Cross-Layer Coordination: Let's Talk About Intentions

Philipp S. Schmidt
philipp@inet.tu-berlin.de
TU Berlin

Theresa Enghardt
theresa@inet.tu-berlin.de
TU Berlin

ABSTRACT

To evolve a network beyond a stupid bitpipe, the network needs information about the applications' intentions and communication preferences. We give an brief overview about our earlier work, *Socket Intents*, that augments the standard BSD socket interface to enable the application to express what it *knows* about its communication patterns and preferences. This information can then be used by our proactive policies to choose the appropriate interface, tune the network parameters, or even combine multiple interfaces. We also give an outlook how similar mechanism can be used in other parts of the network to exploit synergies between the network layers without violating the layering.

1. INTRODUCTION

The end to end argument [5] and the hourglass shape of its protocol hierarchy are, from an architectural perspective, definitely among the most important reasons for the success of the Internet. This architecture allowed for easy adoption of new network protocols on top of IP. However, the introduction of middleboxes that do in-network modification of traffic for performance and security reasons has mostly ended this. New transport and sometimes also application layer protocols become hard to deploy in a network that filters or modifies traffic it does not know. When thinking about the next steps in the IP Stack evolution, we have to ask why we can't have both: In-network performance optimization and security checks through middle boxes and application evolvability?

In this paper, we propose an easy solution for this complicated problem: Let the network elements talk to each other about their intentions and preferences, even across layer boundaries. When signaling what in-network modifications to their communication applications request or tolerate, it is possible to take advantage of the benefits without disturbing new protocols.

The remainder of the paper is structured as follows: We first introduce our earlier work, Socket Intents[6], that augments the standard BSD socket interface to enable the application to express what it knows about its communication patterns and preferences, as a working example of cross-layer coordination. We then present a few examples that can benefit from similar approaches and finally conclude.

2. SOCKET INTENTS

Socket Intents[6] are a proactive, application informed approach that augments the socket interface to enable the application to express its communication preferences. Here, preferences can refer to high bandwidth, low delay, connection resilience, etc. This information can then be used by our dynamic proactive policies to choose the appropriate interface, tune the network parameters, or even combine multiple interfaces.

Intent	Type	Value
Category	Enum	Query, bulk transfer, control traffic, stream
File size	Int	Number of bytes transferred by the application
Duration	Int	Time between first and last packet in seconds
Bitrate	Int	Size divided by duration in bytes per second
Burstiness	Enum	Random bursts, regular bursts, no bursts or bulk (congestion window limited)
Timeliness	Enum	Stream (low delay, low jitter), interactive (low delay), transfer (completes eventually) or background traffic (only loose time constraint)
Resilience	Enum	Sensitive to connection loss, undesirable (loss can be handled) or resilient (loss is tolerable)

Table 1: List of proposed Socket Intents.

The principle observation that the application has critical information is not novel, but has been made before in the context of Quality of Service. The main difference between QoS and our Socket Intents approach is that "(...) the application tells what it *knows* as opposed to what it *wants*, as in prior work on QoS". [1] Therefore, the application expresses what it knows about the communication, what the traffic might look like and what the application can tolerate.

In summary, our Socket Intents of (i) an augmented socket library to communicate the needs of the application, (ii) a set of policies to select or combine appropriate network interfaces, and (iii) mechanisms to combine or select interfaces.

Principle

Let's start with some examples: (i) If the antivirus software needs an update, this usually implies downloading a large file in a bulk transfer, maybe with a known file size. Timing is typically noncritical and the data can be downloaded as background traffic. If the TCP connection was closed during the transfer, it would not hurt, as the download can be continued. Here the application can set the general category bulk transfer, and provide additional information such as file size, timeliness, and resilience.

(ii) If you want to stream a video, the streaming application may know the bitrate and the duration. It does not want the TCP connection to be disconnected because this may be visible to the user. Here it can specify the general category stream transfer with additional information about duration, bitrate and resilience.

Based on this philosophy, we propose a set of Socket Intent options, see Table 1. Socket Intents are optional in the sense that

[1] As smartly summarized by the CoNEXT'13 TPC

they are not required but any number of them can be specified. They are structured hierarchically, starting with the "category" option with possible values of query, bulk transfer, control traffic, and stream, which are realized as enum. Then we have more specific options which include file size, duration of the flow, expected bitrate, whether the traffic is bursty, whether the timeliness of the flow completion matters and how resilient the application is against connection loss. Each of these can either be enums or integers. Note that these are extensible and Table 1 is only a first proposal.

System architecture

In the following we discuss where in the stack Intents should be specified to enable the components to cooperate.

Figure 1(a) shows how communication works within today's OSes.

The application request a socket, either TCP or UDP, via the Socket API, today's almost universal interface. The OS then maps the socket to a specific network interface using a fixed internal policy, which gets minimal information from the application, and then uses it for the communication.

To benefit from the information about the upcoming communication that is available within the application, the programmer has to pass it along with the request for communication.

Accordingly, we augment the Socket library of the Socket API with additional *socket options* and add an explicit user space policy manager, Figure 1(b). Upon (1) recognizing a Socket Intent option, the Socket library (2) calls the user space policy manager which can then (3) choose either a single interface or multiple interfaces. This is then used to (4) override the kernel interface selection policy.

Policies

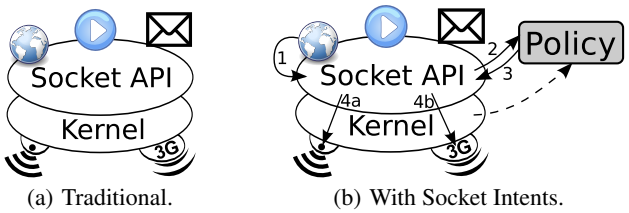
Our system design places the hardest problem, the actual decision which interface(s) to use for which communication, into the policy component. It is important to note that the specific policy that is most beneficial will likely depend on the configuration of the host, the current location, the current network availability, etc.

A rather obvious example for a policy is to use high bandwidth interfaces for application requests of the category bulk transfer and low latency interfaces for application requests of the category query. With Socket Intents, we enable the policy to decide what to optimize for, which would be undecidable otherwise.

Policies are not limited to the use of a single interface if the transport protocol supports the simultaneous use of multiple interfaces. The polices in our current prototype use socket options to control use of MPTCP [3] or multiple paths in SCTP to allow for transport layer innovation without having to change the application.

3. CROSS LAYER COORDINATION

The idea of cross layer coordination is not limited to the Socket Interface. In this section, we give another examples how cross-layer coordination already works on the internet and an interesting candidate of new cross layer coordination mechanisms.



(a) Traditional. (b) With Socket Intents.

Figure 1: Within host: Applications/network interfaces.

Packet Chunking

Chunking refers to splitting an object, a bitstream, or a sets of packets into one or more parts. This process is essential to transmit a byte stream or packet given to the socket using the transport- and network layer through the internet. For chunking, the evolution of the internet has come to a rough consensus how to solve it: Use the largest possible chunks within the application and let the TCP stack of the end hosts find out the largest chunk size that traverse the whole network without re-chunking. To take advantage of this separation, the only cross layer coordination needed is that each layer is aware of the MTU presented by the layer below and a mechanism like *path MTU discovery* allowing the end hosts to adapt their chunking accordingly.

Access Network Load Indication

How to handle hosts and/or networks with *multiple network connections* is a buzzing as well as long standing research problem. Today, the focus is how to seamlessly take advantage of multiple connections within the protocols. Even if all systems have different objections, e.g. to shift traffic from from cellular networks [2], shift traffic to cellular networks [4] or to take advantage of the union bandwidth of multiple access links via Multi-Path TCP [3], or by distributing IP flows [1], all have a common problem, to determine whether the network has the necessary resources for the traffic.

While loss based mechanisms like used in TCP work fine, they are hard to influence from a network management standpoint. QoS with reservation is no alternative, as it has a large signaling overhead and has only been deployed in small parts of the network.

We propose Advisable Access Network Load Indication as an addition, that allows the network to indicate its load to clients before congestion happens. With such an indication, transport layer protocols like MPTCP can decide whether to add a subflow using this network and even applications can decide to postpone actions to take load from the network.

4. CONCLUSION

As demonstrated with Socket Intents, cross layer coordination has a high potential to become one of the key ideas to evolve the IP stack. Where already taking place, like in the case of chunking, it enables coordination of several layers without violating layer boundaries. We think that cross layer coordination can have similar benefits in other areas, like Access Network Load Indication and the signaling whether in-network modification and optimization is advisable, and therefore can also be a tool to solve the conflict between in-network modification and application evolvability.

5. REFERENCES

[1] A. De La Oliva, C. Bernardos, M. Calderon, T. Melia, and J. Zuniga. Ip flow mobility: smart traffic offload for future wireless networks. *IEEE Communication Magazine*, Oct 2011.

[2] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong. Mobile data offloading: how much can wifi deliver? In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 425–426, New York, NY, USA, 2010. ACM.

[3] C. Paasch and O. Bonaventure. Multipath tcp. *Queue*, 12(2), Feb 2014.

[4] C. Rossi, N. Vallina-Rodriguez, V. Erramilli, Y. Grunenberger, L. Gyarmati, N. Laoutaris, R. Stanojevic, K. Papagiannaki, and P. Rodriguez. 3gol: Power-boosting adsl using 3g

onloading. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 187–198, New York, NY, USA, 2013. ACM.

[5] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4), 1984.

[6] P. S. Schmidt, T. Enghardt, R. Khalili, and A. Feldmann. Socket intents: Leveraging application awareness for multi-access connectivity. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 295–300, New York, NY, USA, 2013. ACM.