NEAT – A New, Evolutive API and Transport-Layer Architecture for the Internet

Karl-Johan Grinnemo†, Tom Jones⋆, Gorry Fairhurst⋆, David Ros‡, Anna Brunstrom† and Per Hurtig†
†Karlstad University, Karlstad, Sweden
{karl-johan.grinnemo, anna.brunstrom, per.hurtig}@kau.se
⋆University of Aberdeen, Aberdeen, U.K.
{tom, gorry}@erg.abdn.ac.uk
‡Simula Research Laboratory, Oslo, Norway
dros@simula.no

***Abstract*—There is a growing concern that the Internet transport layer has become ossified in the face of emerging novel applications, and that further evolution has become very difficult. This paper identifies requirements for a new transport layer and then proposes a conceptual architecture, the NEAT system, that we believe is both flexible and evolvable. Applications interface the NEAT system through an enhanced user API that decouples them from the operation of the transport protocols and the network features being used. In particular, applications provide the NEAT system with information about their traffic requirements, prespecified policies, and measured network conditions. On the basis of this information, the NEAT system establishes and configures appropriate connections.**

***Index Terms*—Transport layer, ossification, application-aware networking, transport API, Internet architecture.**

I. Introduction

Introducing new transports in the Internet has become more or less impossible. This seems to be the experience for all new transport protocols that have been designed. Even protocols implemented in networking stacks, such as UDP-Lite [1], SCTP [2], and DCCP [3], have still failed to be widely used across the Internet. The most significant obstacle to evolution has been the deployment by operators of Network Address Translation (NAT) (strictly speaking NAT with port translation). These devices were originally introduced to address the growing shortage of IPv4 addresses, but are also perceived to have benefits in providing flexibility in the use of addresses and some element of security to devices behind the NAT. At the same time, the transport application programming interface (API) offered by operating systems has also become ossified and unable to adapt to the needs of novel applications. The ubiquity of TCP and UDP has meant that APIs typically tie applications to a priori choices of protocol. Or at the very least, application developers only invest in providing support to use these two protocols.

To address this issue, transport protocol designers have more recently proposed development of application-layer transport protocols that use the current underlying transport layer as a communication substrate, e.g., QUIC [4] that uses UDP as a substrate, and the Minion suite of transport protocols [5] that use TCP or UDP as substrates. Still, these protocols are in many ways only point solutions: QUIC primarily targets web traffic and Minion datagram services. SPUD [6] is a new initiative that also tries to use a UDP substrate to enable faster transport evolution.

This paper argues that there needs to be a more complete solution to the ossification problem. It proposes a design that can decouple applications from the choice of actual transport protocol being used, and enable applications to explicitly communicate their service requirements via a new transport API, with the transport protocol and options being selected at *run-time*. This change opens up for a more flexible transport solution that enables new network and transport functions and services to be added incrementally and transparently. Our proposal builds on previous work that has recognized the value of a higher layer transport interface (e.g., [7], [8]) where it can improve use of the network layer and take advantage of capabilities offered by different network paths. We extend this approach to a more complete solution that can enable transport protocol evolution across the Internet.

The remainder of the paper is organized as follows. Section II discusses the requirements that need to be fulfilled by a new transport system. A conceptual architecture addressing these issues is detailed in Section III. The paper concludes in Section IV with a brief discussion of how we intend to continue our efforts in designing and developing this system.

II. Transport Requirements

The requirements for a new transport system must be well designed to offer a remedy to the present ossification problem:

a) Deployable: A key requirement is that any new transport system is itself deployable. Its design must be independent of the operating system and particular network technologies. As an implication of this, it should not rely on protocols or features that are only available on certain host operating systems, or expect applications to include special mechanisms. Instead, these should be an integral part of the transport system itself.

b) Evolvable: A transport system must be evolvable and permit independent evolution of its components, allowing new mechanisms and protocols to be added as required, and for applications to benefit from new features as infrastructure evolves. We recognize that middleboxes (including home

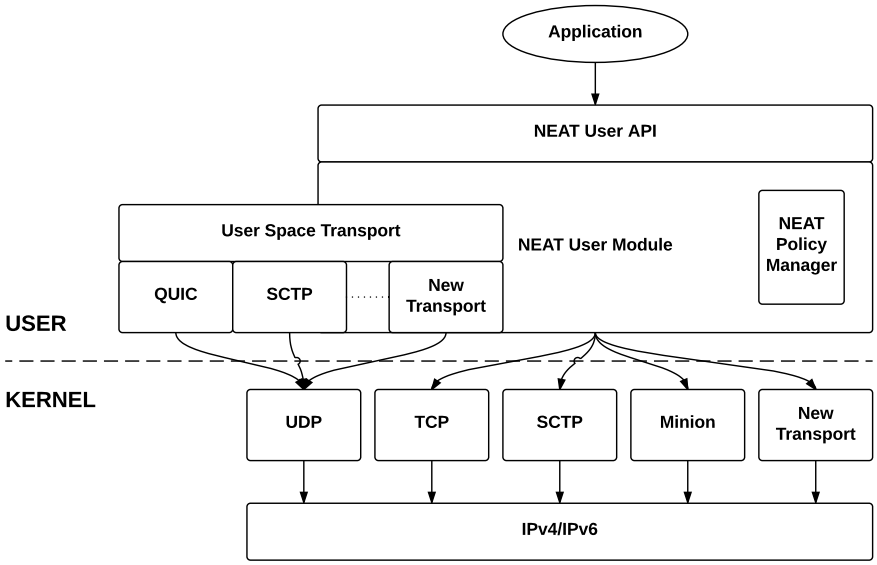

Fig. 1. The NEAT Architecture.

gateway NATs) have become an integral part of the Internet, and a transport system has to be able to discover whether any new solution works across a middlebox-encumbered path.

c) Present a new API: A new API should offer a higher abstraction than the present sockets API, ideally allowing applications to specify their *transport service* needs, rather than configuring a specific transport protocol instance. This allows the transport system to have the option of deciding which transport protocol to use and how it should be configured.

III. The NEAT System

To address the requirements imposed on a new transport system, this paper proposes a comprehensive *transport architecture*, a New, Evolutive API and Transport-layer architecture for the Internet (NEAT). An overview of the NEAT architecture is given in Fig. 1.

NEAT offers an enhanced API for applications to access *transport services*. The User Module is designed to be portable across different operating systems and network stacks. It comprises five groups of components (see Fig. 2): *Framework*, *Selection*, *Policy*, *Transport*, and *Signaling*.

NEAT includes a set of protocol mechanisms that takes care of middlebox traversal and protocol selection, accompanied with a fallback service when paths are incapable of supporting the chosen protocol. NEAT has an evolvable architecture that opens up for new transport services and can enable interaction with network devices to improve the transport service. NEAT also enables the incremental introduction of new transport protocols: both in the kernel and in user space.

The Framework components provide the functionality required to use the NEAT System. They define the structure of the User API and interfaces to the logic that implements the core mechanisms. Applications provide information about the requirements for a desired transport service via the API. The framework also includes components for diagnostic, debugging and measurement. The additional information reported by NEAT can be used to identify which components in NEAT are in use, and how components have been configured.

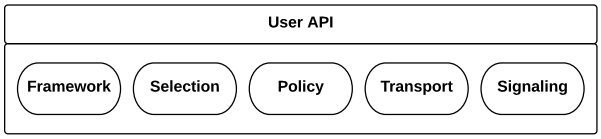

Fig. 2. The NEAT User Module.

The Selection components choose an appropriate transport endpoint and a set of candidate transport components. The additional API information enables the NEAT System to move beyond the constraints of the traditional socket API, making the stack aware of what is actually desired or required for each traffic flow. This is combined with inputs from a Policy Manager (see below). After identification of candidate services, it will test the suitability of the candidates, utilizing information known about the path, and by attempting to make endpoint connections.

The Policy components comprise the Policy Information Base (PIB), the Characteristics Information Base (CIB), and the Policy Manager. The PIB is a repository that contains a collection of policies, where each policy consists of a set of rules linking a set of matching requirements to a set of preferred or mandatory transport characteristics. Policies can be added by the system administrator, external entities or applications, and have different priorities. The CIB is a repository storing information about available interfaces, supported protocols, network properties and current/previous connections between endpoints.

The Policy Manager enables a set of rules to specify the

transport protocol to use for a particular transport service, the configuration of the selected transport protocol, etc., implemented in the PIB. In contrast, acquired knowledge about endpoints and the network, evolves over time, and is stored in the CIB. This enables the Policy Manager to cache characteristics allowing future flows to benefit from previous flow experience.

Together, Selection and Policy enable applications to be designed and implemented to be oblivious to the transport protocols of a particular platform. They facilitate NEAT to make an appropriate decision based on application requirements and what is made available for a network endpoint. Making these decisions at run time rather than at application-design time, ensures that an appropriate choice is made, provides opportunities to consider multiple, possibly conflicting, constraints, and avoids each application having to code for the possibility that a path does not support a particular mechanism or combination of mechanisms.

The Transport components are responsible for providing functions to instantiate the transport service for a particular flow in NEAT. Transport provides a set of transport protocols, e.g., TCP, UDP, SCTP, TLS, DTLS, etc., and other components to realize a transport service. While the selection of transport protocols are handled by the Selection components, the Transport components are responsible for configuring and managing the transport protocols.

The Signaling components can provide advisory signaling to complement the functions of Transport. This could include communication with middleboxes, support for failover and handover and other mechanisms. There are two types of signaling: transport and network. Transport signaling enables the exchange of capability information between NEAT endpoints, and provides input to the Policy Manager concerning what capabilities a peer endpoint supports. Network signaling communicates with devices along the network path. This could be as simple as the distribution of Differentiated Services Code Points, network signaling messages that can assist in applications or transports to choose to up-speed or down-speed for transports, through to something as complex as QoS negotiation.

Placing optional network signaling below the transport API can allow applications to indicate how/whether they would like to use network signaling, without requiring each application to be updated each time a new signaling protocol is introduced. The NEAT system can determine the appropriate signaling mechanisms to use on a path to a particular endpoint. Freeing applications from choosing and supporting signaling protocols is expected to reduce the barriers to introducing new mechanisms in the network, allowing signaling messages to be exchanged with network devices as such new signaling protocols emerge and become supported across the network.

Just as a higher-level transport API decouples transport and interface selection (e.g., to offer potential for services to evolve independently of applications), so also there can be benefits in the way that transports interact with the network. For example, when UDP is used to encapsulate a transport that does not have widespread network support (e.g., SCTP, or any new transport), the NEAT system can select one of a number of signaling protocols (e.g., Port Control Protocol (PCP) [9]) to coordinate with local mappings in a firewall/NAT ensuring that the local network device keeps state while a transport protocol connection remains active. Similarly, a future transport protocol could benefit from signaling received about a path's congestion, disruption, or some other characteristic—all of which can be interpreted within the context of the transport connection. The higher abstraction of the transport API therefore not only enables flexibility, it can provide a path to future evolution of the transport layer.

IV. Conclusion

This paper has examined the requirements for a new transport system and proposed a conceptual architecture, NEAT, which we think can break the current ossification of the Internet transport architecture. In this architecture, applications will interface through an enhanced API that decouples them from the choice of transport protocol and network features to be used. We are currently implementing the main parts of the NEAT system, and use this to explore performance in actual networks. The source code for our NEAT system is available on GitHub [10].

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT). The views expressed are solely those of the authors.

References

[1] L.-A. Larzon, M. Degermark, S. Pink, L.-E. Jonsson, and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)," RFC 3828 (Proposed Standard), Internet Engineering Task Force, Jul. 2004, Updated by RFC 6335. [Online]. Available: http://www.ietf.org/rfc/rfc3828.txt

[2] R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Proposed Standard), Internet Engineering Task Force, Sep. 2007, Updated by RFCs 6096, 6335, 7053. [Online]. Available: http://www.ietf.org/rfc/rfc4960.txt

[3] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340 (Proposed Standard), Internet Engineering Task Force, Mar. 2006, Updated by RFCs 5595, 5596, 6335, 6773. [Online]. Available: http://www.ietf.org/rfc/rfc4340.txt

[4] J. Roskind, "Quick UDP Internet Connections," Google, Tech. Rep., Apr. 2012.

[5] M. F. Nolan, N. Tiwari, J. Iyengar, S. O. Amin, and B. Ford, "Fitting Square Pegs Through Round Pipes: Unordered Delivery Wire-Compatible with TCP and TLS," in *USENIX NSDI*, San Jose, CA, U.S., Apr. 2012.

[6] M. Kuehlewind and B. Trammell, "SPUD Use Cases," Jul. 2015.

[7] B. D. Higgins, A. Reda, T. Alperovich, and J. Flinn, "Intentional Networking: Opportunistic Exploitation of Mobile Network Diversity," in *The* 16^{th} *Annual International Conference on Mobile Computing and Networking*, Chicago, IL, U.S., Sep. 2010.

[8] P. S. Schmidt, T. Enghardt, R. Khalili, and A. Feldmann, "Socket Intents: Leveraging Application Awareness for Multi-Access Connectivity," in *ACM CoNEXT*, Santa Barbara, CA, U.S., Dec. 2013.

[9] D. Wing, S. Cheshire, M. Boucadair, R. Penno, and P. Selkirk, "Port Control Protocol (PCP)," RFC 6887 (Proposed Standard), Internet Engineering Task Force, Apr. 2013, Updated by RFCs 7488, 7652. [Online]. Available: http://www.ietf.org/rfc/rfc6887.txt

[10] NEAT, "NEAT Project," https://github.com/NEAT-project, 2016.