PASTE: Protocol-agnostic Services & Transport Enrichment with intermediate Companion nodes

Helge Backhaus
Institut für Telematik, KIT, Germany
backhaus@kit.edu

***Abstract*—The current socket API is one obstacle that hinders adoption of new transport protocols like e.g. SCTP, MPTCP, or DCCP. It requires an application to specify the desired transport protocol at the time of implementation. The drawback of this approach is: If new transport protocols are introduced, existing applications need to be modified in order to support them. Therefore applications should select abstract transport services rather than specific protocols. We propose Protocol-agnostic Services & Transport Enrichment (PASTE): An application-to-network middle-ware that pushes the protocol selection below the application layer based on the selected transport service. It also handles other networking functionality commonly done by applications, e.g., name-to-address resolution. To further enhance the offered transport services we also introduce intermediate Companion nodes within the network, to enrich different transport services with added functionality like, e.g., in-network data buffering or on-the-fly data compression.**

I. Introduction

Today several new transport protocols like, e.g., SCTP, MPTCP, or DCCP exist, offering applications a multitude of transport services beyond those provided by TCP or UDP. Despite being available on different platforms and operating systems, none of these protocols has reached wide spread adoption thus far. This is among other reasons due to today's socket API, which requires applications to select a specific transport protocol at design time. Thus applications need to be modified in order to benefit from newer transport protocols. To improve this situation, we propose a Protocol-agnostic Services & Transport Enrichment (PASTE) Layer. Figure 1 depicts a general PASTE-enabled network. Applications access

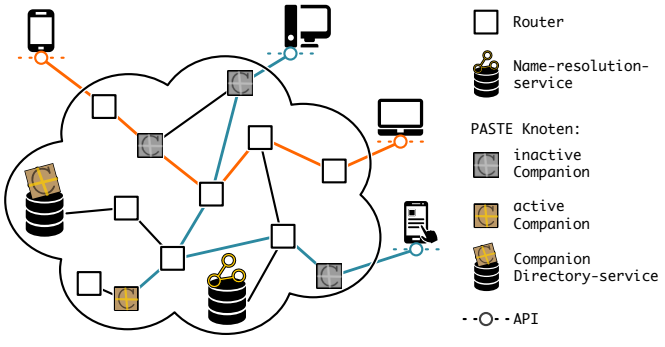

Figure 1. PASTE-enabled network

the network via a name-based API, which is protocol independent. Instead of specific protocols, applications are offered different transport services based on transport characteristics, which can be provided by applications. Names are used by application instances to establish communication associations among each other. Name-resolution is handled by PASTE via a name resolution service. This enables the transparent usage of different addressing schemes like, e.g., IPv4 and IPv6 today.

We also introduce intermediate nodes in the network, to further enrich the offered transport services with added functionality. These nodes, called Companions, are managed by PASTE and can be inserted between two communicating application instances, transparently for the application.

Companions can aid applications in achieving common data transport tasks easier, e.g., by storing data in the network upon encountering bandwidth bottlenecks on any side of a connection, or by doing on-the-fly data compression. Thus alleviating applications from buffering and re-transmitting data, making their implementation easier. This can also enable applications to better utilize available network resources by, e.g., achieving a faster data upload than the receiving node supports, a sender can complete a big data transfer faster. Any node within the network can function as a Companion, by running a PASTE instance. Companions register themselves and their offered services at a Companion directory service, which other PASTE node query for appropriate Companions on demand at run time.

II. Related Work

The following work is partially based on and build upon other related work in this area.

A. Name-based sockets

Name-based sockets aimed at redesigning the socket API to achieve a better decoupling of applications and network protocols [1]. It does name-to-address resolution, and transport selection based on proposed service names and port numbers for services. It supports UDP, TCP, DCCP, and SCTP for transports and relieves applications from implementing features such as multi-homing, mobility, or NAT traversal. Applications still have to choose the desired service and thus protocol at design-time though, which prevents them from employing the different protocols in different environments.

B. Protocol-Independent Internet Transport API

General transport services such as *reliable transport* or *in-order delivery* were extracted in [2] by analyzing UDP, TCP,

SCTP, and DCCP and their offered services. From that follows an extended socket interface to make those general transport services configurable via socket options. There is an ongoing attempt to form an IETF TAPS Working Group, to continue this work, with the desired result of an list of abstract transport services and transport services characteristics, provided by today's available transport and congestion control protocols.

C. A future-proof application-to-network interface

In [3] we proposed a new name-based application-to-network interface as part of the NENA Framework [4]. It aims to reduce the necessary networking know-how at application-level to foster independent evolutions of applications and the network stack. To achieve this goal, networking functionality currently done by the applications themselves is pushed down below the API. The proposed interface aims at being simple and intuitive for the application programmer, separating application and networking concerns.

III. PASTE

Figure 2 shows the different parts and overall structure of the PASTE Layer, which is located between the transport and application layer. It manages connections via available transport protocols, e.g., TCP, UDP, SCTP, etc. These are established between PASTE instances and exposed to applications as communication associations.

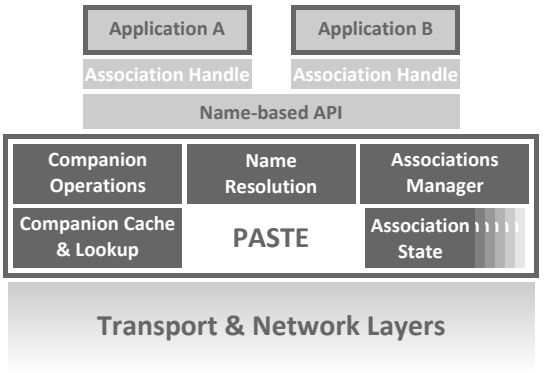

Figure 2. Protocol-agnostic Services & Transport Enrichment Layer structure

A. Name Resolution

At the application level no network specific addresses or locators are used. Applications only have to deal with URI-based names to identify application instances and specify how they can be reached. The actual name resolution is done in PASTE.

B. Companion Operations

We refer to intermediate nodes involved in the fulfillment of a data transport from one application instance to another as Companions. Every node running the PASTE layer can serve as a Companion. It is easily conceivable that routers at home, user operated dedicated servers, or servers provided by an ISP or another third party fulfill this task. For now we just assume that they are placed at strategic points in the network and highly available. There exist many possible usage scenarios for Companions, like acting as ad-hoc message brokers, or aiding with mobility.

PASTE's Companion Operations part listens for incoming Companion requests for all available transport protocols and keeps track of resources for different offered Companion functions. For instance available RAM or disk space for caching and storing data, as well as available processing power for on-the-fly data compression. Companions may offer one or more of these services by registering them at a Companion directory, so that end-systems running PASTE can find and use appropriate Companions for different transport services. Two simple scenarios, where Companions are used to compensate for bandwidth bottlenecks or short outages of end-systems in the network, are detailed in section IV.

C. Associations Manager & Association States

Within PASTE and at the application layer all data transports are represented as communication associations. These are similar to connection handles from the socket API and encapsulate all the state associated with a data transport, while keeping track of individual transport protocol connection states on all systems involved. Applications request an association via the name-based API and receive an association handle to communicate via its provided primitives. Association states are created in the PASTE layer for each requested association and managed by an associations manager. They consist of: An association-ID, the local name of the initiator or source, the initial name of the destination, the name of the current destination, e.g., some Companion, all names of previously involved nodes, and the current connectivity status. The connectivity status can be: Active – Currently data is transmitted via this association. Suspended – No data is send or received on this side, but there is still pending data for later. Unconfirmed – No more data is send or expected on this side and the association is regarded as half closed but there may still be pending acknowledgments. Lost - The associations status is unknown locally and none of the known participating nodes has any information regarding the association-ID. This may happen, if some kind of unrecoverable error occurs due to the failure of a participating node.

D. Companions Lookup and Cache

Initially a Companion directory needs to be queried to find Companions offering a certain functionality, each time it is needed by a service. Known Companions are cached afterwards and can also be exchanged with other end-systems after that, so that Companions can also be obtained from communication partners. Therefore a unique Companion ID is stored, with all known connection information like: The name where a Companion is reachable. Available transport protocols, offered functions, and parameters like previously experienced up- or down-link bandwidth, and encountered latency. These parameters can aide in finding a better Companion of a few available ones for a certain task.

E. Name-based API

The name-based API exposes transport services instead of specific transport protocols to applications. In [5] we proposed a method how protocol or rather service characteristics can be described. Supported service characteristics can include sequence preservation, varying degrees of reliability, or offer low delay or high throughput. Alongside any connection requests application instances pass desired service characteristics. Based on the supplied characteristics a specific transport protocol is chosen. Protocol independent sequence numbers are added to application data units (ADUs), which are kept intact. PASTE employs its own framing, in order to distinguish between ADUs and signaling messages related to, e.g., Companion handovers or acknowledgments. Therefore most of the signaling can be done in-band over existing transport protocol connections between application instances.

IV. Companion Use-case

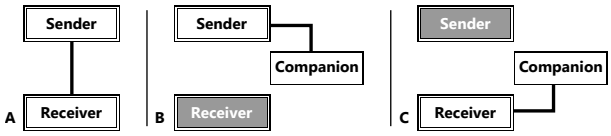

Figure 3. A simple file transfer involving three nodes.

The basic use-case involves three nodes, depicted in Figure 3. (A) A receiver binds itself to a known name. A sender connects to the receiver requesting a fully reliable, sequence preserving, error-checked transport service. Since only TCP is available between the sender and receiver a TCP connection is established by PASTE. An association state for that connection is created, and an association handle returned to the application. The sender then starts a big file transfer. We assume the senders up-link is about an order of magnitude higher than the receivers down-link. (B) The receiver wants to go offline for some time. It might be a smartphone leaving a wireless lan. Thus the association is suspended. The receiver's PASTE layer signals this via the TCP connection to the sender's PASTE layer and its association state switches to `suspended`. Afterwards the TCP connection is closed. The sender's PASTE layer request a Companion, supporting data caching, from the Companion directory service. Since SCTP is available between the sender and returned Companion, the sender's PASTE layer establishes an SCTP connection and sends an updated association state to the Companion. Data meanwhile received from the application is cached by the PASTE layer and now transmitted to the Companion. Afterwards the sender closes the association, leaving the association state in the `unconfirmed` status, while the Companion's association state switches to `suspended`. The sender may go offline now. (C) The receiver goes back online and the file transfer application has a cached association ID, which it uses to reanimate the `suspended` association. The last known source is the sender. There are several possibilities now. The sender may be online, its PASTE layer gets contacted by the receiver's and the receiver obtains an updated association state and thus the Companion information. If the sender is offline either the receiver's PASTE layer may ping the sender, or the Companion's PASTE layer may ping the receiver. Once the receiver's PASTE layer knows about the Companion a TCP connection is established and the remaining data is transmitted. Depending on how the receiver and Companion found each other, different `unconfirmed` association states need to get cleaned up. If the receiver stays online long enough, after suspending the association, its PASTE layer may even receive the Companion information before tearing down the initial TCP connection, thus freeing it from directly contacting the sender again before resuming the download. Other possible scenarios involve the sender requesting a Companion, without the receiver going offline, just to better utilize its bigger up-link to finish the file transfer earlier. Assuming the receiver is a smartphone, a Companion supporting on-the-fly data compression instead of caching may be used in that case, provided that the receiver's PASTE layer signals that it has enough processing power available to decompress the data locally.

V. Conclusion

This paper proposed application-to-network interface changes and introduced PASTE, which allows to reduce the applications' required knowledge about specific networking and transport protocol details. Applications simply state desired communication characteristics alongside connection requests instead of directly selecting transport protocols at design time. Network functionality like name-to-address resolution and protocol selection are pushed down below the current socket API, which fosters the introduction of new transport protocols and services in the Internet without the need to modify existing applications – and thus fosters innovations at the network-level as well as at the application-level. In addition, the PASTE layer located underneath the API, adds intermediate systems – called Companions – on end to end paths, aiding applications to achieve common data transport tasks easier, by alleviating the need to implement common tasks in the application, while at the same time enabling applications to better utilize the locally available network resources.

References

[1] J. Ubillos, M. Xu, Z. Ming, and C. Vogt, "Name-Based Sockets Architecture," Internet Draft (draft-ubillos-name-based-sockets-03), Mar. 2010.

[2] M. Welzl, S. Jörer, and S. Gjessing, "Towards a Protocol-Independent Internet Transport API," in *Fourth International Workshop on the Network of the Future (FutureNet IV)*, 2011.

[3] D. Martin, H. Wippel, and H. Backhaus, "A Future-Proof Application-to-Network Interface," in *Proceedings of the 2011 Second International Conference on the Network of the Future (NoF 2011)*. IEEE, Nov. 2011.

[4] D. Martin *et al.*, "Netlet-based Node Architecture Project Homepage." [Online]. Available: http://nena.intend-net.org/

[5] H. Backhaus, "Towards a Property and Requirement-based Application Interface for Future Networks," in *10th Würzburg Workshop on IP: Joint ITG, ITC, and Euro-NF Workshop "Visions of Future Generation Networks" (EuroView2010)*, Würzburg, Germany, Aug. 2010.