TAPS Working Group
Internet-Draft
Intended status: Experimental
Expires: April 30, 2018

P. Tiesel
T. Enghardt
A. Feldmann
TU Berlin
October 27, 2017

Socket Intents
draft-tiesel-taps-socketintents-01

Abstract

This document outlines Socket Intents, a concept that allows applications to share their knowledge about upcoming communication and express their performance preferences in a generic, intuitive and, portable way. Using Socket Intents, an application can express what it knows, assumes, expects, or wants regarding its network communication. The information provided by Socket Intents can be used by the network stack to optimize communication in a best-effort way.

Socket Intent can be used to stem against the complexity of exploiting transport diversity, e.g., to automate the choice among multiple paths, provisioning domains or protocols. By shifting this complexity from the application developer to the operating system, it enables the use of these transport features to a wider range of applications.

Table of Contents

1. Conventions and Definitions

The words "MUST", "MUST NOT", "SHALL", "SHALL NOT", "SHOULD", and "MAY" are used in this document. It's not shouting; when these words are capitalized, they have a special meaning as defined in [RFC2119].

Association Set, Association, Stream, or Message are used as defined in [I-D.tiesel-taps-communitgrany].

2. Introduction

Despite recent advances in the transport area, the adaption of new transport protocols and transport protocol features is slow. In practice, this only happens in limited fields as Web browsers or within datacenters. The same problem occurs for taking advantage of paths or provisioning domains (PvDs). In both cases, the benefits of the new transport diversity come at the cost of an increased complexity that has to be mastered by the application programmer.

To enable transport features like TCP fast open [RFC7413] or to control how MPTCP [RFC6824] creates subflows requires specialized APIs. These APIs are not part of the standard socket API, usually not portable, and not available in many programming languages. Using them often requires profound knowledge of the transport protocol internals.

To use multiple paths, applications usually have to use their own heuristics to select which paths, provisioning domains, or access network to use. Choosing the right path is difficult as their characteristics differ, e.g., regarding performance. Obtaining the necessary information is difficult since it may require special privileges and non-portable APIs.

In all cases mentioned above, an application that wants to take advantage of the available transport diversity is faced with substantially higher complexity regarding network APIs and networking code.

3. Problem Statement

Application programmers opening a communication channel typically know how this channel will be used. There is more information available than the protocol and destination address needed to establish a communication channel: An application developer has an intuition about many aspects of an upcoming communication. These intuition may include:

preferences: whether to optimize for bandwidth, latency, or cost

characteristics: expected packet rates, byte rates or how many bytes will be sent or received.

expectations: towards path availability or packet loss

resiliences: whether the application can gracefully handle certain error cases

These preferences, expectations and other information known about the upcoming communication should be expressible in an intuitive, generic way, that is independent of the network and transport protocol. Its representation should be independent of the actual API used for network communication and should be expressible in whatever API available, e.g., as socket options for BSD sockets or as part of the address resolution configuration for Post Sockets [I-D.trammell-taps-post-sockets].

Socket Intents should enable the OS to adjust the communication channel according to the application's intents in a best-effort fashion: They should provide the information needed to automatically enabling transport features the application can benefit from or help choosing the most suitable (combination) of paths based on the properties of the access networks or PvD (see [RFC7556], Section 6.2) available. The actual implementation is not part of the Socket Intents concept, it is left to an OS policy that may choose the best transport protocol, default parameters and PvDs available and may also try to further optimize wherever possible.

4. Socket Intents Concept

Socket Intents are pieces of information that allow an application to express what they know about the application's communication. They indicate what the application wants to achieve, knows, or assumes in general, intuitive terms. An application can use them to annotate the characteristics, preferences, and intentions it associates with each communication unit. Depending on the API used, Socket Intents can be used on a per Association Set, Association, Stream or, Message level.

Socket Intents are optional information that can be considered in a _best-effort_ manner. Socket Intents _do not include requirements_, such as reliable in-order delivery. Typical examples include desired transport characteristics, e.g., low delay, high throughput, or minimal cost, as well as expected application behavior, e.g., will send 500 bytes. As this information captures the intents of an

applications and passes them along with the communication socket, we call these pieces of information Socket Intents.

Applications have an incentive to specify their intents as accurately as possible to take advantage of the most suitable existing resources. Applications are expected to selfishly specify their preferences. It is up to the OS's policy to prevent commitment of excessive resources.

4.1. Interactions between Socket Intents and QoS

Socket Intents are not QoS labels, but have an orthogonal meaning. While the purpose of QoS is to specify what an application requires, Socket Intents are used to specify what an application knows or prefers. Therefore,

- Socket Intents SHALL be purely advisory.
- Socket Intents MUST NOT be used to derive IntServ / RSVP style guarantees.
- Socket Intents SHOULD be taken into account on a best-effort basis and MAY be used to derive DiffServ Service Classes as described in [RFC4594].

5. Socket Intent Types

Socket Intents are structured as key-value-pairs.

The key, called short name, specifies the Socket Intent type. It is identified by a string of the lower-case characters [a-z], numbers [0-9] and the separator "-".

The namespace for the short names is partitioned as follows:

- All Socket Intent type not starting with "x-" or "y-" are managed by an IANA registry. The assignment of new types requires an RFC or expert review (TO BE DECIDED).
- Socket Intent type starting with "x-" are for experimental use.
- Private or vendor specific Socket Intent type MUST start with "y-[vendor]-".

Values can be represented as Enum, Int, Float, ASCII-String [RFC0020] or a sequence of the aforementioned data types. Implementations determine how these types are represented on the respective platform.

The data type for the individual Socket Intents are determined by the document defining the Socket Intent and MUST NOT be changed by an implementation. For Enum data types, a list of valid values MUST be provided by the document specifying that intent as well as a default value that is equivalent to not specifying this intent.

6. Initial Socket Intent Types

The following sections contain a list or Socket Intent types and their possible values. Recommended default values for Enum values are marked with an asterisk (*) behind the level name.

6.1. Traffic Category

The Traffic Category describes the dominating traffic pattern of the respective communication unit expected by the application.

Short name: category

Applicability: Association Set, Association, Stream

Data type: Enum

Level	Description
query	Single request / response style workload, latency bound
control	Long lasting low bandwidth control channel, not bandwidth bound
stream	Stream of bytes/messages with steady data rate
bulk	Bulk transfer of large messages, presumably bandwidth bound
mixed*	Don’t know or none of the above

Note: Most categories suggest the use of other intents to further describe the traffic pattern anticipated, e.g., the bulk category suggesting the use of the Size to be Sent intent or the stream category suggesting the Stream Bitrate and Duration intents.

6.2. Size to be Sent / Received

This Intent is used to communicate the expected size of a transfer.

Short name: send_size / recv_size

Applicability: Association Set, Association, Stream, Message

Data type: Int (bytes)

6.3. Duration

This Intent is used to communicate the expected lifetime of the respective communication unit.

Short name: duration

Applicability: Association Set, Association, Stream

Data type: Int (msec)

6.4. Stream Bitrate Sent / Received

This Intent is used to communicate the bitrate of the respective communication unit.

Short name: send_bitrate / recv_bitrate

Applicability: Association Set, Association, Stream

Data type: Int (bits/sec)

6.5. Burstiness

This Intent describes the anticipated burst characteristics of the traffic for this communication unit. It expresses how the traffic sent by the application is expected to vary over time, and, consequently, how long sequences of consecutively sent packets will be. Note that the actual burst characteristics of the traffic at the receiver side will depend on the network.

This Intent can provide hints to the application on what the resource usage pattern for this communication unit will look like, which can be useful for balancing the requirements of different application.

Short name: bursts

Applicability: Association Set, Association, Stream

Data type: Enum

Level	Description
no_bursts	Application sends traffic at a constant rate
regular_bursts	Application sends bursts of traffic periodically
random_bursts	Application sends bursts of traffic irregularly
bulk	Application sends a bulk of traffic
mixed*	Don’t know or none of the above

6.6. Timeliness

This Intent describes the desired delay characteristics for this communication unit. It provides hints for the OS whether to optimize for low delay or for other criteria. There are no hard requirements or implied guarantees on whether these requirements can actually be satisfied.

Short name: timeliness

Applicability: Association Set, Association, Stream, Message

Data type: Enum

Level	Description
stream	Delay and packet delay variation should be kept as low as possible
interactive	Delay should be kept as low as possible, but some variation is tolerable
transfer*	Delay and packet delay variation should be reasonable, but are not critical
background	Delay and packet delay variation is no concern

6.7. Disruption Resilience

This Intent describes how an application deals with disruption of its communication, e.g. connection loss. It communicates how well the application can recover from such disturbance and can have implications on how many resources the OS should allocate to failover techniques for this particular communication unit.

Short name: resilience

Applicability: Association Set, Association, Stream, Message

Data type: Enum

Level	Description
sensitive	Disruptions result in application failure, disrupting user experience
recoverable*	Disruptions are inconvenient for the application, but can be recovered from
resilient	Disruptions have minimal impact for the application

6.8. Cost Preferences

This describes the Intents of an Application towards costs cased by the respective communication unit. It should guide the OS how to handle cost vs. performance and reliability tradeoffs.

Short name: cost

Applicability: Association Set, Association, Stream, Message

Data type: Enum

Level	Description
no_expense	Avoid expensive transports and consider failing otherwise
optimize_cost	Prefer inexpensive transports and accept service degradation
balance_cost*	Do not bias balancing cost and other criteria
ignore_cost	Ignore cost, choose transport solely based on other criteria

Note: the "no_expense" level implicitly asks the OS to fail communication attempts if no inexpensive transports are available.

Application developers MUST be aware that this also no hard requirement and can be ignored or overridden by the OS policy.

7. Implementation Guidelines

Implementations faced with unknown Socket Intent types SHOULD ignore these intents for forward compatibility. The API MAY include a parameter to change this behavior and make specifying unknown Socket Intent types return an error.

Invalid values SHOULD return an error to the application.

For debugging purposes, implementations SHOULD allow to enumerate the Socket Intents that are understood by the implementation. They MAY expose which of the Socket Intents were considered by the implementation.

8. Security Considerations

8.1. Performance Degradation Attacks

We assume that applications specify their preferences in a selfish, but not malicious way and that it is up to the OS to find a compromise between demands.

A malicious application could confuse the OS in a way that leads to scheduling traffic with certain Intents on a more expensive interface, penalizing this traffic, or even rejecting it. The attack vector added by this is negligible: As the malicious application

could also generate the traffic it claims to intend, it already has a much more powerful attack vector.

As a mitigation, the OS could monitor and compare the intents specified with the traffic actually generated and notify the user if the usage of Socket Intents is unusual or defective.

8.2. Information Leakage

Varying the transport or IP layer parameters of packets belonging to different Streams or Messages multiplexed in the same encrypted association might enable an attacker to gain some ground truth about the shares of different kinds of traffic. As this might also be implied by packet timings, application developers might weight the small additional information disclosure against the possible performance gains. Using Socket Intents on Association level can be considered safe.

9. IANA Considerations

The Socket Intents type namespace SHOULD be managed by the IANA registry. Details conforming to [RFC5226] are laid out in Section 5, the initial types for the registry are described in Section 6.

10. Publications History

- The original idea of Socket Intents was published in [CoNEXT2013].
- A performance study "Socket Intents: OS Support for Using Multiple Access Networks and its Benefits for Web Browsing" is under submission.

11. Acknowledgements

This work has been supported by Leibniz Prize project funds of DFG - German Research Foundation: Gottfried Wilhelm Leibniz-Preis 2011 (FKZ FE 570/4-1).

12. References

12.1. Normative References

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

12.2. Informative References

[CoNEXT2013] Schmidt, P., Enghardt, T., Khalili, R., and A. Feldmann, "Socket intents", Proceedings of the ninth ACM conference on Emerging networking experiments and technologies - CoNEXT '13, DOI 10.1145/2535372.2535405, 2013.

[DASH] International Organization for Standardization, "Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats", Standard ISO/IEC 23009-1:2014 , June 2011, <https://www.iso.org/standard/65274.html>.

[I-D.pauly-taps-guidelines] Pauly, T., "Guidelines for Racing During Connection Establishment", draft-pauly-taps-guidelines-01 (work in progress), October 2017.

[I-D.tiesel-taps-communitgrany] Tiesel, P. and T. Enghardt, "Communication Units Granularity Considerations for Multi-Path Aware Transport Selection", draft-tiesel-taps-communitgrany-01 (work in progress), October 2017.

[I-D.trammell-taps-post-sockets] Trammell, B., Perkins, C., Pauly, T., Kuehlewind, M., and C. Wood, "Post Sockets, An Abstract Programming Interface for the Transport Layer", draft-trammell-taps-post-sockets-03 (work in progress), October 2017.

[RFC4594] Babiarz, J., Chan, K., and F. Baker, "Configuration Guidelines for DiffServ Service Classes", RFC 4594, DOI 10.17487/RFC4594, August 2006, <https://www.rfc-editor.org/info/rfc4594>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[RFC7556] Anipko, D., Ed., "Multiple Provisioning Domain Architecture", RFC 7556, DOI 10.17487/RFC7556, June 2015, <https://www.rfc-editor.org/info/rfc7556>.

Appendix A. Usage examples

A.1. Example 1

Consider a cellphone performing an OS upgrade. This process usually implies downloading a large file. This is a bulk transfer for which the application may already know the file size. Timing is typically noncritical and the data can be downloaded as background traffic with minimal cost and power overhead. It would not hurt if the TCP connection was closed during the transfer as the download can be continued.

For this case, the application should set the "Traffic Category" to "bulk", "Timeliness" to "background", and "Application Resilience" to "resilient". In addition, "Message Size to be Received" can be provided. Finally, the application may set the the "Cost Preferences" to "no_expense".

The OS can use this information and therefore may schedule this transfer on a flaky but not traffic-billed WiFi link and may reject the connection attempt if no cheap access link is available.

A.2. Example 2

Consider a user watching non-live video content using MPEG-DASH [DASH]. This usually means fetching a stream of video chunks. The application should know the size of each chunk and may know the bitrate and the duration of each chunk and the whole video. Disconnection of the TCP connection should be avoided because that might have an effect that is visible to the user.

For this case, the application should set the "Traffic Category" to "stream", the "Timeliness" to "stream", and "Application Resilience" to "sensitive". It may also provide the "Stream Bitrate Received" and "Duration" expected. Finally, the application may set the the "Cost Preferences" to "balance_cost".

The OS can use this information and, e.g, use MPTCP [RFC6824] if available to schedule the traffic on the cheaper link (e.g, WiFi) while establishing an additional subflow over an expensive link (e.g., LTE). If the desired bandwidth cannot be matched by the cheaper link, the more expensive link can be added to satisfy the desired bandwidth.

If the application would set the "Cost Preferences" to "optimize_cost", the OS would not schedule traffic on the second subflow and the application would reduce the video quality to adapt to the available data rate.

A.3. Example 3

Consider a user managing a remote machine via SSH. This usually involves at least one long-lived console session and possibly file transfers using SCP or rsync multiplexed on the same association (e.g. TCP connection).

For the packets sent for the console session, the application can set the "Traffic Category" to "control", the "Burstiness" to "random bursts", the timeliness to "interactive" and the resilience to "sensitive". For the packets of the file transfers, SSH may set both, the "Traffic Category" and "Burstiness" to "bulk". It may also know the size of the transfer and therefore sets "Message Size to be Sent" or "Message Size to be Received".

Assuming there are transport opportunities supporting multiple streams in a single association (e.g. SCPT [RFC4960]), the OS can use this information to schedule the streams over different links to meet their requirements (latency vs. bandwidth). In case the OS has to use TCP, it can still optimize by disabling TCP Nagle Algorithm for console session related transmissions.

Appendix B. Changes

B.1. Since -00

- Updates on Terminology (Object -> Message, Flow -> Assocication)
- More detailed Socket Intent Types specification

- Added implementation guidelines
- Many clairfications
- Fixed Authors and affiliations

Authors' Addresses

Philipp S. Tiesel
TU Berlin
Marchstr. 23
Berlin
Germany

Email: philipp@inet.tu-berlin.de

Theresa Enghardt
TU Berlin
Marchstr. 23
Berlin
Germany

Email: theresa@inet.tu-berlin.de

Anja Feldmann
TU Berlin
Marchstr. 23
Berlin
Germany

Email: anja@inet.tu-berlin.de