Tighter application-network interfacing to drive innovation in networked systems

Jetmir Haxhibeqiri
jetmir.haxhibeqiri@ugent.be
IDLab, Ghent University - imec

Amina Seferagic
amina.seferagic@ugent.be
IDLab, Ghent University - imec

Ramyashree Venkatesh Bhat
ramyashreevenkatesh.bhat@ugent.be
IDLab, Ghent University - imec

Ingrid Moerman
ingrid.moerman@ugent.be
IDLab, Ghent University - imec

Jeroen Hoebeke
jeroen.hoebeke@ugent.be
IDLab, Ghent University - imec

ABSTRACT

Applications and their underlying network largely operate in isolation, passing data back and forth. For several use cases, such isolation is no longer desirable. Tighter application-network (APP-NET) interactions can lead to a better allocation of network resources for meeting application performance guarantees. Vice versa, applications can become more adaptive to the underlying network context. In this paper, we present the design of an APP-NET interface where applications become able to pass traffic and monitoring requirements to the network and where networks are empowered to share monitoring and feedback information to the applications. The presented design is evaluated for two different use cases, illustrating the potential gains in functionality or performance compared to situations where such application-network interaction is absent.

CCS CONCEPTS

• **Networks** → **Programming interfaces**; **Network monitoring**; **Cross-layer protocols**.

KEYWORDS

application-network integration, INT, Wireless TSN, Wi-Fi

ACM Reference Format:
Jetmir Haxhibeqiri, Amina Seferagic, Ramyashree Venkatesh Bhat, Ingrid Moerman, and Jeroen Hoebeke. 2021. Tighter application-network interfacing to drive innovation in networked systems. In *ACM SIGCOMM 2021 Workshop on Network-Application Integration (NAI'21), August 27, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3472727.3472801

1 INTRODUCTION

Traditionally, applications are designed to be network agnostic, without knowing in real-time much about the network capabilities or the network's ability to satisfy their requirements. As such, the network behaves like a black box towards applications while being not aware of application requirements. At most, what networks can achieve is to distinguish between different traffic classes, a behavior that is not sufficient to treat increasingly diverse application requirements. Currently, network and application control planes are separated and are performed by different entities in the network. In some approaches, the network design is coupled to a specific application and some of the application logic is moved to the network nodes (like the *"in-network computing"* concept [13]). However, in many use cases, the application types and instances in the network can change over time and cannot be addressed by such an approach. Alternatively, the network can expose an API towards applications [11, 12] for expressing their requirements. Still, the network configuration is performed by a central entity that can hardly be dynamic enough to deal with changes in the application requirements. Also, applications can neither verify network performance nor can enforce certain network configurations directly.

In many dynamic use cases, like in AR/VR applications, wireless time-sensitive applications (AGV communication, assembling-line communication, robot communication), etc., integration between applications and networking is desirable. Moreover, integration between the data plane and the control plane is still a challenge. As such, application requirements should be shared via the data plane while network nodes can react (e.g. reconfigure scheduling) to such requirements and enrich the data plane with real-time monitoring capabilities. On the other hand, higher layer protocol logic can be optimized based on real-time network feedback. In order to move towards real-time network application integration, novel ways of interacting and exchanging a variety of information need to be defined, involving both the end devices and the network nodes.

For utilizing network reconfiguration and higher layer optimization for improved application performance real-time application-network integration needs to be in place, (i) where application data and control plane are integrated, by carrying and performing both applications (APP) requirements and monitoring in-band in real-time, and (ii) where network nodes and application understand and take actions based on such information.

In this paper, we present a network-application integration approach for private professional wireless networks, where both end devices and network nodes can be controlled. The following main contributions are made. (i) We design and implement the Application Network Agent (ANA) that enables applications to pass on real-time their traffic requirements and their monitoring needs, and enables the network to provide feedback towards applications. (ii) We design the logic of network nodes to process application

NAI'21, August 27, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8633-3/21/08…$15.00
https://doi.org/10.1145/3472727.3472801

requirements as well as monitoring and feedback information. (iii) We evaluate the design in two different case studies.

2 ANA DESIGN AND IMPLEMENTATION

Real-time application-network (APP-NET) integration need to be bi-directional and all the exchanged information need to happen in-band with the data plane. With such core principle we targeted a modular design integrating the application's data plane and the control plane by placing an Application Network Agent (ANA) in the path between the application (APP) and the network stack (NET). Such design leads to a network and system independent APP-ANA interface and a network stack specific ANA-NET interface. Via the APP-ANA interface, applications can pass their information regarding their identification, traffic requirements and monitoring needs to the network, while the network can pass back information towards applications, such as the monitored performance of the traffic flow as well as monitoring feedback from the other end node.

Figure 1 shows the architecture of ANA and its interaction with the applications and the network. While we have shown how the in-band monitoring is done in [4], here we describe the ANA architecture and how the feedback information is encapsulated in-band. ANA has a modular architecture that consists of a number of adapters that enable communication towards multi-plane network stack in southbound and towards in northbound.

Transport adapter offers interaction between ANA and the transport layer. For end-to-end communication, transport protocol parameters are crucial, especially in the case of connection-oriented transport protocols. As such, the transport adapter enables exposure of the TCP layer information (e.g. congestion and flow control window size) while in the north-south direction permits application to adapt TCP behavior.

Monitoring adapter interfaces ANA with the in-band network telemetry (INT) plane. The monitoring plane is implemented as an in-band feature in the data plane, where certain data packets will collect monitoring info in an end-to-end and in a per-hop and per-flow fashion, with support for wireless links as well [5]. As we consider only private professional networks that do not expand to a large number of network hops, introduced INT overhead is limited [4].

Feedback adapter interfaces ANA with the feedback plane to receive the monitored information from the in-band monitoring and feedback plane to be passed via the northbound interface towards application(s). Two different types of feedback are distinguished: feedback of data that was collected for the traffic flow with the node as destination and the feedback of data sent from the destination node to the traffic flow initial node. In the first case, the feedback is local as the monitored data are already available in the network monitoring stack. In the second case, the feedback data are received in-band via the reverse path from the destination node. Application requirements, monitoring as well as feedback data are encapsulated in the IPv6 extension header and reported in-band using data packets. To reduce the overhead, the application requirements and feedback data are first encoded using Concise Binary Object Representation (CBOR) before encapsulation.

Exchanged data between ANA and applications and network, respectively, are modeled as JSON data structures. The data structure that ANA passes towards the monitoring stack and the network consists of three main parts: *the application (APP) identifiers, the APP properties* and *the APP requirements* as shown in List 1. The *APP identifiers* include the node ID identifying uniquely the traffic flow initial node, the APP ID identifying uniquely a type of application on that node, and the namespace ID that identifies uniquely each traffic flow of the same APP. The *APP properties* describe the properties of the traffic generated by the application (parameters such as traffic periodicity, payload size, etc.), while the *APP requirements* contain the list of required parameters to be fulfilled by the network. Similarly, the data structure that the network pass to ANA contains *the APP identifiers* to which the monitored data are linked to, *the network path* the monitoring packet followed, and the *INT measurement* part. The *INT measurement* part will contain the end-to-end measurement (type 0) and hop-by-hop monitored data (type 1).

Listing 1: Application requirements that ANA passes to monitoring stack.

```
{
  "app-ctrl:cc":{
    "identifiers":{
      "dev-id":334512,
      "app-id":4325674,
      "ns-id":1234
    },
    "app-prop":{
      "traffic-types":[
        {
          "tt-id":1,
          "type":"periodic_loop",
          "payload_size_bytes":64,
          "period_ms":128
        }
      ],
      "network-paths":[
        {
          "np-id":1,
          "src-ip":"10.0.0.1",
          "src-port":"*",
          "dst-ip":"10.0.0.15",
          "dst-port":"80",
          "proto":"udp"
        }
      ]
    },
    "app-req":[
      {
        "global":true
      },
      {
        "global":false,
        "tt-id":1,
        "np-id":1,
        "jitter_ms":20,
        "rtt_ms":100
      }
    ]
  }
}
```

Listing 2: INT monitoring report towards ANA.

```
{
  "app-ctrl:int":{
    "identifiers":{
      */ Same parameters as in APP list... */
    },
    "network-path":{
      */ Same parameters as in APP list... */
    },
    "int-measurements":[
      {
        "time":6745787244513115940,
        "type":0,
        "measurements":[
          {
            "Source_TS":6745787239823774206,
            "Latency_E2E [ms]":5,
            "counter":338
          }
        ]
      },
      {
        "time":6745787244513115940,
        "type":1,
        "measurements":[
          {
```

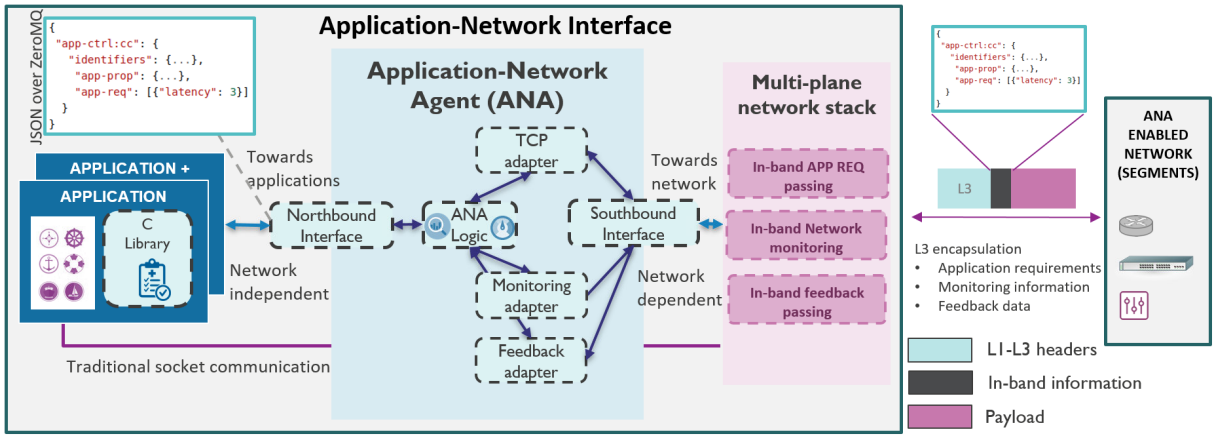


Figure 1: Application Network Agent (ANA) architecture and its integration within network.

```
            "Retransmitted ":0,
            "DataRate_[Mbps]":0,
            "Queue_Length[pkts]":0,
            "Received_TS":1570625986395274000,
            "Node_id":11879605
            */ Other parameters ... */
          },
          ....
        ]
      }
    ]
  }
}
```

ANA logic processes the data received from both ends and creates the respective JSON data structures. In the southbound direction, the monitoring stack will encapsulate APP requirements JSON data structure into the data packet for injecting it into the network and to the destination node. Based on application requirements, ANA logic will instruct the monitoring stack on which flows and which monitoring parameters to monitor, e.g. for latency requirements ANA asks for monitoring of Rx/Tx time-stamping at each network hop, for reliability requirements it asks to monitor losses on a per-hop basis, etc. In the other direction, ANA will receive the INT monitoring feedback as a JSON data structure, extract the necessary information and pass it towards the application.

Application code changes and technical overhead should be limited to enable focusing on high-level features. Current applications interact with the network stack by opening a communication socket via which the datagrams are being passed. This traditional interface is extended with a unified, network-independent interface towards ANA, realized through ZeroMQ[1] (ZMQ) messaging. Currently, to integrate this interface with existing applications, a C library is provided to be embedded with a few lines of code. The ZMQ messaging entity is used to interconnect the ANA logic with adapters and APPs. ZMQ has been chosen as a lightweight implementation of messaging library that supports different messaging patterns, as well as broker-based or broker-less PUB/SUB. The ZMQ communication is performed only locally between services in the node, and for short messages (the case with ANA data structures) the added latency[2] is negligible compared to E2E communication latency.

[1]https://zeromq.org/
[2]http://wiki.zeromq.org/results:copying

3 EVALUATION

In this section, we demonstrate the usage of ANA in two given examples where performance and novel functionalities are achieved and compare it with other SoT solutions.

3.1 Case studies

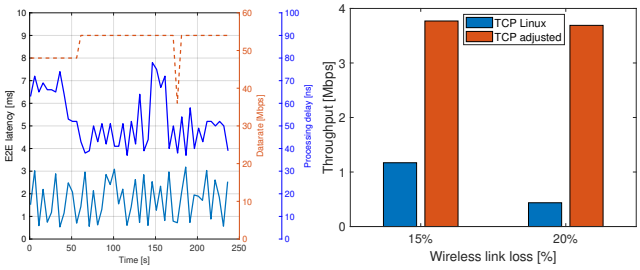
3.1.1 Network monitoring and configuration instruction can be controlled by the applications sharing their requirements in-band. An application passes its requirements to ANA as follow:

```
zmq::socket_t ANA_socket;
zmq::message_t msg;
...
msg.build(src_IP,src_port,dst_IP,dst_port,trans);
bool flag = ANA_socket.send(msg,latency);
```

In this case, the application opens a ZMQ socket towards ANA and passes its identifiers together with the requirements (in this case end-to-end latency). Once ANA receives the application requirements, ANA will instruct the monitoring plane about the traffic flow to be monitored, based on the *APP identifier* part, as well as parameters to be monitored, based on the *APP requirements*. As such, the monitoring plane will collect timestamping and processing latency on each network hop by setting the monitoring bitmap [4]. Using this telemetry data, the end-to-end latency and jitter can be determined. In addition to this, based on throughput requirements, ANA can instruct the monitoring plane to collect info on the data rate of the wireless channel.

In an E2E wireless-wired (1 wireless link and 2 wired links) network topology in Wilab2 testbed, the network is instructed to monitor and to maintain a certain level of E2E latency (3 ms <) for certain time-sensitive (TS) traffic flow. Note that each network node supports time-aware shaping that is configured based on in-band application requirements. Figure 2a shows the E2E latency of a TS traffic flow under network saturated condition with a co-existing UDP stream. Note that only the wireless hop monitoring information and E2E latency for TS flow are shown.

3.1.2 Feedback-based adjustment of higher layer protocols based on the monitored data can be achieved via ANA. It is known that the congestion control mechanism of TCP can not distinguish between



(a) Monitoring instruction **(b) Feedback-based TCP adjustment**

Figure 2: Impact of ANI for different scenarios.

packet loss due to wireless issues or network congestion. Regardless of the packet losses cause, the TCP congestion mechanism will reduce the amount of data sent. Real-time detailed monitored data are used to distinguish between such events, so the application maintains the TCP throughput under wireless losses.

Figure 2b shows the throughput comparison between the default TCP congestion control mechanism in Linux with the adjusted control mechanism based on feedback data from the monitoring plane, for different wireless link loss ratios. The measurements were carried out in a multi-AP network topology setup with two wireless and one wired hops in between the end nodes. It is seen that for the cases with a higher packet loss ratio in wireless links, the feedback monitored data helps the TCP congestion mechanism to maintain the throughput. In the case of a wireless link with a 20% packet loss ratio, the adjusted approach based on the monitored data achieves six times higher throughput than the default Linux TCP congestion mechanism.

3.1.3 ANA Packet processing capabilities can be compared with other SoTA solutions, to evaluate its performance. For performing such measurements we generate data packets that encapsulate application requirements and/or INT monitoring information in addition to the data payload. Measurements are performed in a setup where two nodes (3.3GHz CPU, 256GB RAM running Ubuntu 18) are connected by wire. The wired setup is used for its capabilities to support higher data rates, so we can stress test the proposed design. The first node will generate the INT enabled traffic while the other node will process such packets and reply with the monitored data. To compare our results with the results given in [9], we send 1000 Kpps, where packets are INT-enabled with 50B data payload. Measurements are done 50 times for 30s long. Based on such measurements, ANA can process up to 998 Kpps compared to 855,573 Kpps in [9], resulting in only 2 ‰ throughput decrease compared to IPv6 traffic processing capabilities.

4 RELATED WORK

Attempts to make networks application aware are not new and they can be categorized in four main groups: out-of-band unidirectional [6, 7, 11], out-of-band bidirectional [10], in-band unidirectional [2, 3, 8] and in-band bidirectional [9]. Schmidt et.al [11] extend the socket interface with capabilities to support application demand sharing towards network stack. This way *socket intents* allow the application to share information about their communication patterns, however, the network configuration is still done via the control plane. Recently, application-aware IPv6 Networking [8] and segmentation routing (SRv6) [3] allow the host application to convey application information into the network infrastructure. As such, the network can quickly adapt and perform the necessary network (re)configuration to maintain certain performance guarantees. Miyasaka et.al [9] proposed a network API framework for application-aware traffic engineering (TE) using SRv6 that allows end-host applications to include a TE behavior inside the IP packet, while giving feedback to applications only on failure events.

In addition to application requirements exposure functions, network monitoring functions need to be covered as well to support network-aware applications. The new approach of in-band network telemetry (INT) [1] that offers possibilities of fine-grained network monitoring is extended to wireless networks as well [4], offering full in-band network monitoring information exposure. Zhang et.al [14] integrate in-band and out-of-band network monitoring exposure, reducing the necessity of datapath packet format updates or end-host modification when full in-band is used.

ANA approach benefits compared to out-of-band and unidirectional APP-NET integration are evident. It offers integration of in-band monitoring and feedback transparent to the application side. On the other hand, the detailed feedback towards application enables higher layer optimization (as shown in section 3.1.2), compared to insufficiency of only-on-failure feedback in [9]. Further, due to in-band realization, proposed solution reduces only by 2 ‰ the actual data throughput.

5 CONCLUSION

In this paper, we presented a design approach for a rich and extensible application network interface. Such design, on the one hand, enables applications to pass their identification, traffic properties, and requirements, as well as monitoring needs towards the network, while on the other hand, the network can monitor the achieved performance of individual traffic flows and feedback such information towards applications. The ANA design offers a unified, network-independent interface for applications to express their requirements and to get feedback from the network.

The potential benefits of the proposed ANA architecture has been evaluated in two different use cases. In the first use case, ANA is used to enable network monitoring and reconfiguration according to the application requirements. The second use case demonstrates how feedback information from the monitoring plane can be used to adjust the transport layer behavior for achieving better throughput under wireless link losses. Next to this, the processing capabilities of the data plane are not affected in comparison to cases where such monitoring and feedback plane is absent and are 10% higher compared to other SoTA solutions.

6 ACKNOWLEDGEMENTS

This research was partially funded by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" program, and by the FWO-Flanders, under grant agreements #G055619N and #S003921N. Authors thank also Dr. Kai Gao and anonymous reviewers for their valuable feedback.

REFERENCES

[1] Frank Brockners, Shwetha Bhandari, and Tal Mizrahi. 2021. *Data Fields for In-situ OAM.* Internet-Draft draft-ietf-ippm-ioam-data-12. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-ippm-ioam-data-12 Work in Progress.

[2] Kristian Riktor Evensen, Karl-Johan Grinnemo, Audun Fosselie Hansen, Naeem Khademi, Simone Mangiante, Patrick McManus, Giorgos Papastergiou, David Ros, Michael Tüxen, Eric Vyncke, et al. 2015. The NEAT Architecture. *Online) https://www. neat-project. org* (2015).

[3] Clarence Filsfils, Pablo Camarillo, John Leddy, Daniel Voyer, Satoru Matsushima, and Zhenbin Li. 2019. *SRv6 Network Programming.* Internet-Draft draft-filsfils-spring-srv6-network-programming-07. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-filsfils-spring-srv6-network-programming-07 Work in Progress.

[4] Jetmir Haxhibeqiri, Pedro Heleno Isolani, Johann M Marquez-Barja, Ingrid Moerman, and Jeroen Hoebeke. 2021. In-band Network Monitoring Technique to support SDN-based Wireless Networks. *IEEE Transactions on Network and Service Management* (2021).

[5] Jetmir Haxhibeqiri, Ingrid Moerman, and Jeroen Hoebeke. 2019. Low overhead, fine-grained end-to-end monitoring of wireless networks using in-band telemetry. In *CNSM2019, the 15th International Conference on Network and Service Management.* 1–5.

[6] Benjamin Hesmans and Olivier Bonaventure. 2016. An enhanced socket API for Multipath TCP. In *Proceedings of the 2016 applied networking research workshop.* 1–6.

[7] Sebastian Kiesel, Wendy Roome, Richard Woundy, Stefano Previdi, Stanislav Shalunov, Richard Alimi, Reinaldo Penno, and Y. Richard Yang. 2014. Application-Layer Traffic Optimization (ALTO) Protocol. RFC 7285. https://doi.org/10.17487/RFC7285

[8] Zhenbin Li, Shuping Peng, Daniel Voyer, Chongfeng Xie, Peng Liu, Chang Liu, Kentaro Ebisawa, Stefano Previdi, and Jim Guichard. 2019. *Application-aware IPv6 Networking (APN6) Framework.* Internet-Draft draft-li-apn6-framework-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-li-apn6-framework-00 Work in Progress.

[9] Takuya Miyasaka, Yuichiro Hei, and Takeshi Kitahara. 2021. Networkapi: An in-band signalling application-aware traffic engineering using srv6 and ip anycast. *IEICE TRANSACTIONS on Information and Systems* 104, 5 (2021), 617–627.

[10] Inder Monga, Chin Guok, John MacAuley, Alex Sim, Harvey Newman, Justas Balcas, Phil DeMar, Linda Winkler, Tom Lehman, and Xi Yang. 2018. SDN for end-to-end networked science at the exascale (SENSE). In *2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS).* IEEE, 33–44.

[11] Philipp S Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. 2013. Socket intents: Leveraging application awareness for multi-access connectivity. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies.* 295–300.

[12] Gregory Vander Schueren, Quentin De Coninck, and Olivier Bonaventure. 2017. TCPSnitch: Dissecting the Usage of the Socket API. *arXiv preprint arXiv:1711.00674* (2017).

[13] Yuta Tokusashi, Huynh Tu Dang, Fernando Pedone, Robert Soulé, and Noa Zilberman. 2019. The case for in-network computing on demand. In *Proceedings of the Fourteenth EuroSys Conference 2019.* 1–16.

[14] Yunfei Zhang, Gang Li, Chunshan Xiong, Yixue Lei, Wei Huang, Yunbo Han, Anwar Walid, Y Richard Yang, and Zhi-Li Zhang. 2020. MoWIE: Toward Systematic, Adaptive Network Information Exposure as an Enabling Technique for Cloud-Based Applications over 5G and Beyond. In *Proceedings of the Workshop on Network Application Integration/CoDesign.* 20–27.