A Transparent Performance Enhancing Proxy Architecture To Enable TCP over Multiple Paths for Single-Homed Hosts
draft-ayar-transparent-sca-proxy-00

Abstract

This draft complements the work of MPTCP by defining a TCP Splitter/Combiner Architecture (SCA) that enables non-MPTCP-capable single-homed hosts to benefit from the multiple paths within Internet by means of performance enhancing proxies (PEPs) placed in the access networks.

SCA Proxies (SCAPs) make use of multiple paths in a way which is completely transparent to end-hosts. Since the existence of the SCAPs is shielded from the TCP end-points, they can be deployed in the Internet as well as on the end-systems.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 20, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
1. Introduction

Despite the fact that, the number of end-systems with multiple interfaces increases every day, usually only one interface is used to connect to the Internet, as for the single-interface hosts. This is motivated, among others, by the energy consumption. That is, the use of multiple interfaces drastically increases the power consumption and thus decreases the battery lifetime. When only one IP address is assigned to the end-system, the end-system is single-addressed.

Single addressed end-hosts may not benefit from the MPTCP [2] [3] [4] even if they are MPTCP-capable. MPTCP uses IP address pairs of the end-hosts to create the subflows. When two MPTCP-capable single-addressed end-systems transfer data via MPTCP, they will use regular TCP [5] like the non-MPTCP-capable hosts.

Moreover, the fact that an end-system is MPTCP-capable does not mean that MPTCP MUST be used by TCP connections. MPTCP MAY be disabled with TCP_MULTIPATH_ENABLE socket option on MPTCP-capable hosts [6].

Nevertheless, single-addressed end-hosts may benefit from multiple paths within the Internet by means of proxies located within the access networks. Such a proxy is described in this document and called further SCA Proxy (SCAP). It detects the TCP connection and then splits and shapes TCP traffic over multiple paths. If the access network is connected to the Internet via multiple gateway (GW) links, then TCP traffic may be distributed within the access network so that the packets will leave the network from different GW links. Thus, the bandwidth aggregation of the links may be achieved. If an access network is connected to the Internet via only one GW link, the SCAP may distribute TCP traffic to multiple paths within the access network and then another SCAP on the path may combine the packets before they leave the access network.

This document describes a splitter/combiner architecture (SCA) that may be used to develop transparent proxies for TCP over multiple paths solutions. SCA is a thin layer on top of IP which captures all the TCP packets. SCA distributes data/ACK packets of a TCP connection to the multiple paths and/or combines the distributed data/ACK packets of a TCP connection from multiple paths.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].
1.2. Terminology

Single-addressed end-system: An end-system, either with single or multiple interfaces, which is assigned only one IP address to connect to the Internet.

Network Device: An access point (AP) or router located in between a TCP sender and receiver.

SCA Proxy (SCAP): A running instance of the SCA.

SCAP Device: A network device on which an instance of SCA runs.

Pipe: A network path between a network interface of a SCAP device and a network interface of another SCAP device. Pipe is uniquely identified by the IP address pair of two SCAP device interfaces.

1.3. Reference Scenario

Figure 1 shows a reference scenario for two SCAP devices in different access networks. Host A and B are single-addressed non-MPTCP-capable hosts. They are connected to the Internet via network devices which have multiple (i.e., N) GW links to the Internet. The multiple interfaces of the network devices have IP addresses A-1..N and B-1..N.

When there is a TCP connection between an application on Host A and an application on Host B, TCP packets will follow a single path (e.g., via A-1<->Pipe-1<->B-1). If the GW links have low capacities (e.g., T1 links or DSL lines), then they constitute the bottleneck for the TCP connection.

As shown in Figure 1, when network devices are used as SCAP devices they may use multiple GW links to aggregate the capacity of the links for the TCP connection:

- SCAP-1 splits TCP data packets from Host A to Host B via its interfaces A-1 to A-N. Similarly, SCAP-2 splits TCP data packets from Host B to Host A via its interfaces B-1 to B-N.

- On the other side, SCAP-1 combines the packets split by SCAP-2. That is, it buffers and reorders the packets before they reach to the Host A. Similarly, SCAP-2 combines the packets split by SCAP-1.
2. SCA Design

In addition to increasing the TCP throughput, SCA aims at high deployment and adoption. To achieve that, the existence of the SCA MUST be shielded locally (i.e., from the applications and TCP on the TCP end-points) as well as on end-to-end (i.e., end points SHOULD NOT need to know which network devices are SCAP devices). Thus, SCA is designed as protocol-stack and end-host transparent.

2.1. SCA Protocol-Stack Transparency

In order to provide protocol-stack transparency, SCA MUST be located underneath the TCP. All the TCP multiple path related issues MUST be handled in the SCA which MAY be implemented as a thin layer. SCA SHOULD be located on top of the IP on the network devices. Alternatively, SCA MAY be located between the IP and data link layer. In both cases, all the TCP packets MUST pass through the SCA.

In order to recognize TCP connection establishment/release requests and TCP data/ACK packets, SCA MUST access and read the content of the TCP header. Thus, SCA SHOULD NOT be used on network devices if TCP headers can not be read. SCA SHOULD NOT access and read TCP payload.

Network devices MAY access TCP headers to work as middleboxes (i.e., performance-enhancing proxies (PEPs), firewalls, or NATs) [2]. As a kind of middlebox, SCAP devices function as PEPs that split and combine TCP traffic while preserving TCP end-to-end semantics (Figure 2).
2.2. SCA End-Point Transparency

SCAPs MUST work without any support from TCP end hosts. Neither TCP sender nor TCP receiver MUST be aware of the presence of any SCAPs. Thus, TCP end-hosts SHALL NOT be configured to find SCAPs. Instead, SCAPs SHOULD find other SCAPs to collaborate with.

3. SCA Components

SCA includes the following components to solve TCP problems [7] when multiple pipes are used:

- Packet Classifier that captures TCP packets and classifies them based on the header information.
- Connection Handler that reacts to new TCP connections by creating records of the new TCP connection and discarding connection information in case of a TCP connection release.
- Multiple Pipes Adapter that provides a common interface to use pipes in the network layer.
- Data/ACK Processors that process the data/ACK packets of the TCP connections.
- Signaling Unit that processes the SCA signaling packets.
- Configuration and Management Unit that is used to configure parameters used by the components.
3.1. Packet Classifier

Packet Classifier is responsible for looking at the TCP header to determine the packet type. An admission control policy MAY be applied to packets (Section 3.6) before accepting them into the SCAP. The packets that are not admitted MUST be passed to the lower/upper layer without SCAP processing.

SCAP-accepted TCP packets MUST be passed to the related SCA component:

- SCA signaling packets are used for detecting other SCAPs and exchanging control information among the SCAPs. All the signaling packets MUST be handled by the Signaling Unit.
- TCP connection establishment packets (i.e., packets with the SYN flag set) MUST be passed to the Connection Handler to construct connection record entries.
- TCP connection release packets MUST be passed to Connection Handler to remove entries of the connection records.
- Data/ACK packets MUST be passed to the Data/ACK Processor.

3.2. Connection Handler

Connection Handler manages the records of the TCP connections. TCP connection establishment procedure is followed to detect the TCP connections. TCP connection requests are detected by means of TCP segments with SYN flag.

TCP connection records MUST contain a unique connection identifier (tcp_conn_id) that is generated based on the TCP sender and receiver end-points (i.e., sender and receiver IP addresses and port numbers).

When TCP connection release packets (i.e., packets with the FIN or RST flag set) are received, the TCP connection release procedure is followed. When the connection tear-down is complete, all the connection related records MUST be deleted.

3.3. Multiple Pipes Adapter

MPTCP uses IP addresses of the end-hosts in the subflow end-points. Similarly, SCAPs SHOULD use IP address pairs of SCAP devices as pipes. In that case, the peer MUST be informed about the used addresses, as MPTCP informs its peer by means of ADD_ADDR option.
TCP packets MAY be sent over these pipes to the SCAP peer by means of:

- IP source routing [8]:
  
  TCP packets may be requested to follow a specific path. IP Loose/Strict Source and Record Route option MAY be used to specify the path they must follow.

- IP-in-IP encapsulation [9]:
  
  The source and destination SCAP IP addresses may be specified in the outer IP header to create tunnels between the SCAP peers.

- IP-in-IP tunneling [10]:
  
  SCAPs may use special headers to exchange signaling information. Tunnel Headers in the IP-in-IP tunneling may be used to carry SCA signaling information.

Since different methods may be used to send/receive data via SCA pipes, Multiple Pipes Adapter is defined as a component that interacts via specified APIs with other components of the SCA and handles interaction with IP.

Multiple Pipes Adapter interacts with IP to access and use the available pipes. Other SCA components use the Multiple Pipes Adapter APIs to get information about the pipes that may be used and to send data via them.

Multiple Pipes Adapter provides following APIs to the SCA components:

1. PIPE-ID-LIST find_pipes(tcp_conn_id, destination_address)

   is used to find pipes that may be used to send TCP data packets to the destination_address and returns the list of them, if any.

   tcp_conn_id is used to identify the connection that needs the pipes to be assigned. Multiple Pipes Adapter may access information about the connection (e.g., source/destination TCP end-point) by using the tcp_conn_id. The API also associates the list of pipes found with the connection.

   The PIPE-ID-LIST contains pipe_ids which are assigned by the Multiple Pipes Adapter to the available pipes (i.e., each pipe has a unique pipe_id).
Pipe implementation is shielded from the other components by means of pipe_ids: other components only know pipe_ids and Multiple Pipe Adapter handles pipes.

2. PIPE-PROPERTIES get_pipe_properties(tcp_conn_id, pipe_id)

is used to probe for characteristics of a pipe.

PIPE-PROPERTIES includes the required pipe parameters by the SCA instance (e.g., bandwidth, delay, utilization, packet loss rate, etc.).

In order to get pipe parameters, following methods may be used [7]: packet-pair based estimation of the pipe delays and capacities, tracking the TCP timestamps [11], SACK [12] or D-SACK [13] to estimate packet arrivals at the receiver, or probing pipe capacities with TCP-like cwnd increase and decrease [3].

In addition, the local MIB information (e.g., [14] [15] [16] [17]) may be used to set the initial values just after the pipes are found by Multiple Pipes Adapter.

If tcp_conn_id is NULL, then the total pipe characteristics are returned. Otherwise, tcp_conn_id is used to get pipe characteristics relevant to a TCP connection. For example, the bandwidth used by a TCP connection may be asked as well as the total bandwidth of the pipe.

3. PIPE-ID-LIST employ_pipes(tcp_conn_id, pipe_id_list)

is used by Multiple Pipes Adapter to associate pipes which become available after the connection establishment. For example, an interface becomes active or a host associates with an AP/router after the pipes were assigned to the TCP connection by use of the find_pipes() API.

Pipes in the pipe_id_list are associated with the connection with tcp_conn_id.

4. PIPE-ID-LIST withdraw_pipes(tcp_conn_id, pipe_id_list)

is used when a connection terminates or connection doesn’t need some pre-assigned pipes.

Pipes in the pipe_id_list are disassociated from the connection with tcp_conn_id.
It MUST be called after the connection termination and MAY be called during the connection to cancel the use of some pipes which are previously assigned to the connection.

In addition, Multiple Pipes Adapter may use this API to disassociate some pipes which become unavailable during the connection. For example, an interface becomes inactive or a host looses its connection to an AP/router.

5. PIPE-ID-LIST select_pipes(tcp_conn_id, pipe_id_list)

is used to select a subset from a set of pipes based on a pipe selection policy (Section 3.6).

tcp_conn_id is used to determine the pipe selection policy for the connection. The API uses withdraw_pipes() API to disassociate filtered pipes by the pipe selection policy and returns back the remaining pipes.

6. send_mp_packet(tcp_packet, tcp_conn_id, pipe_id)

is used to send a packet via one of the multiple pipes.

Since other SCA components only know the pipe_ids and do not know what the real pipes are, they use this API to send a tcp_packet which belongs to the connection with tcp_conn_id by using the the pipe with pipe_id.

tcp_conn_id is supplied to enable update for pipe characteristics related with the TCP connections. For example, if a pipe is assigned to more than one TCP connection, what proportion of the pipe is used by the assigned TCP connections.

7. TCP-PACKET check_encapsulation(tcp_packet)

is used to check whether tcp_packet is encapsulated or not.

Packets may be sent/received encapsulated. Since other components are shielded from the use of encapsulation, only the Multiple Pipes Adapter knows about the encapsulation.

send_mp_packet() API encapsulates packets before sending them when encapsulation is necessary.

When packets are received by the Packet Classifier, it uses check_encapsulation() to get decapsulated packets, if encapsulation is used for the connection. It returns the packet immediately if encapsulation is not used.
3.4. Data/ACK Processor

Data/ACK Processor is responsible for handling data/ACK packets. It shapes TCP traffic and decides on the scheduling of packets to the pipes.

Data/ACK Processor may apply to a TCP data/ACK packet one of the following operations (called 4-D) [7]:

1. duplicate: TCP packets may be (one or more times) duplicated and scheduled to multiple pipes to create robustness against packet losses.

2. delay: TCP packets may be buffered and delivered later, based on a timer or a condition. For example, some packets of a TCP flow may be intentionally delayed to shape the TCP traffic (e.g., to reduce number of out-of-order packet arrivals at the receiver side, or to separate sent packets by a given interval). Another example is to buffer and reorder out-of-order packets before they are passed to the TCP receiver.

3. deliver: TCP packets may be released immediately after their capture, or after the delay or duplicate operation. Delivery may be done locally for the incoming packets or over any available pipe to outgoing packets. For outgoing packets, one of the available pipes must be selected to efficiently utilize the pipes.

4. drop: TCP packets may be dropped by SCAP. For example, assume that a packet is buffered for early retransmission purposes. If the ACK for that packet arrived and the buffered packet is not needed to be kept any more, it may be dropped.

3.5. Signaling Unit

SCAPs on different SCAP devices MAY exchange signaling information by means of the following methods:

1. in-band signaling, by means of TCP options as in MPTCP. The SCAP that wants to send a signaling information will generate an SCA option and add it to the TCP packet. The peer SCAP will use the content of the SCA control option and remove it from the packet.

   The main drawback of the in-band signaling is that its size is limited by the TCP option space allowed. 16-21 bytes of space is left for the SCA options when the most common TCP options are encountered [4].
2. out-of-band signaling that may use signaling channels established between the SCAPs.

SCAPs may use a well-known port number to exchange signaling packets (e.g., RIP routing processes use UDP port 520 [18] or BGP systems use TCP port 179 [19]).

3. hybrid signaling that may use in-band signaling to carry out-of-band signaling channel end-point information between SCAPs. First a connection between the signaling channel end-points will be established. Then, out-of-band signaling over the connection may be used to exchange the signaling data.

Details of the signaling mechanism are out-of-scope of this document.

3.6. Configuration and Management Unit

Configuration and Management Unit is used to set parameters for the SCA components. The configuration parameters may be defined for each component:

- Packet Classifier may accept packets into the SCAP or reject them based on an admission control policy.

  Based on the TCP end-points, some applications or hosts may not benefit from the SCA:

  * Splitting the traffic may not be necessary for some applications. For example, short-lived flows like web traffic may be carried over only one path instead of multiple paths as they will most likely end within a couple of packets anyway. Thus, they may not be accepted for the SCA processing.

  * SCA may be provided as a service for some set of users. The users may be identified from the IP addresses they use to connect to the Internet. Thus, an admission policy may be applied based on the IP addresses of the TCP end-points.

- Connection Handler may configure the number of acceptable connections based on the current number of connections or the total number of pipes used by the connections.

- Multiple Pipes Adapter may use a pipe selection policy to select a subset of the pipes from the set of available pipes. Using different policies in pipe selection gives additional flexibility to the SCA.
Pipe selection policy may be based on the pipe parameters (e.g., delay, loss rate, throughput, etc.) to decide which pipes will be selected. For example, only the pipes that have close RTT values may be selected or some pipes with high BERs may be excluded.

In addition, pipe selection policy may be based on the TCP endpoints. For example, some privileged flows (e.g., flows originated from some IP addresses) may be assigned more pipes and some other flows may be penalized and forced to use only a single pipe.

4. SCA Deployment Scenarios

SCAPs may work in three modes:

1. Standalone SCAP: If there is only one SCAP on the path between the TCP sender and the receiver, SCAP must work alone
2. SCAP Pair: If there are two SCAPs on the path, then they may work as a peer
3. SCAP Chain: If there are multiple SCAPs on the path, then they may be used as cascaded chains

4.1. Standalone SCAP

As shown in Figure 3, SCA may still be used when there is only one SCAP on the path between the TCP end-hosts.

---

Figure 3: Single SCAP Between TCP End Hosts

---
If there are multiple pipes from the SCAP to the TCP end-hosts, the SCAP may split and shape TCP traffic over them (e.g., [20]). Since there is only one SCAP, it must work without any feedback via the signaling messages of the peer SCAP. Therefore, SCAP MUST NOT use signaling information during data transfer.

TCP end-hosts MUST NOT notice the existence of the SCAP. The TCP end-hosts will only see the effects of TCP traffic shaping.

4.2. SCAP Peer

As shown in Figure 2, SCAPs MAY work as a peer. The SCAP device peer may be located on the same access network as well as different access networks.

Section 1.3 discusses a scenario of SCAP devices located on different access networks.

Figure 4 shows two SCAP devices within the same access network. A TCP connection is established between an application on the single-homed Host A to an application on single-homed Host B.

In the data flow from Host A to Host B, SCAP-1 may split TCP packets via Pipe-1 and Pipe-2. This distribution may be used for load balancing purposes within the access network. In addition, if Pipe-1 and Pipe-2 are the bottleneck, then bandwidth aggregation of the pipes may be achieved. SCAP-2 combines the distributed packets, reorders them and sends them in-order to the Internet. Similarly, for the data flow from Host B to Host A, SCAP-2 may split packets over Pipe-1 and Pipe-2 while SCAP-1 may combine them.

---

Figure 4: SCAPs on The Same Access Network
4.3. SCAP Chain

As shown in Figure 5, more than two SCAPs may be available on the path between the TCP end-hosts. In that case, SCAP peers constitute a SCAP chain. A SCAP distributes TCP packets over multiple pipes, and its peer combines them. This split/combine process is then repeated between each SCAP pair.

![SCAP Chain Diagram]

Figure 5: A SCAP Chain Between TCP End Hosts

5. SCA Extensions

Since SCA is a thin layer underneath the TCP, it MAY also be placed on multi homed TCP end-hosts. SCA SHOULD be located in between the TCP and IP when it is used on the TCP end-hosts. Alternatively, SCA MAY be located between the IP and data link layer.

Figure 6 shows a SCAP peer on TCP end-hosts. SCA may be also used in the standalone setting (i.e., only on one TCP end-host).

Neither TCP nor IP SHOULD perceive the intervention of the SCAP:

- TCP sends outgoing TCP packets to its lower layer entity.
- IP passes incoming TCP packets to its upper layer entity.

In both cases, SCAP captures packets and processes them without TCP or IP having knowledge about that.
Finally, in the most general case, SCAPs on TCP end-hosts MAY collaborate with the SCAPs on network devices as shown in Figure 7.

6. IANA Considerations

This draft includes no IANA considerations.

7. Security Considerations

Will be added in a later version of this document.

8. References
8.1. Normative References


8.2. Informative References


Authors’ Addresses

Tacettin Ayar
Technical University Berlin
Telecommunication Networks Group,
Sekr. FT5, Einsteinufer 25,
Berlin 10587
Germany

Phone: +49 30 314 28225
Email: ayar@tkn.tu-berlin.de
MPTCP Proxies and Anchors
draft-hampel-mptcp-proxies-anchors-00

Abstract

MPTCP proxies and anchors are network-based functions, which support MPTCP connections. The MPTCP proxy provides multipath support for MPTCP-capable hosts on behalf of their MPTCP-unaware peers. This facilitates incremental deployment of MPTCP. The MPTCP anchor permits subflow establishment for MPTCP connections when direct interaction between end hosts fails. This permits tolerance to local IP protocol restrictions and it provides robustness in case of break-before-make mobility events. MPTCP proxies and anchors are especially suited for wireless access environments.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 11, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction .............................................. 3
2. MPTCP Network Functions ................................. 4
   2.1. MPTCP Proxy ........................................ 4
   2.2. MPTCP Anchor ........................................ 4
   2.3. Implicit vs. Explicit Proxies ....................... 5
   2.4. Implicit vs. Explicit Anchors ...................... 5
   2.5. End-Host Authentication ............................ 6
3. Deployment Scenarios .................................... 7
4. Operation with MPTCP Proxies ............................. 8
   4.1. Introduction of Implicit Proxy ..................... 8
   4.2. Subflow Management with Implicit Proxy ............ 11
   4.3. Introduction of Explicit Proxy .................... 12
   4.4. Subflow Management with Explicit Proxy .......... 15
5. Operation with MPTCP Anchors ............................ 16
   5.1. Introduction of Implicit Anchor ................... 16
   5.2. Subflow Management with Implicit Anchor .......... 17
   5.3. Introduction of Explicit Anchor ................... 19
   5.4. Subflow Management with Explicit Anchor .......... 21
   5.5. Protocol Translation with Anchor .................. 21
   5.6. Connection Robustness with Anchor ................. 22
6. Host Configuration ....................................... 23
7. New Signaling ........................................... 24
   7.1. PROXY Flag ......................................... 24
   7.2. ANCHOR Flag ........................................ 24
   7.3. JOIN Flag .......................................... 25
   7.4. Anchor-Reserved Address-Id Value .................. 26
   7.5. SEEK_ADDR Option .................................. 26
   7.6. FWD_ADDR Option .................................. 26
8. Security .................................................. 27
9. Acknowledgements ........................................ 28
10. References ............................................... 29
Authors’ Addresses ........................................ 30
1. Introduction

Currently, a host can enjoy the advantages of MPTCP only if its peer supports MPTCP as well [1]. This requirement creates an impediment to incremental deployment since the incentive for a host to upgrade to MPTCP is small as long as its potential peers have not upgraded too.

The incremental deployment problem especially applies to wireless environments, where traffic is dominated by interactions between mobile clients and network-side servers. While MPTCP can be rolled out rather quickly on mobile devices due to their short life cycle and frequent kernel upgrades, changes on application servers are usually harder to conduct. Further, the benefit of MPTCP may be more obvious to mobile users than to application service providers.

The incremental deployment problem can be overcome through the introduction of the MPTCP proxy, which resides in the network and provides MPTCP support for MPTCP-capable hosts (e.g. mobile devices) on behalf of their MPTCP-unaware peers (e.g. application services).

Since MPTCP proxies will most likely be run by network operators rather than application service providers they can support a multitude of application services, which makes incremental deployment of MPTCP rather efficient. Further, network operators may see a benefit in MPTCP deployment since it adds value to the network services they provide and since they mostly support a billing mechanism to reimburse themselves from MPTCP operation.

The MPTCP anchor is another MPTCP network function whose main purpose is to support end-to-end multipath connections. It operates as a subflow relay to facilitate subflow establishment between end points that do not enjoy direct reachability. This may happen, for instance, if the end points pertain to different IP protocols or if the hosts have lost end-to-end connectivity after a break-before-make mobility event.

The anchor function is most beneficial for peer-to-peer applications such as voice/video communications, which are run on MPTCP-enabled mobile or multi-homed devices. Flexibility in IP protocol support is important for this use case during the rollout of IPv6. The anchor function further allows the network operator to provide differentiated services for over-the-top applications.

This document discusses relevant features and signaling enhancements needed for the support of MPTCP proxies and MPTCP anchors.
2. MPTCP Network Functions

All network-based functions that interact with MPTCP connections through MPTCP signaling are referred to as "MPTCP network functions". MPTCP network functions are assumed to reside on "MPTCP network nodes". We consider two types of MPTCP network functions namely the MPTCP proxy and the MPTCP anchor. Anchor- and proxy functions can be collocated on one MPTCP network node.

2.1. MPTCP Proxy

The MPTCP proxy supports MPTCP on behalf of an MPTCP-unaware host. It splits the connection between multipath-capable and multipath-unaware host into a MPTCP section and a TCP section, respectively (Figure 1). All subflows established by the multipath-capable host terminate at the proxy.

Proxy operation is discussed in Section 4.

```
Host +----------+ Proxy +----------+ Host
   | MPTCP      | TCP      |
     \----------\        \----------\      /
      \          \              \       /
       \       \                \     /
        \     \                 \   /
         \  \                  \ /
          \ \                 / \
           \ \              /   \
            \ \            /     \
             \ \          /       \
              \ \        /         \
               \ \      /           \
                \ \    /             \
                 \ \ /               \
                  \ \                 \\ Split connection with MPTCP Proxy

Figure 1
```

2.2. MPTCP Anchor

The MPTCP anchor provides a network-based access point (i.e. IP address), which a MPTCP host can use to create additional subflows to the peer. The anchor relays all packets arriving from this host to the peer and vice versa. This creates a split subflow consistent of one section between host and anchor and the other between anchor and peer (Figure 2). The anchor’s operation involves address- and eventually also port translation. Anchors can also insert or modify MPTCP options of passing or relayed packets.

Anchor addresses can be introduced during connection establishment or at any later point in time. Anchor functions can be invoked or released during the entire lifetime of the connection.

An anchor function can interconnect end points using different IP
protocol versions with a subflow. In this case the anchor operates as an IP protocol translator (Section 5.5). The anchor also serves as a "meeting point" for the establishment of a new subflows when all other subflows have failed and direct end-to-end subflow establishment is not possible. This applies to scenarios where both end hosts have simultaneously moved or when one host moves while the other resides behind a firewall (Section 5.6).

Anchor operation is discussed in Section 5.

An implicit proxy resides on the direct routing path between two hosts engaging into a connection. This allows the hosts to establish the connection directly with each other, while the proxy can derive all information via packet inspection, insert and modify packets as necessary and thereby create the MPTCP-TCP split connection. This proxy is referred to as "implicit" since not explicit signaling is necessary.

When the proxy does *not* reside on the direct routing path between both hosts, explicit signaling is needed to introduce the proxy to the connection. The same applies to a proxy that does not reside on the path used for connection initiation. Such a proxy is referred to as "explicit" proxy.

An implicit proxy typically resides on a central router in the access network used by one of the hosts during connection establishment. An explicit proxy can reside in any network.

2.4. Implicit vs. Explicit Anchors

The terms "implicit" and "explicit" can also be defined for anchors.

An implicit anchor resides on the routing path used by a subflow of a...
MPTCP connection. This allows the anchor to derive all necessary connection-related information via packet inspection during the establishment of this subflow. Then, it can insert and modify packets as necessary and thereby offer anchor services to the end hosts.

When an implicit anchor resides on the initial subflow, it can offer services to *both* end hosts. Otherwise, it can offer services only to the subflow-initiating end host (see Section 5).

When the anchor does not reside on a direct routing path between both connection end points, explicit signaling is needed to introduce the anchor to the connection. Such an anchor is referred to as "explicit" anchor.

Anchors can support connections between two hosts as well as between a host and a MPTCP proxy. Usually, anchors are more beneficial in the former of the two scenarios.

2.5. End-Host Authentication

MPTCP proxies and anchors should support an explicit or implicit mechanism to authenticate one of the connection’s end hosts. This allows the proxy- or anchor operator to charge for operation of the respective MPTCP network function. There are also security reasons that require end-host authentication as outlined in Section 8.
3. Deployment Scenarios

The predominant use case for MPTCP proxies and MPTCP anchors is seen in wireless access networks. This is motivated by the increasing number of wireless devices that support multiple access technologies as well as multi-homing.

In one deployment scenario, the MPTCP network function resides on a central router of a wireless access network, e.g. a 3G/4G mobile network. Especially 3G and 4G mobile network operators may see an incentive for MPTCP proxy support since it allows them to dynamically offload traffic from licensed to unlicensed spectrum. Further, 3G- and 4G mobile networks already provide a centralized architecture, security support and charging functions, which can be used for MPTCP proxy or anchor operation.

There are also technical reasons to place MPTCP proxies inside cellular networks which are related to the wide-area coverage these networks typically provide. Therefore, the connection can be established via the cellular interface and subsequently migrated to other paths and networks. This substantially simplifies signaling since an implicit proxy/anchor can be used. Further, the cellular network can be used for reachability.

It is expected that anchor- and proxy functions are collocated.

For any deployment scenario, MPTCP-capable hosts need to be configured appropriately so that they can take advantage of implicit and explicit MPTCP network functions. Some aspects of host configuration are discussed in Section 6.
4. Operation with MPTCP Proxies

Proxies must be introduced to the connection during connection establishment and stay engaged during the entire lifetime of the connection.

4.1. Introduction of Implicit Proxy

<table>
<thead>
<tr>
<th>MPTCP</th>
<th>TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host A</td>
<td>IP_A0</td>
</tr>
<tr>
<td></td>
<td>______</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SFL_0</td>
<td></td>
</tr>
</tbody>
</table>

MPTCP-TCP split connection with implicit MPTCP proxy

![Figure 3]

The MPTCP-capable host starts a MPTCP connection by sending a TCP SYN packet with MP_CAPABLE option to its peer. The proxy inspects the packet and caches the end point locators consistent of IP addresses and port numbers as well as the key enclosed in the MP_CAPABLE option. Based on these locators, the proxy identifies and intercepts the peer’s SYN-ACK response packet. The implicit proxy does not change the locators contained on the packet.

In case the SYN-ACK response does not hold the MP_CAPABLE option, the proxy initiates multipath support. It creates a key on behalf of the peer, inserts a MP_CAPABLE option with this key into the SYN-ACK packet, and then forwards the packet to the connection-initiating host.

If the SYN-ACK response *does* contain an MP_CAPABLE option, the proxy is not needed. In this case, the network node can provide anchor functionality (see Section 5).
Connection initiation by MPTCP-capable host with implicit proxies on initial path

Figure 4

If multiple implicit proxies reside on the initial path, the proxy closest to the peer should become the MPTCP end point. Since this proxy is the first to receive the peer’s SYN-ACK packet, it automatically assumes multipath support by inserting the MP_CAPABLE option.
Connection initiation by MPTCP-unaware host with implicit proxies on initial path  

Figure 5

The implicit proxy can also support scenarios, where the peer rather than the connection-initiating host is MPTCP-capable. In this case, the MPTCP proxy adds the MP_CAPABLE option with its own key to the initial SYN packet. If the SYN-ACK response by the peer carries the MP_CAPABLE header, the proxy assumes multipath support.

If multiple proxies reside on the initial path in this latter case, the proxy closest to the session-initiating host should become the MPTCP end point. Since this proxy is the first to receive the peer’s SYN packet, it automatically assumes multipath support by inserting the MP_CAPABLE option into this SYN packet.

These signaling procedures work fine as long as at least one of the end hosts supports MPTCP. A problem occurs, when multiple proxies reside on the initial path but *neither* of the end hosts supports MPTCP. In this case, one proxy may add MP_CAPABLE to the SYN packet and the other to the SYN-ACK response packet. In this manner, both proxies end up creating a TCP-MPTCP-TCP split connection with multipath support between each other. Such a situation is likely to occur when each of the hosts’ access networks supports a proxy.

To avoid such a situation, the proxy inserting the MP_CAPABLE option into the SYN packet has to reveal its true nature by adding a PROXY flag to this option. When another proxy inspects the SYN packet and finds the MP_CAPABLE option with PROXY flag set, it should not insert...
MP_CAPABLE to the SYN-ACK response.

For implicit proxies, end-host authentication is implicitly provided by the host’s access authentication as long as the proxy resides in the access network of one of the end hosts. This makes additional signaling for end-host authentication unnecessary.

While this solution restricts operation of implicit proxies to access providers and their affiliates (e.g. roaming partners), it covers the most relevant deployment scenarios.

4.2. Subflow Management with Implicit Proxy

Since the proxy splits the connection into a MPTCP section and a TCP section, it becomes the end point for all further subflows. These subflows may be initiated by the MPTCP-capable host or by the proxy itself.

When the proxy is implicit, it must inform the multipath-capable host about its existence as well as its IP address. Otherwise, the multipath-capable host may try to establish subflows with the multipath-unaware peer. For this purpose, implicit proxies should set the PROXY flag on those MP_CAPABLE options they insert into SYN or SYN-ACK packets. This flag informs the multipath-capable host that the remote end point is represented by a proxy.

After connection establishment, the proxy should advertise its address via ADD_ADDR to the multipath-capable host. This step is necessary since the host does not know the proxy’s address.

Currently, the ADD_ADDR option also conveys the request for immediate subflow establishment to the enclosed address. This request has the purpose to enable subflow creation in reverse direction, i.e. when the peer resides behind a firewall.

Obviously, immediate subflow creation is not desirable when a proxy announces its IP address as an alternative end point. Therefore, the ADD_ADDR option should be furnished with a JOIN flag, which allows differentiating between the two purposes of ADD_ADDR. Hence subflow creation is only requested when the JOIN flag is set.

Since MPTCP options are not delivered reliably, the ADD_ADDR option may get lost. In this case, the host has no means to find out about the proxy’s IP address. For that reason, an additional SEEK_ADDR option should be supported which allows the host to solicit address advertisements by MPTCP network nodes and the peer.

SEEK ADDR should hold a field for the IP version requested. If this
field is set to zero, addresses pertaining to any IP version can be advertised.

4.3. Introduction of Explicit Proxy

![Diagram of MPTCP-TCP split connection with explicit MPTCP proxy](image)

Figure 6

If the proxy does not reside on the direct routing path of the intended connection the connection initiator must provide the proxy with explicit information on the peer’s network locator, i.e. IP address and port number. Since the explicit proxy may reside in a different network, additional signaling for host authentication has to be supported as well.

In case connection establishment reveals that both end hosts support MPTCP (or if the peer is supported by an implicit proxy), the explicit proxy function is not needed. In this case, the MPTCP network node automatically assumes explicit anchor function since it splits the initial subflow.

For connection establishment, the following signaling approaches are considered:

- In-band MPTCP signaling: The peer's network locator (i.e. IP address and port number) and the host’s authentication information are sent in-band on MPTCP options. Since the amount of
information is too large to fit into the TCP header of the initial SYN packet additional packets need to be exchanged for signaling purposes. A simple handhake can be realized where the MPTCP keys are used as authenticators (Figure 7):

Connection establishment with explicit proxy and in-band MPTCP signaling

Figure 7

* The connection-initiating host (host A) sends the SYN packet with MP_CAPABLE containing key_A as authenticator to the explicit MPTCP network node which caches key_A and host A’s locator.

* The MPTCP network node answers with SYN_ACK enclosing MP_CAPABLE with key_P as its own authenticator. It should *not* set the PROXY flag, since it doesn’t know at this point if proxy function is required.

* Host A sends an ACK enclosing FWD_ADDR, which holds the peer’s (i.e. host B’s) IP address. FWD_ADDR may also hold a port number if it is different from the port number used to address
the MPTCP network node.

* The MPTCP network node sends SYN with MP_CAPABLE holding key A to host B using its own IP address. It also sets the ANCHOR flag in MP_CAPABLE as discussed in Section 5.

* If host B is not MPTCP-capable, it responds with a simple SYN-ACK packet. Otherwise, it inserts MP_CAPABLE with key B into the SYN-ACK packet. If MP_CAPABLE is absent, the MPTCP network node assumes proxy function. Otherwise, it assumes anchor function.

* The proxy function sends an ACK to host A and encloses the MP_CAPABLE header with the PROXY flag set. This informs host A that host B does not support MPTCP and that the MPTCP network node has assumed proxy function. The MP_CAPABLE option does not have to hold any key at this point since all keying information has already been exchanged.

* The proxy function also sends a simple ACK to host B.

- Out-of-band MPTCP signaling: MPTCP introduces a separate signaling connection to exchange the necessary signaling information prior to establishment of the traffic connection. Since such an out-of-band solution substantially extends the present scope of MPTCP it is not further considered.

- Independent signaling: The host and the explicit MPTCP network node use an independent signaling protocol, in which the host authenticates itself and provides the peer’s locator. This protocol can be supported on session or application layer such as SIP [2], for instance. In this protocol, host and MPTCP network node establish the 64-bit key, which is cached by the proxy together with the peer’s locator and inserted by the host into MP_CAPABLE when initiating the MPTCP connection. This allows the network node to find the peer’s locator and to forward the SYN packet to the peer using its own IP address. The network node should set the ANCHOR flag when relaying the MP_CAPABLE packet to the peer. In case the SYN-ACK return packet arriving from the peer does *not* contain an MP_CAPABLE option, the network node assume proxy function. In this case, the proxy inserts MP_CAPABLE into the SYN-ACK packet, sets the PROXY flag and sends the packet to the connection-initiating host using its own IP address and port number as the packet’s source. The host responds with an ACK holding its own key as well as the key contained in the SYN-ACK packet.

Security issues related to such explicit proxy solutions are
discussed in Section 8.

It makes little sense to consider explicit-proxy scenarios where the connection-initiating host is not MPTCP-capable.

4.4. Subflow Management with Explicit Proxy

Subflow establishment with an explicit proxy follows along the same lines as for an implicit proxy. The explicit proxy, however, does not have to send an ADD_ADDR option since the host already knows the proxy’s address.
5. Operation with MPTCP Anchors

The anchor function splits subflows into two subflow sections, where each section interconnects an end host with one of the anchor's IP addresses (Figure 8). The anchor relays all packets arriving on one subflow section to the other by rewriting the IP addresses of the packet headers. The anchor may also translate port numbers. Anchors can also insert or modify MPTCP options of passing packets.

To keep end-to-end semantics in tact, the end nodes must have full awareness of the anchor's presence and its operation, i.e. if subflows are split and if an IP address belongs to an anchor or to the peer. Further, each host must know about the address-id its peer uses on the remote section of a split subflow. This ensures proper subflow tear-down in case the peer announces address removal via REMOVE_ADDR option.

Anchors can be introduced during connection establishment or at any later point in time. Anchor services can be invoked or released during the entire lifetime of the connection.

5.1. Introduction of Implicit Anchor

When an implicit anchor resides on the initial path, it caches the locators (i.e. IP addresses and port numbers) of the initial subflow as well as the keys exchanged during connection establishment. This allows the anchor to derive the corresponding tokens and cache them together with the end hosts' locators of this subflow.
Then, the anchor advertises its IP address to the end hosts by sending an ADD_ADDR option to one or to both end hosts. The ADD_ADDR option can be inserted into a packet that is passing on the initial subflow. The anchor may also insert a port number into the ADD_ADDR option.

The anchor has to mark the ADD_ADDR option in a manner that allows the host receiving the option to distinguish it from an ADD_ADDR option sent by the peer. For this purpose, the anchor should set the address-id in the ADD_ADDR option to an anchor-reserved value (e.g. 255). This does not lead to any conflict in case multiple anchors advertise their addresses with the same address-id value, since anchor addresses are considered invariants that need not be removed. Obviously, neither end hosts nor proxies should use this anchor-reserved address-id value.

When an implicit anchor resides on the path used by a later subflow, it caches the subflows locators as well as the token used during subflow establishment. Obviously, anchor support can only be provided for the host that initiated this subflow (host A) but not for its peer (host B) since the anchor only knows host B’s token. Therefore, the anchor advertises its IP address (and port number) only to host A.

The host receiving an ADD_ADDR options from an anchor caches the anchor’s address and port number contained in this option. When the ADD_ADDR option does not carry a port number, the remote port number of the subflow, where the option arrived, is cached instead.

Since the delivery of ADD_ADDR is not reliable, an end host may proactively seek anchor addresses via the SEEK_ADDR option introduced above. Both anchor and peer should respond with an ADD_ADDR option. The host can differentiate the originators of these replies by the enclosed address-id value.

5.2. Subflow Management with Implicit Anchor

When a host wishes to establish a subflow via anchor, it initiates a subflow to the address and port number cached for the anchor. Based on the destination port number of the SYN packet and the token contained in MP_JOIN, the anchor identifies the peer’s locator and forwards the packet to the peer using one of its own addresses and port numbers as the packet’s source. The peer’s SYN-ACK return packet and all following packets are relayed by the anchor in the same manner. Since the anchor does not change the address-ids contained in the MP_JOIN options of the initial handshake, each host learns the peer’s address-id used for this split-subflow.
While the host initiating the subflow (host A) is aware of the anchor’s presence, its peer (host B) may not know that this subflow is split because the anchor has not introduced itself to the peer or because the corresponding ADD_ADDR option got lost. In such a case, host B may falsely assume that the anchor’s IP address belongs to host A and map it to the address-id contained in MP_JOIN. This may lead to a conflict, in case host A has announced (or will announce) this address-id for another address. Further, host B may be tempted to use the anchor’s IP address for further subflows without knowing that this may invoke triangular routing.

To avoid such misunderstanding, the MP_JOIN option on the SYN packet has to be marked with an ANCHOR flag. This flag tells host B, that the source address on the packet header belongs to an anchor and that it is not associated with the address-id carried in the MP_JOIN option. The ANCHOR flag should be set by the anchor when relaying the SYN packet.

While host B may implicitly learn the anchor’s IP address in this manner, it is not advised to use this anchor for new subflows unless the anchor has explicitly advertised its IP address. Host B can solicit such IP address advertisement via SEEK_ADDR sent on the split subflow.

Each host should cache the peer’s address-id together with the state information it holds for the corresponding split subflow. In case the host receives an REMOVE_ADDR option, it can identify and tear down all split-subflows pertaining to the address-id held in this option.

The establishment of split subflows via anchor may introduce address-ids without the corresponding IP addresses. This is a similar situation as when direct end-to-end subflows pass network address translators, and it does not pose any principle problem.

The anchor caches the host’s locators and address-ids of the split subflow together with all information it holds for this connection. The anchor further keeps subflow-related state information for a short time frame after the subflow has been closed. The tokens and address ids are held for a short time after the last subflow known by this anchor has been closed. The tear-down delay permits the anchor to support break-before-make mobility scenarios discussed below.
5.3. Introduction of Explicit Anchor

If the anchor does not reside on a direct routing path it has to be introduced via explicit signaling by one of the hosts. The signaling has to include authentication information and the peer's locator. Since these are the same conditions as for explicit proxies the same solution scenarios can be applied as discussed in Section 4.3. For the reasons mentioned above, only scenarios with in-band MPTCP signaling and independent signaling are considered.

- In-band MPTCP signaling: The first four steps of the connection establishment are identical to those discussed for the explicit proxy (see Figure 7 and Figure 10):
* Steps 1-4 of connection establishment with explicit proxy.

* In case host B is MPTCP-capable, it inserts MP_CAPABLE with key B into the SYN-ACK response packet. Upon reception of this packet, the MPTCP network node assumes anchor function instead of proxy function.

* The anchor function sends an ACK to host A and encloses the MP_CAPABLE header with key_B and it sets the ANCHOR flag. This informs host A that host B does support MPTCP and that the MPTCP network node has assumed anchor function. At this point, host A overwrites key_P with key_B.

* The anchor function also sends an ACK to host B, where it inserts MP_CAPABLE with key_A and key_B and sets the ANCHOR flag. This tells host B that an anchor resides on the initial path.
Independent signaling: The explicit MPTCP network node relays the host's SYN packet holding the MP_CAPABLE option to the peer. If the SYN-ACK return packet holds the MP_CAPABLE option, the MPTCP network node assumes anchor function and the initial subflow becomes a split subflow. When relaying the SYN-ACK packet to the connection-initiating host, the anchor should set the ANCHOR flag. The host responds with an ACK holding MP_CAPABLE with both keys.

In case host B is not multipath-aware it may be supported by an implicit proxy residing on the path between host B and the explicit anchor. This proxy may reside in host B's access network for instance. The implicit proxy sets the PROXY flag in the MP_CAPABLE option of the SYN-ACK return packet as described in section 4.1. Since the explicit anchor sets the ANCHOR flag at the same time, host A can infer that the PROXY flag was set by an implicit proxy.

A host can also introduce an explicit anchor after connection establishment. This has only limited benefit since the peer won't be able to proactively use this anchor. Further, it is rather complicated to embed such an anchor introduction into the MP_JOIN handshake. For that reason, only methods involving independent signaling protocols are considered here. Such a protocol has to provide authentication information, the remote end point locator and the remote tokens used on this connection.

5.4. Subflow Management with Explicit Anchor

After introduction of the explicit anchor, establishment of further split subflows follows the same procedure as discussed for implicit anchors in Section 5.2.

5.5. Protocol Translation with Anchor

The anchor can be used for IP protocol translation on a split subflow in case host A wishes to support IPv6 on a new interface while host B only supports IPv4. Protocol translation further becomes necessary when one host moves from an IPv4 network to an IPv6 network while the peer's network only supports IPv4 (and vice versa).

In such scenarios, host A sends SEEK_ADDR on all subflows with the IPVer field set to IPv6. In response, anchors will send their respective IPv6 addresses. Then, host A initiates a new subflow to one anchors' IPv6 address. Since the anchor has cached at least one of host B's IPv4 addresses, it can create an IPv6/IPv4 split-subflow using an IPv6 and an IPv4 address.
5.6. Connection Robustness with Anchor

The anchor can provide enhanced connection robustness in scenarios where the only remaining subflow breaks and direct end-to-end subflow establishment is not possible. This may happen, for instance, when both hosts simultaneously move to a new address. Direct subflow establishment is not possible in this case since neither host knows the peer’s new IP address.

In another scenario, a host moves to a new IP address while the peer resides behind a firewall. The host cannot reach the peer since the firewall blocks packets arriving from a new address. The peer cannot reach the host either since it does not know the host’s new IP address.

In these scenarios, each host will try to establish a direct subflow first. If this fails each host tries subflow establishment via an anchor. Since the anchor recognizes the connection based on token and port number contained in each host’s SYN-packet, it can cache the host’s new address contained on the packet and use it as the destination for SYN-packets sent by the peer. In this manner, a new subflow can be established via the anchor.

For this purpose, the anchor should keep connection-related state information for some time after the subflow it is residing on has been torn down.

The procedure further requires that the anchor holds both end hosts’ tokens. This applies to anchors that reside on the initial path during connection establishment.
6. Host Configuration

MPTCP-capable hosts should be appropriately configured to take advantage of MPTCP network functions. In a deployment scenario, where proxies and anchors are integrated with a central router of a 3G/4G cellular network, the host should initiate connections that deserve MPTCP support via the cellular interface if possible. After connection establishment, additional paths can be established and utilized for traffic exchange.

In case explicit MPTCP network functions are provided, the host must be configured to support the proprietary protocol that introduces these nodes to the MPTCP connection. It must further be configured with the IP addresses for explicit proxies.

The details on host configuration and the criteria on path selection are beyond the scope of this document.
7. New Signaling

The following subsections discuss signaling changes necessary to support MPTCP network functions.

7.1. PROXY Flag

The PROXY flag needs to be added to the MP_CAPABLE option. The PROXY flag is set by MPTCP network nodes to announce that they assume proxy function.

The PROXY flag serves two purposes. It avoids that implicit proxies residing on the initial path between MPTCP-unaware hosts sustain a MPTCP connection with each other. It also informs a MPTCP-capable host that a proxy provides MPTCP on behalf of an MPTCP-unaware peer. This avoids unnecessary attempts by this host to establish subflows directly with the MPTCP-unaware peer.

The PROXY flag can be added into the header of the MP_CAPABLE option (shown as "P" in Figure 11).

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------+-------+-+-+-+-------+-+
|     Kind      |    Length     |Subtype|Version|C|P|A|(resvd)|S|
+---------------+---------------+-------+-------+-+-+-+-------+-+
|                   ...                 ...                     |
```

Figure 11

7.2. ANCHOR Flag

The ANCHOR flag needs to be added to the MP_CAPABLE option and to the MP_JOIN option. The flag informs the receiving host (or proxy) that an anchor has relayed this packet. This avoids misunderstandings about the source IP address of the packet and the address-id it carries.

The ANCHOR flag can be added to the headers of MP_CAPABLE and MP_JOIN (shown as "A" in Figure 11 and Figure 12).
7.3. JOIN Flag

The ADD_ADDR option is currently overloaded with two requests: 1) Cache this address and 2) initiate a subflow to this address right away. While this bundling of requests makes sense for end-to-end interactions, it becomes problematic for proxies and anchors, which only want to inform the peers about their respective addresses.

The issue can be resolved by adding a JOIN flag to the ADD_ADDR option. This, however, creates some issues since the option has no room left for additional information. The option is further rather long, especially if IPv6 addresses and port numbers have to be carried.

The following approaches can be considered:

- The IPVer field is reduced from 4 to 3 bits as proposed by Olivier Bonaventure. This still leaves room for 5 future IP versions apart from IPv4 and IPv6. (Note that IP version = 0 is used by SEEK_ADDR to refer to "all IP versions"). The released bit is available for the JOIN flag.

- The ADDRESS ID field is reduced by 1 bit to allocate room for JOIN as proposed by Costin Raiciu. This reduces the number of simultaneously supported addresses from 256 to 128 (or 255 and 127 if the anchor-reserved address-id is included as well).

- The ADD_ADDR option only provides addresses and address-ids while a new option conveys the request to create a subflow with respect to a specific address id. A similar proposal was also made by Yoshifumi Nishida.

- An octet is added to the ADD_ADDR option.
7.4. Anchor-Reserved Address-Id Value

The anchor-reserved address-id value is used when anchors advertise their IP address via ADD_ADDR. It informs the receiving host that the address belongs to an anchor and not to the peer.

The anchor reserved address-id value could be set for 255, for instance.

7.5. SEEK_ADDR Option

The SEEK_ADDR option is sent by a host to solicit its peer as well as proxies and anchors to advertise their addresses. This option is necessary for operation with proxies and anchors, which rely on reliable address advertising.

The SEEK_ADDR option holds the IP version field. If the value of this field is set to zero, addresses to all IP versions are sought.

SEEK_ADDR also permits peers and MPTCP network nodes to reduce address advertising. It is not necessary, for instance, to preemptively advertise IPv6 addresses on connections that only use IPv4 and vice versa.

The SEEK_ADDR option only holds the IP version field which leads to length of 3 octets (Figure 13).

```
1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+---------------+---------------+-------+-------+
|     Kind      |     Length    |Subtype| IPVer |
+---------------+---------------+-------+-------+
```

SEEK_ADDR option

Figure 13

7.6. FWD_ADDR Option

The FWD_ADDR option is used by a host to forward its peer’s IP address and port number to an explicit MPTCP network node.

The fields of the FWD_ADDR are identical to that of the ADD_ADDR option. Since both options have different semantic meanings they should also carry different subtypes.
8. Security

Mobility and multi-homing protocols are vulnerable to session redirection attacks such as session hijacking and distributed DoS (DDoS)[3]. For MPTCP, these matters have been discussed in [4]. The introduction of implicit proxies and anchors does not add new principal vulnerabilities.

One potential weakness is seen in connections via explicit proxy (or anchor), since the proxy can be used by the adversary to disguise its true location. In a DDoS attack, the adversary establishes multiple connections with the victim host and then floods the victim with a high volume of traffic on each connection. The severity of such an attack does not change when these connections are conducted via explicit proxy. Since the proxy uses its own IP address to forward the attacker’s packets to the victim, the attacker’s IP address remains hidden to the victim. This makes it impossible for the victim to identify an adversary prior to accepting a connection and to trace back the traffic flood to the attacker’s location.

One could argue that this situation could be improved by specifying a strong authentication method to be exercised between host and proxy. This, however, is not necessarily the case since a strong authentication protocol by itself does not enforce the use of strong authenticators.

Note that this situation is different for mobility protocols like Mobile IPv6. In Mobile IPv6, the home agent uses the mobile host’s unique home address as the source for traffic originated by the mobile host. The home address is therefore an authenticator of the traffic originator.

To support the same level of security, the explicit proxy could use a unique IP address for each host. While such an approach is feasible in IPv6 it may have limited applicability in IPv4 due to IP address exhaustion.
9. Acknowledgements

The authors wish to acknowledge suggestions and contributions on proxies and anchors by Olivier Bonaventure, Philip Eardley, Alan Ford, Mark Handley, Yoshifumi Nishida and Costin Raiciu.
10. References


Authors’ Addresses

Georg Hampel
Alcatel-Lucent
600 Mountain Ave
Murray Hill, NJ 07974
US

Phone: +1 908 582 2377
Fax: +1 908 582 8222
Email: georg.hampel@alcatel-lucent.com

Thierry Klein
Alcatel-Lucent
600 Mountain Ave
Murray Hill, NJ 07974
US

Phone: +1 908 582 3585
Fax: +1 908 582 8222
Email: thierry.klein@alcatel-lucent.com
Multipath TCP Support for Single-homed End-systems
draft-wr-mptcp-single-homed-07

Abstract

Multipath TCP relies on the existence of multiple paths between end-systems. These are typically provided by using different IP addresses obtained by different ISPs at the end-systems. While this scenario is certainly becoming increasingly a reality (e.g. mobile devices), currently most end-systems are single-homed (e.g. desktop PCs in an enterprise). It seems also likely that a lot of network sites will insist on having all traffic pass a single network element (e.g. for security reasons) before traffic is split across multiple paths. This memo therefore describes mechanisms to make multiple paths available to multipath TCP-capable end-systems that are not available directly at the end-systems but somewhere within the network.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 22, 2016.
1. Introduction

The IETF has specified a multipath TCP (MPTCP) architecture and protocol where end-systems operate a modified standard TCP stack which allows packets of the same TCP connection to be sent via different paths to an MPTCP-capable destination ([RFC6824], [RFC6182]). Paths are defined by sets of source and destination IP addresses. Using multiple paths has a number of benefits such as an increased reliability of the transport connection and an effect known as resource pooling [resource_pooling]. Most end-systems today do not have multiple paths/interfaces available in order to make use of multipath TCP, however further within the network multiple paths are the norm rather than the exception. This memo therefore describes ways how these multiple paths in the network could potentially be made available to multipath TCP-capable hosts that are single-homed.
In order to illustrate the general mechanism we make use of a simple reference scenario shown in Figure 1.

![Figure 1: Reference Scenario](image)

The scenario in Figure 1 depicts e.g. a possible SOHO or enterprise setup where a gateway/router is connected to two ISPs and a DHCP server gives out leases to hosts connected to the local network. Note that both, the gateway and the DHCP server could be on the same device (similar to current home gateway implementations). Also, the two ISPs could really be two different access technologies (e.g. LTE and DSL) provided by a single ISP.

The host is running a multipath-capable IP stack, however it only has a single interface. The methods described in the following sections will let the host make use of the gateway’s two interfaces without requiring modifications to the MPTCP implementation.

2. Approaches to Use Multiple Paths in the Network

All approaches in this document do not require changes to the wire format of MPTCP and both communicating hosts need to be MPTCP-capable. The benefit this approach has is that a) it has no implications on MPTCP standards, b) it will hopefully encourage the deployment of MPTCP as the number of scenarios where MPTCP brings benefits vastly increases and c) these approaches do not require complex middle-boxes to implement MPTCP-like functionality in the network as other approaches have suggested before.

2.1. Exposing Multiple Paths Through End-host Auto-configuration

Multipath TCP distinguishes paths by their source and destination IP addresses. Assuming a certain level of path diversity in the Internet, using different source and destination IP addresses for a given subflow of a multipath TCP connection will, with a certain probability, result in different paths taken by packets of different subflows. Even in case subflows share a common bottleneck, the
proposed multipath congestion control algorithm [RFC6356] will make sure that multipath TCP will play nicely with regular TCP flows.

In order to not require changes to the TCP implementation, we keep the above assumptions multipath TCP makes, i.e. working with different IP addresses to use different paths. Since the end-system is single-homed, all IP addresses are bound to the same physical interface. In our reference scenario in Figure 1, the host would e.g. receive more than one RFC1918 [RFC1918] private IP address from the DHCP server as depicted in Figure 2.

<table>
<thead>
<tr>
<th>Host</th>
<th>Gateway</th>
</tr>
</thead>
<tbody>
<tr>
<td>+------</td>
<td>+-----------------+</td>
</tr>
<tr>
<td>virt.</td>
<td>dst. 10.1.0.0/16</td>
</tr>
<tr>
<td>phys.</td>
<td>10.1.2.5</td>
</tr>
<tr>
<td>+-------+ +----------+</td>
<td></td>
</tr>
<tr>
<td>10.2.2.6</td>
<td>src. 10.2.0.0/16</td>
</tr>
</tbody>
</table>

Figure 2: Gateway internals

The gateway that is shown in Figure 2 has received two IP addresses, one from each ISP that it is connected to (ISP1 and ISP2). The NAT that the gateway is implementing needs to "map" each private IP address of the host consistently to a one of the addresses received by the ISPs, i.e. each private IP to a different public IP. Packets sent by the host to the gateway are then routed based on the source address found in the packets as illustrated in the figure. In other words, depending on the source address of the host, the packets will either go through ISP 1 or ISP 2 and TCP will balance the traffic across those two links using its built-in congestion control mechanism.

The way the gateway has received its public IP addresses is not relevant. It could be via DHCP, IPCP or static configuration. In order to configure the hosts behind the gateway, we propose to make use of provisioning domains [RFC7556], more specifically one provisioning domain per external gateway interface (the two interfaces to ISP1 and ISP2 in Figure 2). The DHCPv6 specification for encoding provisioning domains can be found in [I-D.ietf-mif-mpvd-dhcp-support].
In order to signal to the host, that each provisioning domain will result in a different path towards the Internet, this memo introduces a new DHCP option called EXT_ROUTE, which will be included in each provisioning domain sent by the server. The option value will determine which external interface is used to send the traffic when using the configuration information present in the respective provisioning domain.

Upon receipt of a DHCP offer including multiple provisioning domains, or multiple offers each including one or more provisioning domains, the client SHOULD create up to n virtual interfaces, where n is one less than the number of different EXT_ROUTE option values found in all received provisioning domains. Each virtual interface will contact the DHCP server and will request configuration information for the respective provisioning domains, excluding the configuration of the physical interface.

2.2. Heuristic Use of Multiple Paths

The auto-configuration mechanism above has the advantage that available paths and information on how to use them are directly sent to the end-host. In other words, there is an explicit signalling of the availability of multiple paths to the end-host. This has the advantage that the host can efficiently use these paths.

This method works well when multiple paths are available close to the end-host and means for auto-configuration are available. But that is not always the case. Another method to use different paths in the network without prior knowledge of their existence is to apply heuristics in order to exploit setups where Equal Cost Multi-path [RFC2991], a widely deployed technology [ECMP_DEPLOYMENT], or similar per-flow load-balancing algorithms are employed.

The ADD_ADDR option defined in [RFC6824] can be used to advertise the same address but a different port to open another subflow. Additionally, the MP_JOIN option can also be used to open another subflow with the same IP address and e.g. a different source port given that a different address ID is used. This means there are multiple scenarios possible (e.g. either sender-initiated or receiver-initiated) where single-homed end-hosts can influence the 5-tuple (source and destination IP addresses and port numbers plus protocol number) which is often used as the basis for per-flow load balancing. Changing the 5-tuple will only with a certain probability result in using a different path unless the load-balancing algorithm that is used is known to the MPTCP implementation (an assumption we cannot generally make). This means that a number of subflows might end up on the same path. Fortunately, the MPTCP congestion control
algorithm will make sure that the collection of subflows on that path will not be more aggressive than a single TPC flow.

3. Other scenarios and extensions

The reference scenario is only one conceivable setting. Other scenarios such as DSL broadband customers or mobile phones are conceivable as well. As an example, take the DSL scenario. The home gateway could be provided with multiple IP addresses using extensions to IPCP. The home gateway in turn can then implement the DHCP server and gateway functionality as described before. More scenarios will be described in future versions of this document.

4. Alternative approaches

One alternative is that a DHCP server always sends n offers, where n is the number of interfaces at the gateway to different ISPs. The client could then accept all or a subset of these offers. This approach seems interesting in environments where there are multiple DHCP servers, one for each ISP connection (think multiple home gateways). However, accepting multiple offers based on a single DHCP request is not standard’s compliant behavior (at least for the DHCPv4 case). Also, to cater for a scenario that only contains a single DHCP server, server changes are needed in any case. Finally, correct routing is not always guaranteed in these scenarios.

An interesting alternative is the use of ECMP at the gateway for load distribution and let MPTCP use different port numbers for subflows. Assuming that ECMP is available at the gateway, this approach would work fine today. The only drawback of the approach is that it involves a little trial and error to find port numbers that actually hash to different paths used by ECMP [RFC2991].

5. Acknowledgements

Part of this work was supported by Trilogy (http://www.trilogy-project.org), a research project (ICT-216372) partially funded by the European Community under its Seventh Framework Program. The views expressed here are those of the author(s) only. The European Commission is not liable for any use that may be made of the information in this document.

6. IANA Considerations

One new DHCP options is required by this version of this document.
7. Security Considerations

TBD.

8. References

8.1. Normative References


8.2. Informative References


[resource_pooling]


Authors’ Addresses

Rolf Winter
NEC Laboratories Europe
Kurfuersten-Anlage 36
Heidelberg 69115
Germany

Email: rolf.winter@neclab.eu

Michael Faath
University of Applied Sciences Augsburg
An der Hochschule 1
Augsburg 86161
Germany

Email: michael.faath@hs-augsburg.de

Andreas Ripke
NEC Laboratories Europe
Kurfuersten-Anlage 36
Heidelberg 69115
Germany

Email: andreas.ripke@neclab.eu