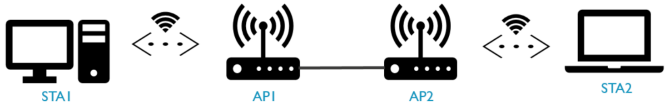

Fig. 3. Multi-AP network setup in Mininet-wifi

the INT data structure, along with the INT header. The data is collected for different network scenarios and the results obtained are explained in the next subsection.

C. Test Setup and Results

The proposed design and implementation are validated on a multi-AP network setup in Mininet-wifi[2] with two access points (AP1 and AP2) and two host devices (STA1 and STA2) as shown in Figure 3. Each interface is assigned an IPv6 address with each of the nodes running the INT-enabled architecture for network monitoring. In this section, we discuss the relationship between the INT parameters and the congestion window size.

To collect sufficient amounts of data, INT along with eBPF traced TCP data is collected for each packet, for a fixed amount of data bytes in iperf3. The data was collected for different network conditions by introducing intentional loss in a wireless link and delay with bounded queue capacity in the intermediate nodes for the CUBIC algorithm. The recorded data points were analyzed over time to understand the correlation between the two parameters. The key outcomes are summarized in the graphs in Figure 4 and Figure 5.

From the graph in Figure 4 it can be seen that after every loss event, there is a reduction in the congestion window size. These loss events are detected by INT and are due to the lossy wireless medium. The higher the number of losses, the higher the reduction in the window size. Hence they are inversely proportional to each other. The second graph in Figure 5 shows the queue filling or packets in the queue in

[1]BCC - Tools for BPF-based Linux IO: https://github.com/iovisor/bcc

[2]https://mn-wifi.readthedocs.io/en/latest/

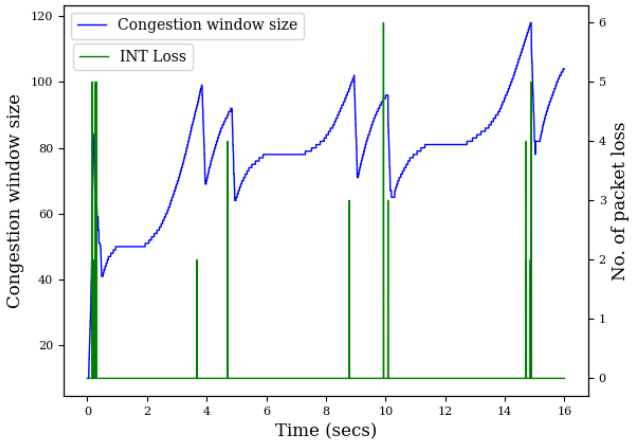

Fig. 4. Relation between packet losses detected by INT and *cwnd*

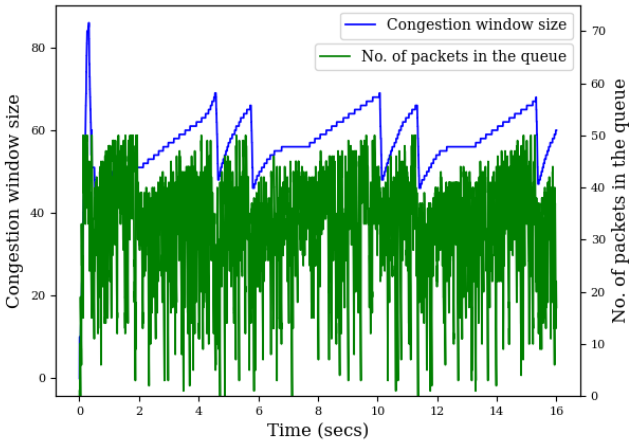

Fig. 5. Relation between the no. of packets in queue and *cwnd*

an intermediate node with bounded queue capacity and the congestion window size at the source over time. When buffer space is available in the intermediate node, the congestion window starts increasing slowly. The buffer space gets filled and the number of packets in the queue keeps increasing. The congestion control algorithm, being unaware of the situation in intermediate nodes, keeps increasing the congestion window size unless there is a packet loss due to the buffer overflow. In the Figure 5 between the points *12 s* and *16 s*, we can clearly see how increasing the congestion window size affects the buffer space in intermediate node. Therefore, the congestion window size increase can be related to the available queue capacity left in the intermediate nodes, that can easily be tracked by INT data. Similarly, an increase in the congestion window size increases the packet arrival rate at intermediate nodes, that can as well be easily tracked by the INT data.

To answer the questions 1 and 2 from section IV, there is a clear relation between the congestion window size and the parameters such as available queue space and packet loss events that are obtained from INT data. As such, these parameters can be used to predict and modify the congestion window size.

VI. INT-BASED ADAPTIVE TCP

Based on the previous section's insights regarding the relation between INT and TCP parameters, in this section, we propose and validate the adaptive congestion control algorithm for TCP based on INT monitored parameters.

A. Modifying the congestion control algorithm based on INT

The INT implementation in [6] collects intermediate node characteristics such as available queue capacity, processing delay, packet arrival rate, flow count, and Tx/Rx timestamping values, along with wireless link information (such as data rate, RSSI, SNR, the channel used, etc.) and end-to-end flow characteristics such as flow latency, flow jitter, and flow packet loss ratio. Since INT collects the detailed information of the intermediate nodes, it can easily differentiate between the packet losses due to buffer overflow and lossy wireless medium based on the flow packet loss ratio and the available queue capacity. The conventional congestion control algorithm reduces the window size irrespective of the reason for the packet loss. Whenever the packet loss is due to the lossy medium, we design two different techniques to maintain the throughput,

1) To keep the congestion window size constant: We implement a new function where on every packet loss due to the lossy medium, the congestion window size is kept constant.
2) To reduce the percentage of multiplicative decrease of the window size: On packet loss, CUBIC Linux decreases the window size multiplicatively by a factor of β, where β is a window decrease constant set to 0.7 [19]. In our design, for every packet loss which is not due to network congestion, the β value of 0.95 is used.

The available queue capacity, packet arrival rate, and flow count information of intermediate nodes indicate the business of the network and the buffer capacity of the node. This information is useful to decide the amount of data transfer at the sender side to avoid congestion in the node. From the result in Figure 5 it is evident that congestion window size is directly proportional to available queue capacity (*available_queue_capacity*) and inversely proportional to the packet arrival rate (*arrival_rate*) and number of flows (*flow_count*) in the intermediate node. On every ACK packet, if the congestion window size is less than the *ssthresh*, the conventional congestion control algorithm increases the window size in steps of one. In our design, we utilize the information obtained from INT and increase the window size based on that. We calculate the congestion window size based on the following formula,

$$cwnd = cwnd + \frac{(available\_queue\_capacity * increment)}{(arrival\_rate * flow\_count)} \quad (1)$$

TABLE I
Behavior of new congestion control algorithms on packet loss due to lossy medium

	β	**Congestion window size (*cwnd*) at loss event**
CUBIC New L1	-	*cwnd* is kept constant
CUBIC New L2	0.95	*cwnd* is decreased

where *increment* is a variable that decides the factor by which congestion window size is increased after every ACK.

B. Implementation

To modify the congestion window size of Linux in real-time, we use the Congestion control plane (CCP) library [22]. CCP is an API by MIT which offers a separate plane for congestion control algorithms, thus enabling the option to write our congestion control methods. The CUBIC algorithm used in Linux is implemented in Python using CCP, which subscribes to a central broker to receive the real-time INT information. Each of the above-mentioned designs is integrated into the existing CUBIC algorithm separately and tested to compare the throughput with the original algorithm.

C. Results and discussion

The proposed design and implementation are validated on the same setup as shown in Figure 3. In this subsection, we firstly present the overall end-to-end performance of the new congestion control algorithm design based on INT data under wireless network loss conditions. Lastly, we discuss the progression of congestion window size, especially during the slow start phase, by comparing it between our proposed enhanced congestion control algorithm and the CUBIC algorithm.

1) End-to-end performance of the new designs: In subsection VI-A, we presented two different techniques to obtain congestion window size for a wireless medium when the packet losses are not due to network congestion. These techniques were tested separately on the emulated setup and the end-to-end performance was measured to compare them with the original CUBIC algorithm. Figure 6 indicates the throughput comparison between the existing TCP CUBIC in Linux and the new designs. CUBIC New L1 and CUBIC New L2 are the algorithms designed to address the issue of packet losses in wireless networks due to reasons other than network congestion, whose parameters are shown in Table I. The throughput values were measured for wireless medium with different loss percentages. From the graph, it can be seen that the new algorithms are capable of maintaining better throughput values even with the higher link loss percentages. In the case of a wireless link with a 20% packet loss ratio, the new designs using the real-time INT data achieve seven times higher throughput than the default Linux TCP congestion mechanism. Even for the scenario without any wireless link loss, CUBIC New L2 performs 12% better than the existing CUBIC algorithm.

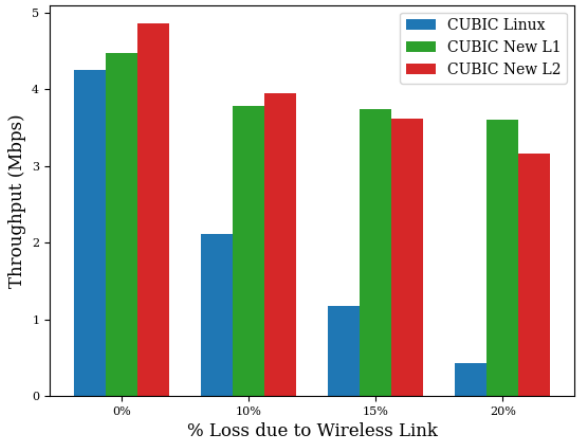

Fig. 6. Comparison of original CUBIC algorithm with the new algorithms

2) Progression of congestion window size: Using the design in equation 1, we increase the congestion window size based on the information obtained from the intermediate nodes using INT. The graph in Figure 7 indicates the evolution of the congestion window over time for *increment* values of 1 (CUBIC New 1), 2 (CUBIC New 2), and 3 (CUBIC New 2) and compared with the original CUBIC algorithm (CUBIC Linux) implemented in Linux.

From the graph Figure 7, it can be noticed that for the algorithms with *increment* values 2 and 3, the congestion window size reaches *ssthresh* way before the original CUBIC algorithm. All the newer algorithms quickly reach the optimal value based on the intermediate node conditions and maintain the same congestion window size throughout the data transfer (especially the algorithm with increment = 1). For wireless network scenario, we prefer the algorithms with *increment* values 2 and 3. They transfer the data consistently over time, with better throughput and lower packet retransmissions.

To answer the question from section IV, the designed algorithms were tested for different network conditions and the results were compared with the original CUBIC algorithm. When the intentional loss was introduced in the network, the designed algorithms not only maintained the throughput, but also achieved higher throughput than the original one. The algorithm designed using equation 1 showed consistency in the data transfer rates and lower packet retransmissions.

Finally to summarise all the answers, from the conducted tests, we were able to correlate between the data obtained from INT and the congestion window size of TCP. Using the recent revolution in the Linux kernel called eBPF, we could modify the congestion window size in real-time. The newly designed techniques were tested for different network scenarios. There was an improvement in the overall network performance as compared to the original CUBIC algorithm, especially for a wireless network with a lossy medium.

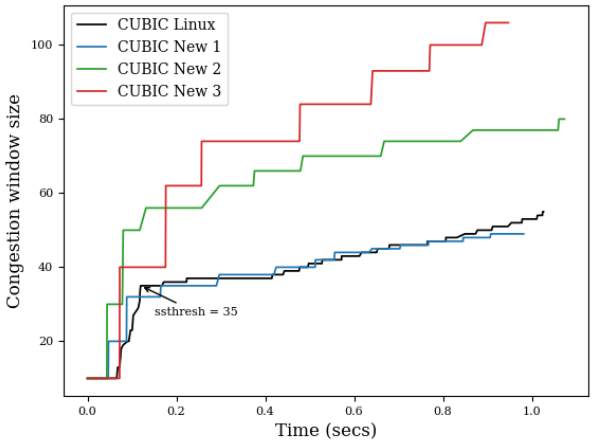

Fig. 7. Progression of congestion window over time for different algorithms

VII. Conclusion and Future Work

The current TCP behavior is mainly based on the partial information obtained from the TCP acknowledgments, which leads to suboptimal decisions, particularly in wireless settings. In this paper, we utilized the innovation of network monitoring and verification in designing newer algorithms for congestion control in wireless networks. With this we could achieve up to seven times better throughput than the existing algorithm under 20% wireless link loss condition. This low overhead design helps the sender to adapt to the changes in the network, thus improving the overall network performance. It provides flexibility to change the data transfer parameters in real-time, allowing designing new transport protocols which can adjust based on the application requirements, resulting in tighter interaction between the protocol and the network. The designs presented in this work are just a step towards achieving tighter application-network interactions. TCP fairness, Flow control, Fast convergence are the areas that are to be explored in the future.

VIII. Acknowledgements

This research was partially funded by the FWO-Flanders under grant agreements #S003921N and #G055619N and by the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie Vlaanderen” program.

References

[1] Bakre, Ajay, and B. R. Badrinath. ”I-TCP: Indirect TCP for mobile hosts.” Proceedings of 15th International Conference on Distributed Computing Systems. IEEE, 1995.

[2] The Snoop Protocol: http://nms.lcs.mit.edu/ hari/papers/snoop.html

[3] A technical tutorial on the 802.11 standard: http://www.sss-mag.com/pdf/802_11tut.pdf

[4] Hoebeke, J., Van Leeuwen, T., Peters, L., Cooreman, K., Moerman, I., Dhoedt, B., & Demeester, P. (2002). Development of a TCP protocol booster over a wireless link. In Proceedings of the 9th Symposium on Communications and Vehicular Technology in the Benelux (SCVT 2002), organised by the IEEE Benelux Chapter on Communications and Vehicular Technology, October 17, 2002, Louvain-La-Neuve, Belgium (pp. 27-34).

[5] Lachos, D., Xiang, Q., Rothenberg, C., Randriamasy, S., Contreras, L. M., & Ohlman, B. (2020, August). Towards deep network & application integration: Possibilities, challenges, and research directions. In Proceedings of the Workshop on Network Application Integration/CoDesign (pp. 1-7).

[6] Haxhibeqiri, J., Isolani, P. H., Marquez-Barja, J. M., Moerman, I., & Hoebeke, J. (2021). In-band Network Monitoring Technique to support SDN-based Wireless Networks. IEEE Transactions on Network and Service Management. vol. 18, no. 1, pp. 627-641.

[7] Metter, C., Burger, V., Hu, Z., Pei, K., & Wamser, F. (2018, November). Towards an Active Probing Extension for the ONOS SDN Controller. In 2018 28th International Telecommunication Networks and Applications Conference (ITNAC) (pp. 1-8). IEEE.

[8] Van Adrichem, Niels LM, Christian Doerr, and Fernando A. Kuipers. ”Opennetmon: Network monitoring in openflow software-defined networks.” 2014 IEEE Network Operations and Management Symposium (NOMS). IEEE, 2014.

[9] Claise, B., Sadasivan, G., Valluri, V., & Djernaes, M. (2004). Cisco systems netflow services export version 9.

[10] B. Claise, “Rfc 5101: Specification of the ip flow information export (ipfix) protocol for the exchange of ip traffic flow information,” IETF, January, 2008.

[11] Harrington, David, Randy Presuhn, and Bert Wijnen. ”RFC3411: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks.” (2002).

[12] Kim, C., Sivaraman, A., Katta, N., Bas, A., Dixit, A., & Wobker, L. J. (2015, August). In-band network telemetry via programmable dataplanes. In ACM SIGCOMM (Vol. 15).

[13] Gulenko, Anton, Marcel Wallschläger, and Odej Kao. ”A practical implementation of in-band network telemetry in open vswitch.” 2018 IEEE 7th International Conference on Cloud Networking (CloudNet). IEEE, 2018.

[14] Karaagac, Abdulkadir, Eli De Poorter, and Jeroen Hoebeke. ”Alternate marking-based network telemetry for industrial WSNs.” 2020 16th IEEE International Conference on Factory Communication Systems (WFCS). IEEE, 2020.

[15] Miano, S., Bertrone, M., Risso, F., Tumolo, M., & Bernal, M. V. (2018, June). Creating complex network services with ebpf: Experience and lessons learned. In 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR) (pp. 1-8). IEEE.

[16] Xhonneux, Mathieu, Fabien Duchene, and Olivier Bonaventure. ”Leveraging ebpf for programmable network functions with ipv6 segment routing.” Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies. 2018.

[17] Tran, Viet-Hoang, and Olivier Bonaventure. ”Making the Linux TCP stack more extensible with eBPF.” Proc. of the Netdev 0x13, Technical Conference on Linux Networking. 2019.

[18] Henderson, T., Floyd, S., Gurtov, A., & Nishida, Y. (2012). The NewReno modification to TCP’s fast recovery algorithm. RFC6582.

[19] Ha, S., Zimmermann, A., Eggert, L., & Scheffenegger, R. (2018). Internet Engineering Task Force (IETF) I. Rhee Request for Comments: 8312 NCSU Category: Informational L. Xu.

[20] Ha, S., Rhee, I., & Xu, L. (2008). CUBIC: a new TCP-friendly high-speed TCP variant. ACM SIGOPS operating systems review, 42(5), 64-74.

[21] Haxhibeqiri, J., Seferagic, A., Bhat, R. V., Moerman, I., & Hoebeke, J. (2021, August). Tighter application-network interfacing to drive innovation in networked systems. In Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration (pp. 53-57).

[22] Narayan, A., Cangialosi, F., Raghavan, D., Goyal, P., Narayana, S., Mittal, R., ... & Balakrishnan, H. (2018, August). Restructuring endpoint congestion control. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (pp. 30-43). Github: https://github.com/mit-nms/ccp