Towards an API for the Path-Aware Internet

Thorben Krüger
OVGU Magdeburg, Germany
thorben.krueger@ovgu.de

David Hausheer
OVGU Magdeburg, Germany
hausheer@ovgu.de

ABSTRACT

Path-aware networking (PAN) architectures promise to support practical multipath communication with increased availability and to accommodate heterogeneous communication requirements.

However, at a time when most applications merely rely on "HTTP(S) as a Substrate" for their networking requirements, the quick adoption and effective use of such new communication capabilities at the application level is challenged from a deployment standpoint. This raises the question of how to create an adequate application programming interface for a path-aware Internet that comes with measurable improvements in network communication.

Using concepts from current IETF-IRTF in-progress documents, our work reasons about the demands and implications of path-aware application programming, sketches a possible approach and provides an early glimpse of our practical implementation of a suitably abstract interface to a next-generation networking stack.

CCS CONCEPTS

• **Networks** → **Programming interfaces**;

ACM Reference Format:
Thorben Krüger and David Hausheer. 2021. Towards an API for the Path-Aware Internet. 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/10.1145/3472727.3472808

1 INTRODUCTION

Emerging networking architectures promise to empower Internet hosts through a range of additional communication capabilities and guarantees. With path-aware networking (PAN) in particular, hosts gain the ability to actively steer their traffic according to dynamic constraints, prevailing network conditions and application requirements [8].

On the application side however, Internet data-exchange has been following a trend of increased avoidance of specialized protocols and technologies in favor of "HTTP(S) as a Substrate" as the pragmatic lowest common denominator for simple and reasonably reliable network interaction [5]. While from the point of view of individual applications, this approach often is indeed the most rational choice [6], it has also led to a higher level of general ignorance about underlying Internet technology among application developers. In order to address the widening discrepancy between the sophisticated capabilities soon available on the networking side and the typical communication features required on the application side, the IETF TAPS Working Group [5] is (as of mid-2021) drafting a novel, abstract transport services API [13]. An interface like this would enable an application to simply indicate high-level communication requirements and characteristics like, e.g., "constant-rate streaming" or "capacity-seeking", leaving the concrete network-level realization to the underlying communication stack.

Our work draws significant inspiration from [13] to reason about the required capabilities of a suitable corresponding networking back-end in the context of the next-generation path-aware SCION Internet architecture [8]. We explore some of the resulting implications and apply our insights to design the "PANAPI" system, which leverages the novel opportunities of a path-aware network architecture to maintain an up-to-date local view of current network properties in a central database. Based on this information, path alternatives are ranked and chosen in response to application requirements that are expressible in simple, high-level terms through the next-generation API. In this way, path-awareness has a measurable benefit on the application side, without further complicating the task of network communication.

2 BACKGROUND

In the following, we look at the status quo in network programming and traffic optimization based on the traditional BGP-based Internet, and then put this in contrast with emerging path-aware networking approaches such as the SCION Internet architecture.

2.1 Network Programming Status Quo

On the current, BGP-based, Internet, most user-facing applications these days perform network communication via one of two mechanisms (see Figure 1):

(1) TCP or UDP, using thin, programming-language specific abstractions over the BSD socket interface, or
(2) more abstract libraries with convenient interfaces that behind the scenes usually use "HTTP(S) as a Substrate"

The first approach leaves implementing any traffic optimization mechanisms that are not already dealt with in the OS kernel or the network medium itself to the application. Some attempts [10] have been made to extend the Socket Programming paradigm to support novel networking capabilities, ultimately contributing [2] to the wider recognition of the general need for "An Abstract Application Layer Interface to Transport Services" [13], which is currently under active development in the TAPS Working Group [5] at the IETF and which we discuss below.

The second, more popular approach based on the use of higher-level (mostly HTTP-based) libraries frees the application programmers from worrying about some of the low-level networking details. While some HTTP(S) libraries [12] or DNS resolvers [7] implement optimizations like RFC 8305 [9], their scope is limited and their APIs are hardly standardized.

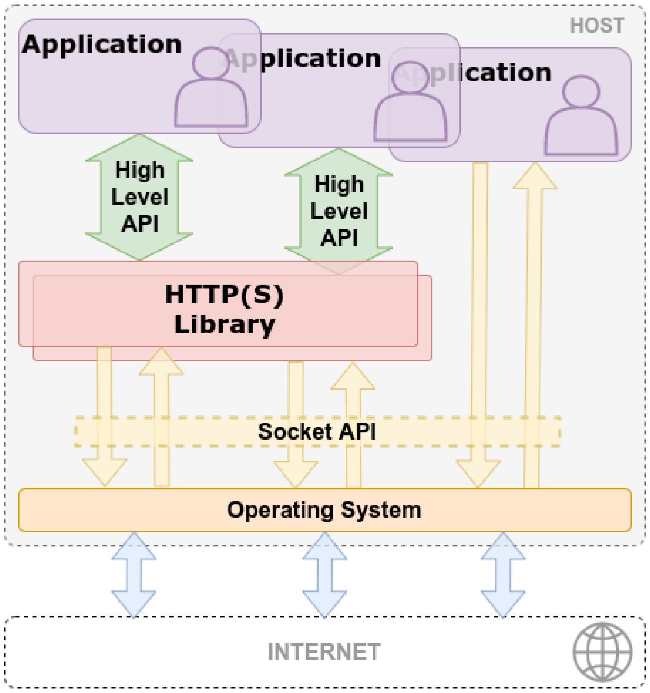

Figure 1: Network Programming Status Quo: Most applications use "HTTP(S) as a Substrate"; interaction with the low-level Socket API directly has become increasingly unusual.

2.2 Traffic Optimization Status Quo

On the BGP-based Internet, various techniques to achieve improved communication have been proposed on the different network levels.

On the application layer, the "happy eyeballs" approach from RFC 8305 [9] relies on simultaneous connection establishment attempts across competing network substrates (i.e., IPv4 and IPv6). In such a "connection racing" scenario, the connection with the quickest established handshake is likely to also deliver the lowest latency in general and will be used henceforth in favor of any runner-ups.

In a somewhat related approach on the routing side, the proposed REPLEX [3] traffic engineering algorithm balances traffic loads across alternative routes by allocating traffic portions in an exploratory manner while monitoring performance. On the other hand, Espresso [15] is an established BGP-based inter-domain traffic-engineering system that uses the privileged vantage point of a large CDN to leverage centrally aggregated performance observations for fine-grained traffic steering and improved user experience.

The general problem of how to best select from a range of possible paths on the host is already familiar from peer-to-peer (P2P) networking research: resources offered by P2P systems often exist in multiple replicas, i.e., they are accessible across "paths" of different lengths and, therefore, of different desirability.

The "Application-Layer Traffic Optimization (ALTO) Problem Statement" from RFC 5693 [11] explores the p2p-centric question of how to obtain and distribute the necessary information for applications to perform better-than-random peer selection, i.e., to pick shorter paths over longer ones, preferring local copies of the data to remote ones.

One early proposed solution is an "oracle" mechanism [1] that could be deployed by ISPs and can be queried by applications wishing to improve performance. The idea is that both the users and the ISPs would benefit from such an arragement.

In a data-center context, the somewhat related, learning-based "Prophet" [4] system centrally processes network performance data to predict throughput of reactive (TCP) flows.

2.3 Next-Generation Network Programming

In recognition of the problems associated with using "HTTP(S) as a Substrate", the IETF TAPS Working Group envisions an API that

> "[. . .] supports the asynchronous, atomic transmission of messages over transport protocols and network paths dynamically selected at runtime. It is intended to replace the traditional BSD sockets API as the common interface to the transport layer, in an environment where endpoints could select from multiple interfaces and potential transport protocols." [13]

If such an interface is realized as intended according to the above excerpt, applications are set to gain the ability to express high-level communication demands via a modern API while leaving lower-level optimizations of network behavior to the underlying system that provides this interface.

At the time of this writing, the concrete specification is still work-in-progress and in a draft stage at the IETF. Nevertheless, it already provides a realistic framework to explore a plausible future way of network communication from the application perspective.

2.4 Path-awareness in the SCION Architecture

The SCION architecture offers *path-awareness* on inter-network granularity through a design based on *packet-carried forwarding state*. It cryptographically guarantees that any autonomous system (AS) on the chosen path can only be entered and exited via the exact interfaces specified in the packet header for that network [8].

On the SCION Internet, every host can select paths to a destination on its own. Given the desired destination, it is the responsibility of each AS to provide its hosts with reasonable path *options*[1], while the final path *selection* authority lies with the hosts.

While a network can dynamically restrict set of path options it is prepared to offer to its hosts, it has little additional influence on the concrete choice of paths that the hosts make. However, good network resource utilization is in the interest of all parties. On the host-side, there is an incentive to find paths with unused resources to maximize performance. On the network side, better usage of otherwise under-utilized paths is equivalent to better balanced traffic patterns and less chance for congestion.

The standard SCION network API currently only applies simple, static heuristics (e.g., path length) to automatically select communication paths on behalf of an application. While this represents a reasonable default, there is no simple way for an application with specific communication requirements to influence this process and adapt the path choice to its specific requirements.

[1]"Path options" do not necessarily mean "ready-made paths". In SCION, path servers disseminate "path segments" from which the hosts then construct complete paths.

3 CHALLENGE

We now turn to the following question:

How to design a system that supports a useful application programming interface for the path-aware Internet?

3.1 Requirements

Based on the background given above, we can now identify and reason about the requirements for a meaningful interaction with a path-aware Internet:

Measurable benefit. To gain the necessary acceptance among users and developers, path-awareness must have a meaningful and measurable benefit from the perspective of the application. It must result in increased network performance or reliability, while not making the task of implementing network communication any more difficult for the developer when compared to the status quo.

Access to path quality information. Selecting paths for a given communication requirement depends on the ability to obtain information about the suitability of each option, beyond simple heuristics merely based on "path length".

Good network utilization. The concrete path choices that are made by the networking stacks on the hosts on behalf of their applications directly translate to concrete traffic patterns in the larger network. Inaccurate path quality information could result in sub-optimal network utilization and have a detrimental effect on application performance.

3.2 Consequences

Taken together, these requirements have clear implications:

Path selection must be automated behind useful abstractions. In particular, programming applications with path-aware features *must not require* dealing with the selection of suitable paths from sets of alternatives directly. Optimized path choice should rather be conducted *on behalf* of the application and behind higher-level abstractions that represent the desired communication properties in a straight-forward manner. These must be automatically mapped to suitable paths that achieve those properties. Assessing path suitability in turn requires access to path quality information.

Path properties must be actively collected. In the absence of a dedicated mechanism for dissemination of path quality information on the network level, independent collection of path properties is required on each host.

Path quality information must be re-used. When re-assessing the suitability of a path on behalf of different applications, reference to any recent, already obtained path properties can help to improve the selection process.

Path-awareness must be provided through one host-wide service. Such a service will necessarily be ideally positioned

- to ensure that different applications running in parallel are not interfering with each other
- to direct traffic across maximally disjoint paths
- to respect system-wide configuration parameters and run-time preferences
- to focus the implementation efforts of low-level network optimizations, traffic engineering approaches and new features behind a single, general-purpose networking API

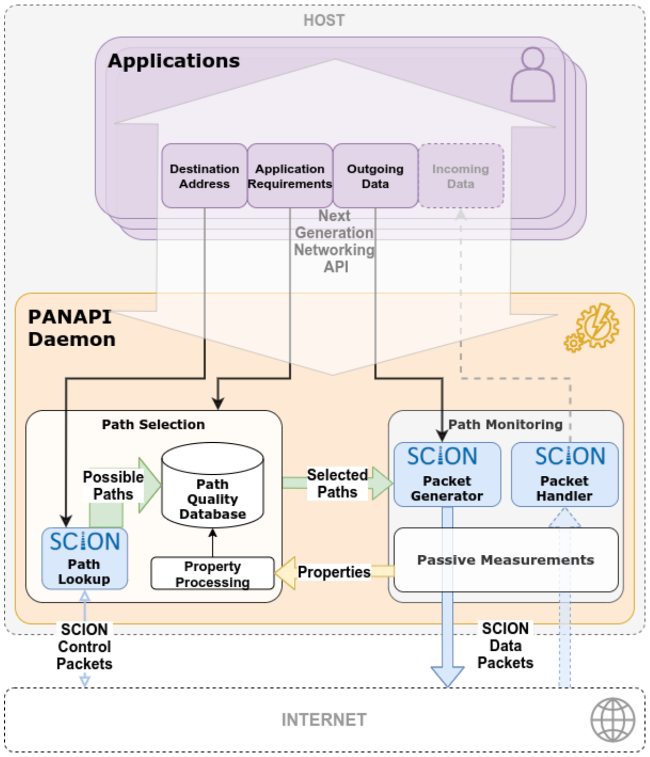

Figure 2: PANAPI - High-level system overview.

These insights form a rough framework for our prototype of a next-generation networking stack.

4 PROPOSED APPROACH

We are currently in the early stages of development of a path-aware networking stack for the SCION architecture, entitled "PANAPI". It consists of two core components: the developer facing abstract networking API as a front end, and the daemon in the back end that is responsible for processing observed and assessed performance data and performing path selection in accordance with application requirements. For a high-level overview, see Figure 2.

Networking API. The design of our API closely follows the ideas developed in [13]. One example are "Capacity Profiles" which an application can use to indicate

> "the desired network treatment for traffic sent by the application and the tradeoffs the application is prepared to make in path [. . .] selection to receive that desired treatment. [. . . T]he Transport Services system SHOULD select paths [. . .] to optimize the tradeoff between delay, delay variation, and efficient use of the available capacity based on the capacity profile specified." [13]

Such high-level next generation features have a profound impact on network programming. The example in Listing 1 shows all the necessary API interactions that are required to request that a) the new connection to a remote host should use multiple *aggregated* paths in parallel, and b) paths that have a high available network *capacity* are preferred.

Property Collection. Our principal method to obtain path property information is through passive measurements on end hosts. Here, directly observed path performance on active flows give indications about metrics such as *bandwidth, round-trip-time* and *jitter.* Where available, less transient information in the form of long-lived properties like, e.g., *link capacity, MTU,* or even the *CO_2 footprint* of a path segment complement the directly observed metrics.

Further measurements can also take place at a larger scale on network ingress and egress. Network equipment can monitor metrics such as *queue lengths* or *link utilization* additionally.

Property Processing. Currently, a host has to rely exclusively on locally collected path property information to inform its path choice. Recently observed communication peformance on a path provides a first estimate of that path's suitability for future communication.

Additional information that has been obtained at the network level could also be incorporated in the final decision. By default, the path selection process does not rely on the presence of any dedicated network-level property services. Purely host-based path quality assessments form the core of our approach.

Path Selection and Exploration. When no recent information is available about the potential paths to a destination, simultaneous connections are initiated over all possible paths in parallel. The completion times of the different handshakes give a first indication about the latencies on each respective path and serve as initial data points to be entered into the path quality database. This is an adaption of the "connection racing" idea from RFC 8305 [9].

Similar to the REPLEX [3] mechanism, the system also occasionally migrates non-critical connections to alternative paths while closely monitoring the associated performance. The change is immediately reversed if the relevant communication metrics do not improve. Otherwise, a better path has likely been discovered that could subsequently also be used for more critical communication.

4.1 Preliminary Findings and Outlook

While our proposed solution is still under development, we have already demonstrated the practical feasibility of a next-generation networking API in the spirit of the IETF TAPS proposal through a first working prototype [14], which enables high-level network application programming in the manner depicted in Listing 1.

Carefully reasoning about the requirements for the PANAPI daemon has resulted in a promising design for the service architecture depicted in Figure 2 that we have already begun to implement. While it is a bit too early to present concrete results, the important pieces are all starting to fall into place. In particular, we are in the process of identifying several useful strategies for passive path performance monitoring, path property derivation and informed path choice.

PANAPI is envisioned to soon serve as a platform to test novel traffic steering approaches in a path-aware environment and to explore new ideas that could have over-arching benefits for network applications in general. Once the core functionality has been realized on a purely host-based basis, further research will focus on improvements for path quality assessments, e.g., via an external "path quality oracle" or "throughput prophet" service that could be queried.

```
import "panapi"

r := panapi.NewRemoteEndpoint()
r.WithSCIONAddress("19-ffaa:0:1303,10.2.1.89")
r.WithPort(1337)
r.WithProtocol("QUIC")

t := panapi.NewTransportProperties()
// pre-requiste for requirement a)
t.Set("multipath", "active")

p := panapi.NewPreconnection(r,t)
// requirement a)
p.SetProperty("multipathPolicy", "aggregate")
// requirement b)
p.SetProperty("capacityProfile", "capacitySeeking")

Connection := p.Initiate()

// simplified asynchronous connection handling
C <- Connection.Ready()
C.Send(Request)
Response <- C.Receive()

Connection.Close()
```

Listing 1: Code example for interacting with PANAPI, demonstrating connection setup with advanced networking features and simplified asynchronous connection handling.

5 CONCLUSION

We have shown that path-awareness has novel implications for practical network programming on both the application-facing as well as the network-facing side. For reasons of simplicity and developer acceptance, path-awareness must be largely hidden from the application in favor of a more abstract communication interface in the spirit of current IETF work. Accordingly, this requires automation of lower-level network interactions.

In the absence of a dedicated way to obtain information about the suitability of each of the different possible network paths that could be picked on behalf of an application, the networking back-end must maintain a database with any already assessed path quality information. To maximize its usefulness, this database must aggregate any available information from the host's network interactions. This leads us to the insight that all network communication on the host must ideally be managed by a central system. With PANAPI, we combine these insights into an integrated design for a next-generation networking stack.

REFERENCES

[1] Aggarwal, V., Feldmann, A., and Scheideler, C. Can ISPs and P2P users cooperate for improved performance? *ACM SIGCOMM Computer Communication Review 37*, 3 (2007), 29–40.

[2] Enghardt, T. *Informed access network selection to improve application performance.* Doctoral thesis, Technische Universität Berlin, Berlin, 2019.

[3] Fischer, S., Kammenhuber, N., and Feldmann, A. REPLEX: Dynamic Traffic Engineering based on Wardrop Routing Policies. In *Proceedings of the 2006 ACM CoNEXT conference* (2006), pp. 1–12.

[4] Gao, K., Zhang, J., Yang, Y. R., and Bi, J. Prophet: Fast accurate model-based throughput prediction for reactive flow in dc networks. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications* (2018), IEEE, pp. 720–728.

[5] IETF. Charter for Working Group. Tech. Rep. charter-ietf-taps-02, Internet Engineering Task Force, Mar. 2021. Work in Progress.

[6] Moore, K. On the use of HTTP as a Substrate. RFC 3205, Feb. 2002.

[7] Obser, F. unwind(8); "happy eyeballs", Nov. 2019. Archived mailing list email.

[8] Perrig, A., Szalachowski, P., Reischuk, R. M., and Chuat, L. SCION: a secure Internet architecture. https://www.scion-architecture.net/pdf/SCION-book.pdf, 2017.

[9] Schinazi, D., and Pauly, T. Happy Eyeballs Version 2: Better Connectivity Using Concurrency. RFC 8305, Dec. 2017.

[10] Schmidt, P. S., Enghardt, T., Khalili, R., and Feldmann, A. Socket intents. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies* (12 2013).

[11] Seedorf, J., and Burger, E. Application-Layer Traffic Optimization (ALTO) Problem Statement. RFC 5693, Oct. 2009.

[12] Sternberg, D. curl vs wget, Mar. 2021.

[13] Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kühlewind, M., Perkins, C., Tiesel, P. S., Wood, C. A., Pauly, T., and Rose, K. An Abstract Application Layer Interface to Transport Services. Internet-Draft draft-ietf-taps-interface-12, Internet Engineering Task Force, Apr. 2021. Work in Progress.

[14] With, N., and Krüger, T. Implementation of a Transport-Agnostic, High-Level Networking API. https://code.ovgu.de/hausheer/taps-api, 2021.

[15] Yap, K.-K., Motiwala, M., Rahe, J., Padgett, S., Holliman, M., Baldus, G., Hines, M., Kim, T., Narayanan, A., Jain, A., et al. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), pp. 432–445.