

Packet Flow in RouterOS

- Understanding Packet Flow
 - Overall Packetflow Diagram
 - Chains
 - Flow of Routed Packet
 - Forward
 - Input
 - Output
 - Flow of Bridged Packet
 - Bridge Forward
 - Bridge Input
 - Bridge Output
 - Forward With Firewall Enabled
 - Flow of Hardware Offloaded Packet
 - Switch Forward
 - Switch to CPU Input
 - CPU Output to Switch
 - Flow of MPLS Packet
 - Pop Label
 - Switch Label
 - Push Label
 - MPLS IP VPN
 - Logical Interfaces
 - IPSec Policies
- Fast Path
 - How Fast Path Works
- FastTrack
 - Configuration example: excluding specific host, from being Fast-Tracked
 - Requirements
- Packet flow for the visually impaired

Understanding Packet Flow

More advanced firewall setups, or complicated tasks, such as traffic prioritization, routing policies, where it is necessary to utilize more than one RouterOS facility, require knowledge: How do these facilities work together? What happens when and why?

RouterOS packet flow diagram and flow examples will try to answer these questions.

It would be very complicated to represent what is going on with the packet in one diagram, therefore a packet flow diagram is divided into three parts:

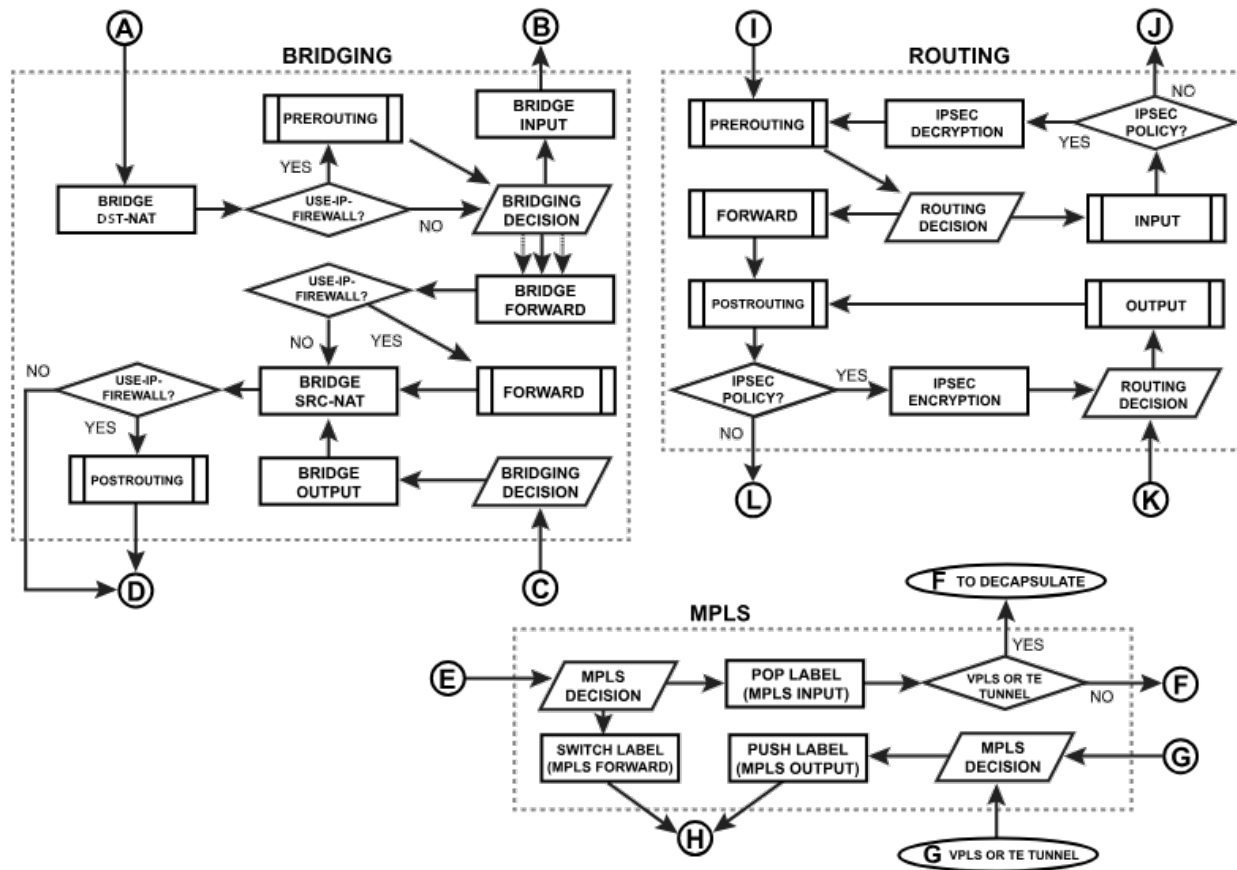
- overall diagram;
- detailed bridging, routing, and MPLS flow diagram;
- a diagram that shows what facilities and in what order are included in prerouting, input, forward, output, and postrouting.

Overall Packetflow Diagram

Let's look at the overall diagram. It looks complicated at first, but after we go through the diagram with examples it will become much clearer.



There are 4 boxes in the center of the diagram: Bridging, Routing, MPLS decisions, and local router processes. So for example, if the packet needs to be routed over the router, a packet will flow as illustrated in the image below. Without looking deeper into each facility, the packet enters the in-interface, the router determines that it is IP traffic and needs to be routed, the packet goes through all routing processes and exits the out-interface.



A simple explanation of each box before we go further with examples:

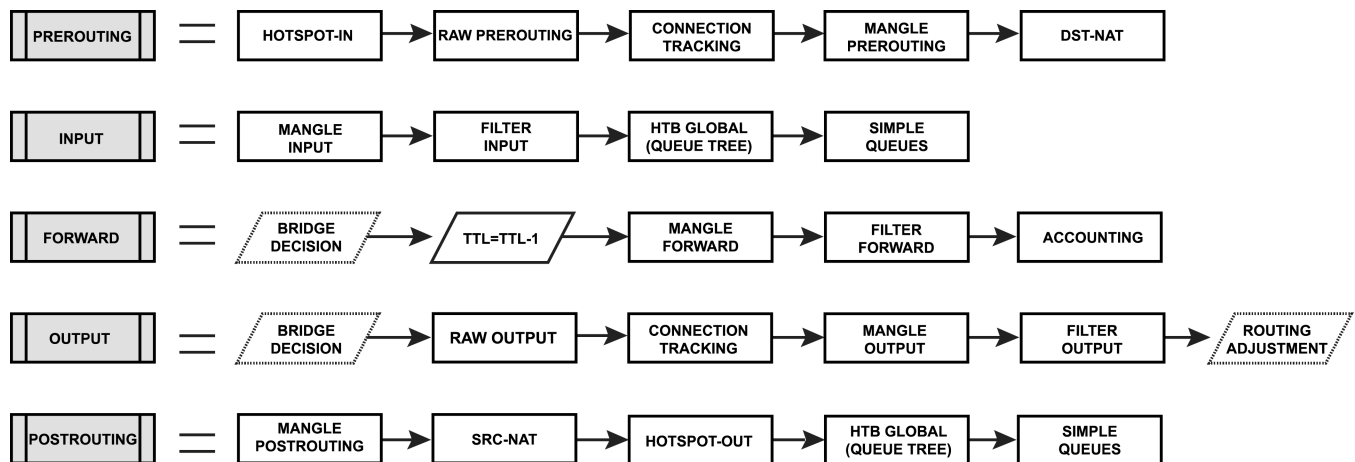
- **routing decision** - go through routes in the routing table to find a match for the destination IP address of the packet. When a match is found - the packet will be processed further, in case of no match - the packet will be discarded.;
- **mpls decision** - what to do with the packet based on MPLS forwarding tables;
- **bridging decision** - bridge goes through the MAC address table to find a match for the destination MAC address of the packet. When a match is found - the packet will be processed further, in case of no match - multiple copies of the packet will be created and packets will be flooded (sent out via all bridge ports). A single packet copy will also reach a bridge input chain as the bridge interface itself is one of the many destinations. When using `vlan-filtering=yes`, packets that are not allowed due to the `/interface bridge vlan` table, will be dropped at this stage.
- **use-ip-firewall** - whether a `'use-ip-firewall'` option is enabled in bridge settings;
- **ipsec-policy** - whether a packet matches any of configured IPsec policies;

Chains

RouterOS consist of a few default chains. These chains allow you to filter packets at various points:

- The **PREROUTING** chain: Rules in this chain apply to packets as they just arrive on the network interface. This chain is present in the *nat*, *mangle* and *raw* tables.
- The **INPUT** chain: Rules in this chain apply to packets just before they're given to a local process. This chain is present in the *mangle* and *filter* tables.
- The **OUTPUT** chain: The rules here apply to packets just after they've been produced by a process. This chain is present in the *raw*, *mangle*, *nat*, and *filter* tables.
- The **FORWARD** chain: The rules here apply to any packets that are routed through the current host. This chain is only present in the *mangle* and *filter* tables.
- The **POSTROUTING** chain: The rules in this chain apply to packets as they just leave the network interface. This chain is present in the *nat* and *mangle* tables.

Each of the prerouting, input, forward, output, and postrouting blocks contains even more facilities, which are illustrated in the third part of the packet flow diagram:



A simple explanation of each box before we go further with examples:

- **Hotspot-in** - allows to capture traffic that otherwise would be discarded by connection tracking - this way our Hotspot feature is able to provide connectivity even if networks settings are an incomplete mess ;
- **RAW Prerouting** - RAW table prerouting chain;
- **Connection tracking** - packet is processed by connection tracking;
- **Mangle Prerouting** - Mangle prerouting chain;
- **Mangle Input** - Mangle input chain;
- **Filter Input** - Firewall filter input chain;
- **HTB Global** - Queue tree;
- **Simple Queues** - is a feature that can be used to limit traffic for a particular target;
- **TTL** - indicates an exact place where the Time To Live (TTL) of the routed packet is reduced by 1 if TTL becomes 0, a packet will be discarded;
- **Mangle Forward** - Mangle forward chain;
- **Filter Forward** - Filter forward chain;
- **Accounting** - Authentication, Authorization, and Accounting feature processing;
- **RAW Output** - RAW table output chain;
- **Mangle Output** - Mangle output chain;
- **Filter Output** - Firewall filter output chain;
- **Routing Adjustment** - this is a workaround that allows to set up policy routing in mangle chain output (routing-mark) ;
- **Mangle Postrouting** - Mangle postrouting chain;
- **Src Nat** - Network Address Translation srcnat chain;
- **Dst Nat** - Network Address Translation dstnat chain;
- **Hotspot-out** - undo all that was done by hotspot-in for the packets that are going back to the client;

Flow of Routed Packet

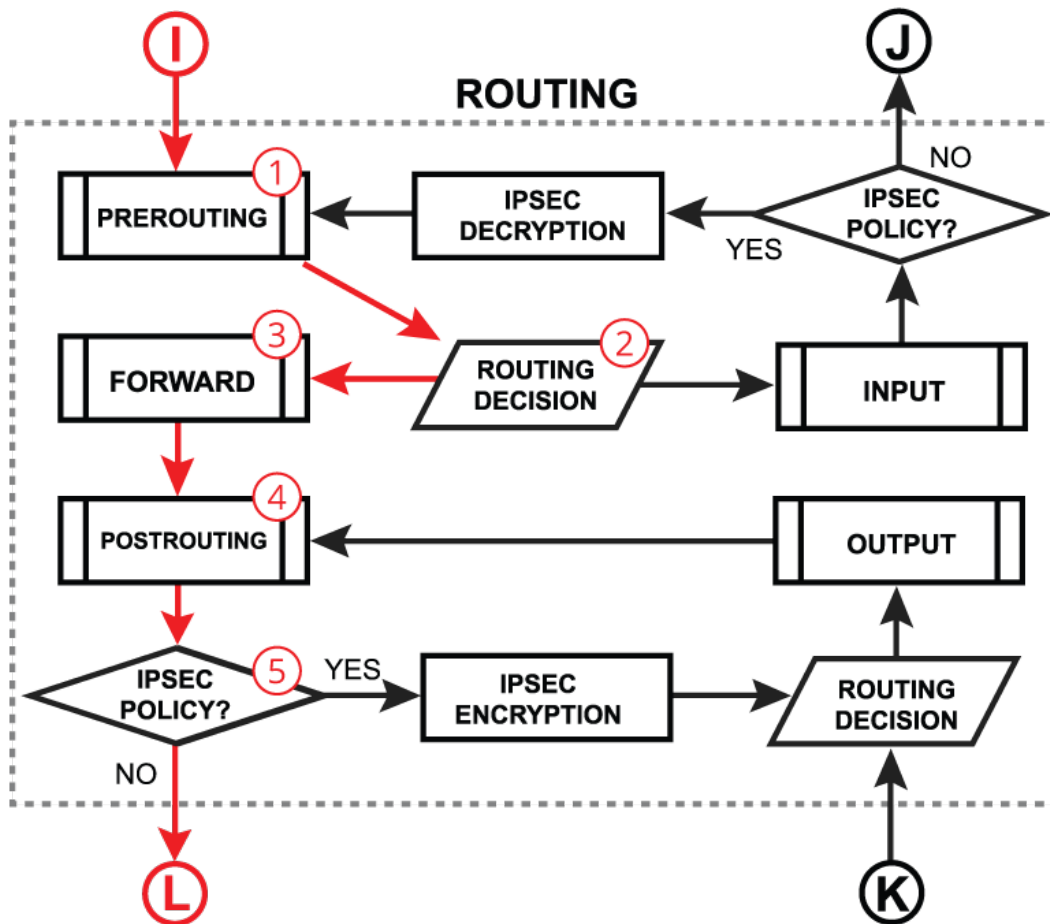
Forward

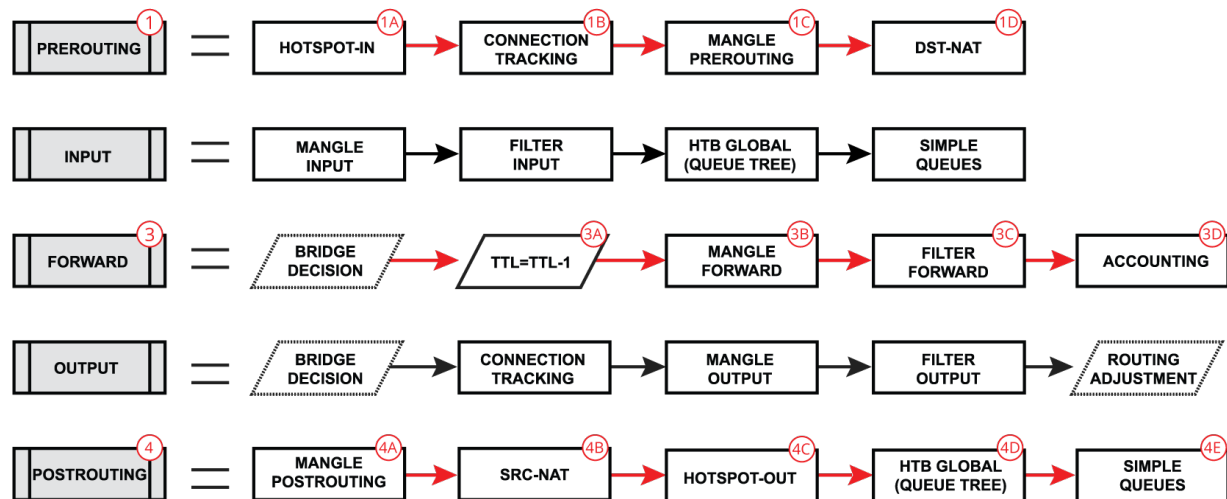
Now, let's take our first example where the packet gets routed over the router and look deeper through what facilities packet goes:

We already learned that packet goes into the in-interface, the router determines that it is an IP packet and needs to be routed, and here starts the complicated process:

1. The packet enters prerouting processing:
 - a. check if there is a hotspot and modify the packet for hotspot use
 - b. process packet through RAW prerouting chain;
 - c. send the packet through connection tracking;
 - d. process packet through Mangle prerouting chain;
 - e. process packet through NATs dst-nat chain;
2. Run packet through routing table to make routing decision;
3. The packet enters the forward process;

- a. check TTL value;
 - b. process packet through Mangle forward chain;
 - c. process packet through the Filter forward chain;
 - d. send the packet to accounting processes;
4. A packet enters postrouting process;
- a. process packet through Mangle postrouting chain;
 - b. process packet through NATs src-nat chain;
 - c. if there is a hotspot undo any modifications made in hotspot-in;
 - d. process packet through queue tree (HTB Global);
 - e. process packet through simple queues;
5. Check if there is IPsec and then process through IPsec policies;





Input

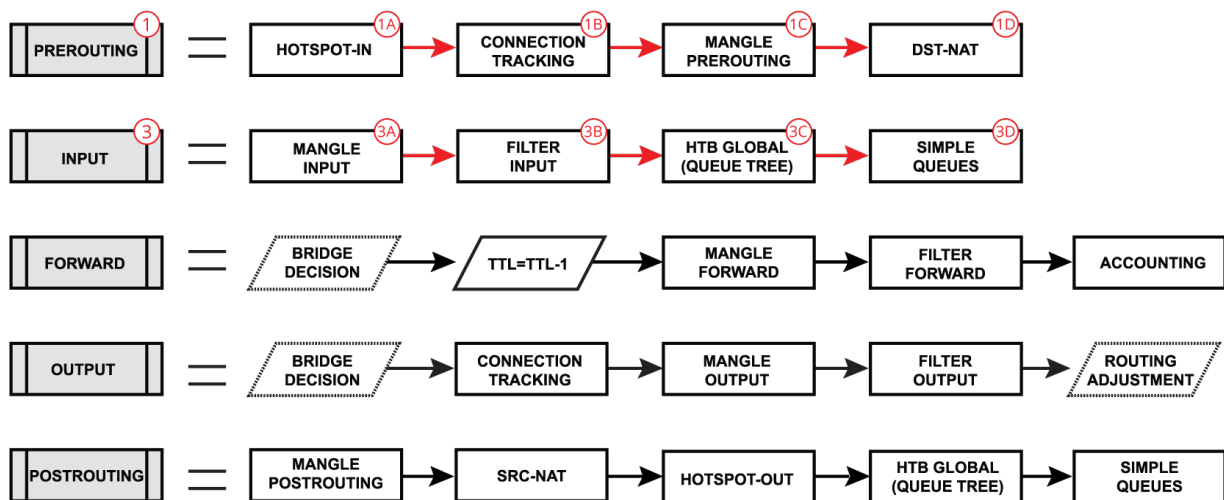
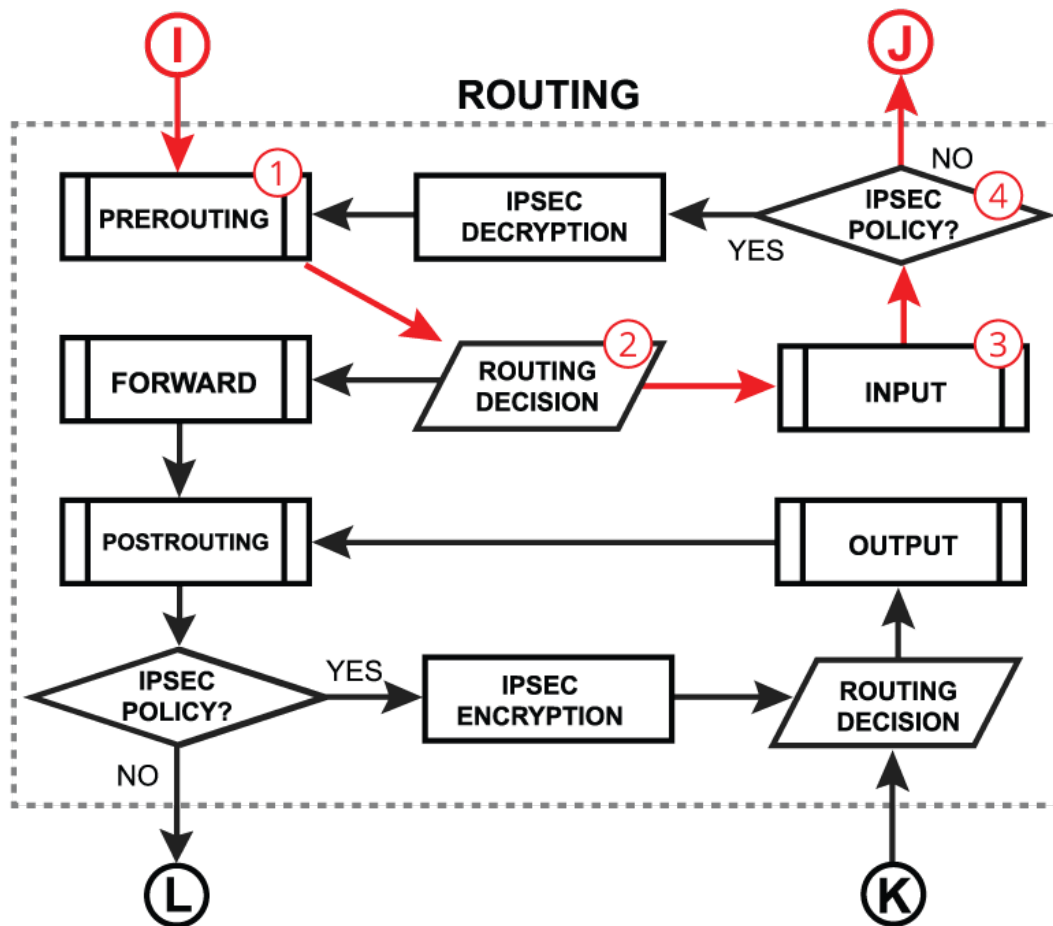
We already learned that packet goes into the in-interface, the router determines that it is an IP packet and needs to be routed, and here starts the complicated process:

1. A very similar process happens when a packet's destination is a router (routing input): Packet enters prerouting processing:
 - a. - check if there is a hotspot and modify the packet for hotspot use;
 - b. - process packet through RAW prerouting chain;
 - c. - send a packet through connection tracking;
 - d. - process packet through Mangle prerouting chain;
 - e. - process packet through NATs dst-nat chain;
2. Run packet through routing table to make routing decision;
3. A Packet enters the input process;
 - a. - process packet through Mangle input chain;
 - b. - process packet through Filter input chain;
 - c. - process packet through queue tree (HTB Global);
 - d. - process packet through simple queues;
4. Check if there is IPsec and then process through IPsec policies.

Output

Or when a packet is originated from the router (routing output):

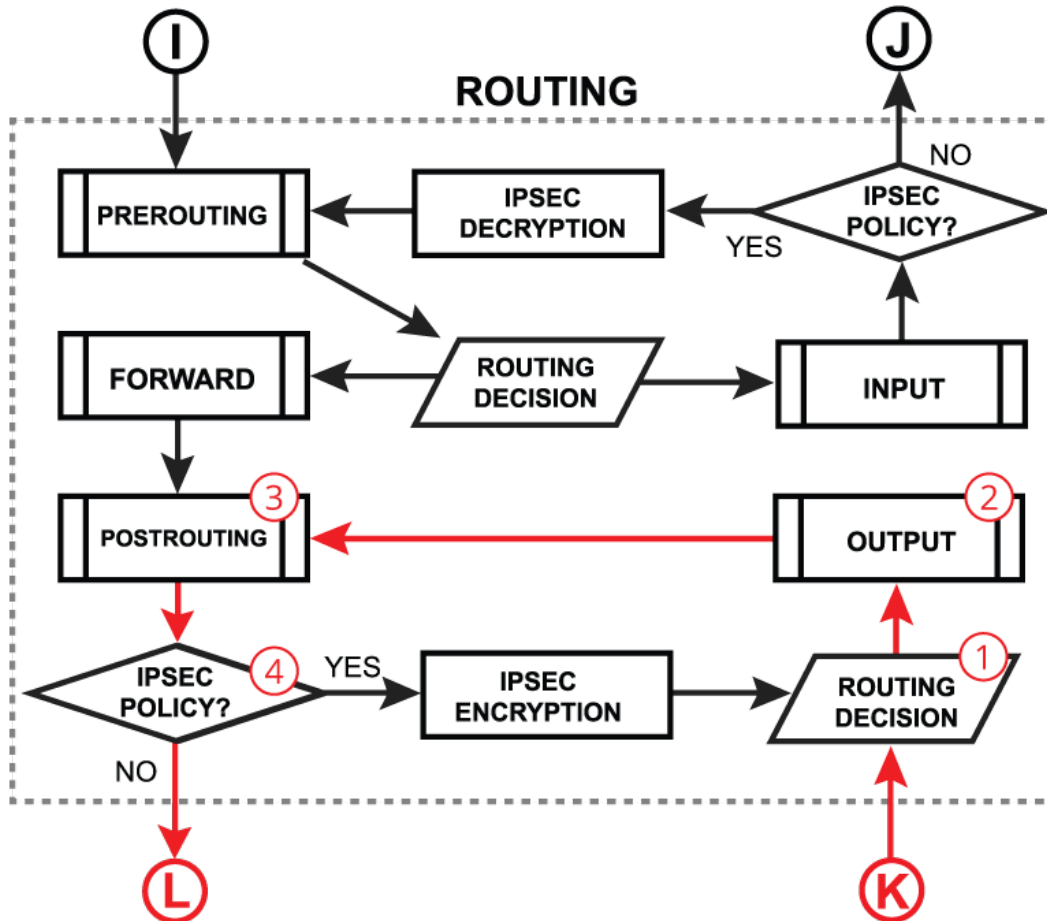
1. The packet is originated from the router itself
 - a. the packet goes through the routing table to make a routing decision
2. A packet enters the output process
 - a. process packet through the Bridge decision;
 - b. send the packet through connection tracking;
 - c. process packet through the Mangle output chain;
 - d. process packet through the Filter output chain;
 - e. send the packet to routing adjustment (policy routing)
3. The packet enters postrouting process;



- b. - process packet through NATs src-nat chain;
 c. - if there is a hotspot undo any modifications made in hotspot-in;

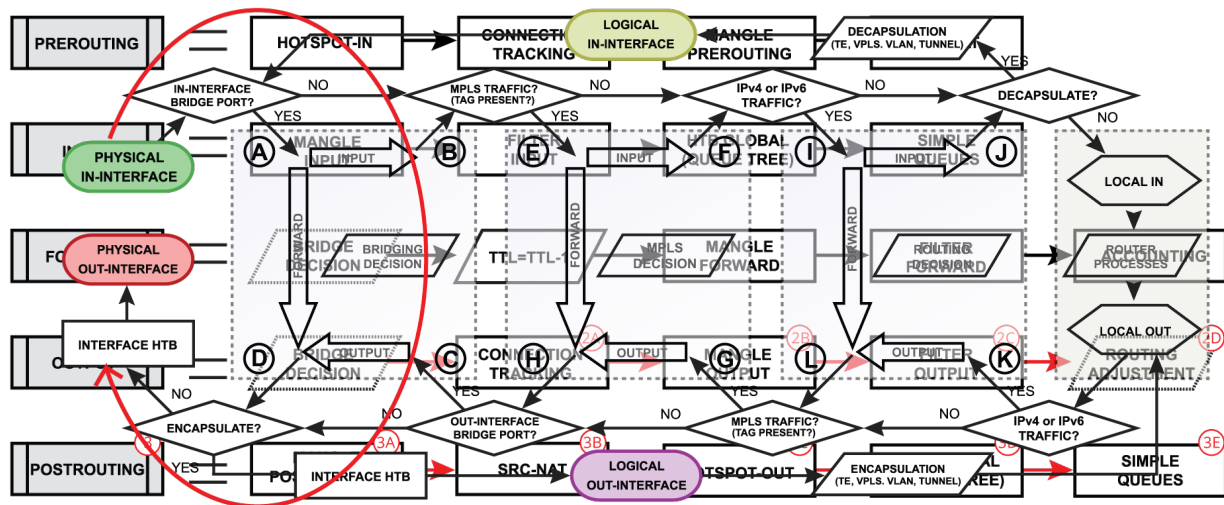
- d. - process packet through queue tree (HTB Global);
- e. - process packet through simple queues;

4. Check if there is IPsec and then process through IPsec policies;



Flow of Bridged Packet

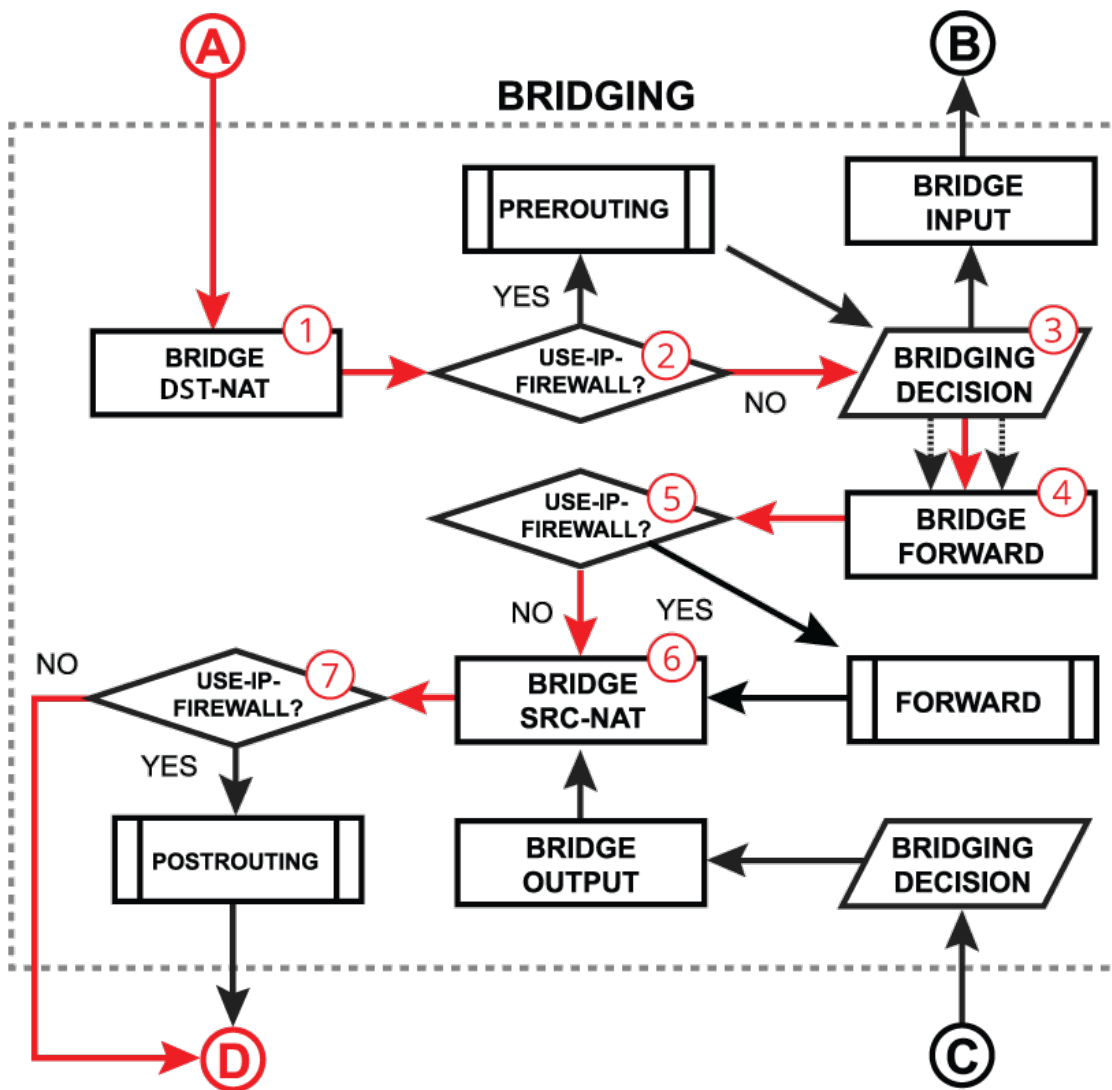
Below is discussed a general bridging process in RouterOS. Most of the packets will always follow the same processing path, but in certain configurations (e.g. with enabled VLAN filtering, horizon, STP, DHCP, or IGMP snooping) some packets can be treated differently. Please visit the bridging manual for more specific information.



Bridge Forward

Bridge forward is a process that takes place when a packet is forwarded from one bridge port to another, essentially connecting multiple devices on the same network. After receiving a packet on the in-interface, the device determines that the in-interface is a bridge port, so it gets passed through the bridging process:

1. A packet goes through the bridge NAT dst-nat chain, where MAC destination and priority can be changed, apart from that, a packet can be simply accepted, dropped, or marked;
2. Checks whether the use-ip-firewall option is enabled in the bridge settings;
3. Run packet through the bridge host table to make a forwarding decision. A packet that ends up being flooded (e.g. broadcast, multicast, unknown unicast traffic), gets multiplied per bridge port and then processed further in the bridge forward chain. When using `vlan-filtering=yes`, packets that are not allowed due to the `/interface bridge vlan` table, will be dropped at this stage.
4. A packet goes through the bridge filter forward chain, where priority can be changed or the packet can be simply accepted, dropped, or marked;
5. Checks whether the use-ip-firewall option is enabled in the bridge settings;
6. A packet goes through the bridge NAT src-nat chain, where MAC source and priority can be changed, apart from that, a packet can be simply accepted, dropped, or marked;
7. Checks whether the use-ip-firewall option is enabled in the bridge settings;



For RouterOS v6:

When bridge `vlan-filtering` is enabled, received untagged packets might get encapsulated into the VLAN header before the "DST-NAT" block, which means these packets can be filtered using the `mac-protocol=vlan` and `vlan-encap` settings. Encapsulation can happen if the outgoing interface has `frame-types` set to `admit-all` or `admit-only-untagged-and-priority-tagged`.

Tagged packets might get decapsulated on the "BRIDGING DECISION" block, which means these packets will no longer match the `mac-protocol=vlan` and `vlan-encap` settings. Decapsulation can happen if the packet's VLAN ID matches the outgoing port's untagged VLAN membership.

For RouterOS v7 and newer:

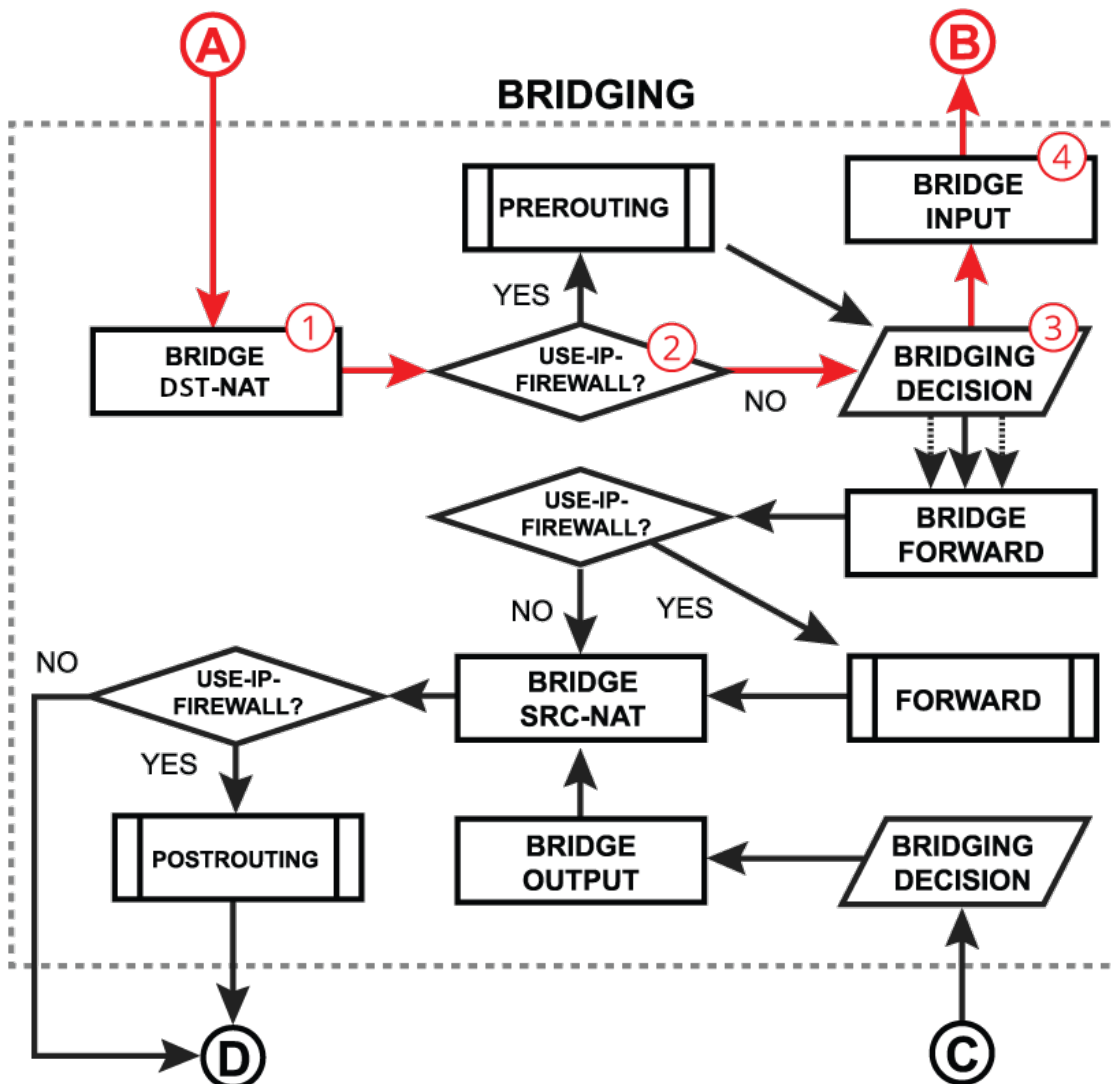
When bridge `vlan-filtering` is enabled, received untagged packets might get encapsulated into the VLAN header on the "BRIDGING-DECISION" block, which means these packets can be filtered using the `mac-protocol=vlan` and `vlan-encap` settings. Encapsulation can happen if the outgoing interface has `frame-types` set to `admit-all` or `admit-only-untagged-and-priority-tagged`.

Tagged packets might get decapsulated on the "BRIDGING DECISION" block, which means these packets will no longer match the `mac-protocol=vlan` and `vlan-encap` settings. Decapsulation can happen if the packet's VLAN ID matches the outgoing port's untagged VLAN membership.

Bridge Input

Bridge input is a process that takes place when a packet is destined for the bridge interface. Most commonly this happens when you need to reach some services that are running on the bridge interface (e.g. a DHCP server) or you need to route traffic to other networks. The very first steps are similar to the bridge forward process - after receiving a packet on the in-interface, the device determines that the in-interface is a bridge port, so it gets passed through the bridging process:

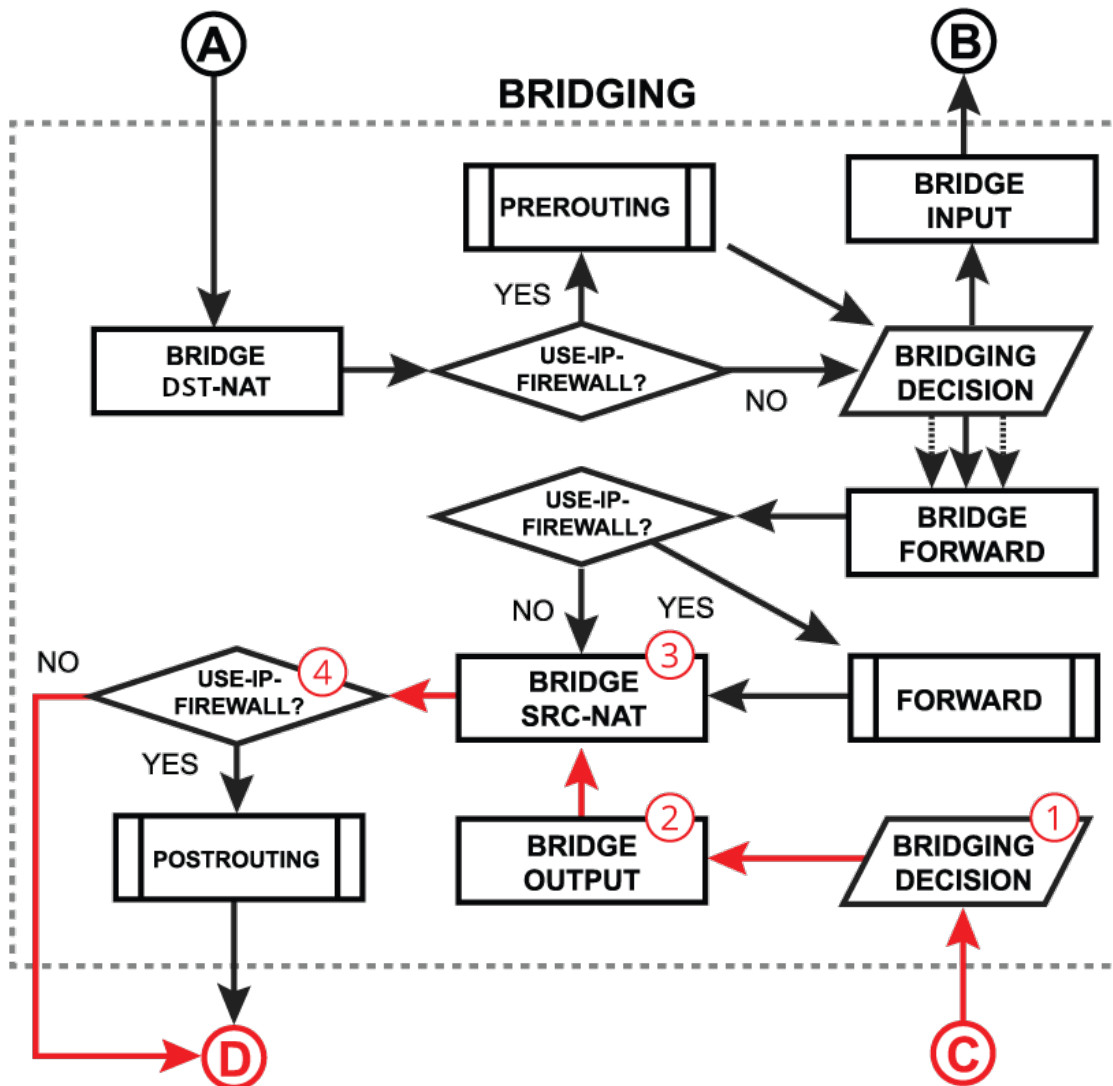
1. A packet goes through the bridge NAT dst-nat chain, where MAC destination and priority can be changed, apart from that, a packet can be simply accepted, dropped, or marked;
2. Checks whether the use-ip-firewall option is enabled in the bridge settings;
3. Run packet through the bridge host table to make a forwarding decision. A packet where the destination MAC address matches the bridge MAC address will be passed to the bridge input chain. A packet that ends up being flooded (e.g. broadcast, multicast, unknown unicast traffic), also reaches the bridge input chain as the bridge interface itself is one of the many destinations;
4. A packet goes through the bridge filter input chain, where priority can be changed or the packet can be simply accepted, dropped, or marked;



Bridge Output

Bridge output is a process that takes place when a packet should exit the device through one or multiple bridge ports. Most commonly this happens when a bridge interface itself tries to reach a device connected to a certain bridge port (e.g. when a DHCP server running on a bridge interface is responding to a DHCP client). After a packet is processed on other higher-level RouterOS processes and the device finally determines that the output interface is a bridge, the packet gets passed through the bridging process:

1. Run packet through the bridge host table to make a forwarding decision. A packet that ends up being flooded (e.g. broadcast, multicast, unknown unicast traffic), gets multiplied per bridge port and then processed further in the bridge output chain.
2. A packet goes through the bridge filter output chain, where priority can be changed or the packet can be simply accepted, dropped, or marked;
3. A packet goes through the bridge NAT src-nat chain, where MAC source and priority can be changed, apart from that, a packet can be simply accepted, dropped, or marked;
4. Checks whether the use-ip-firewall option is enabled in the bridge settings;

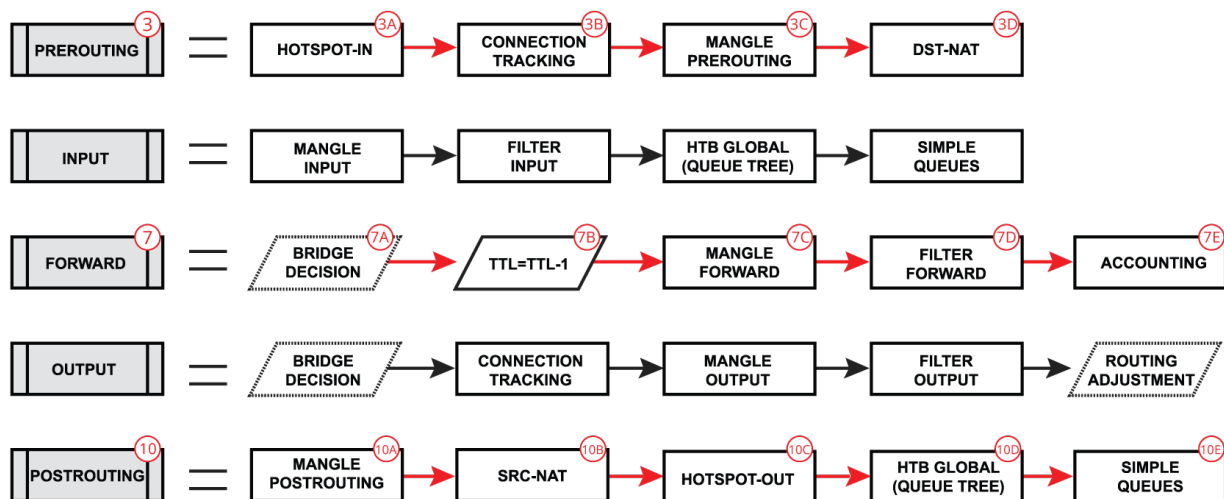
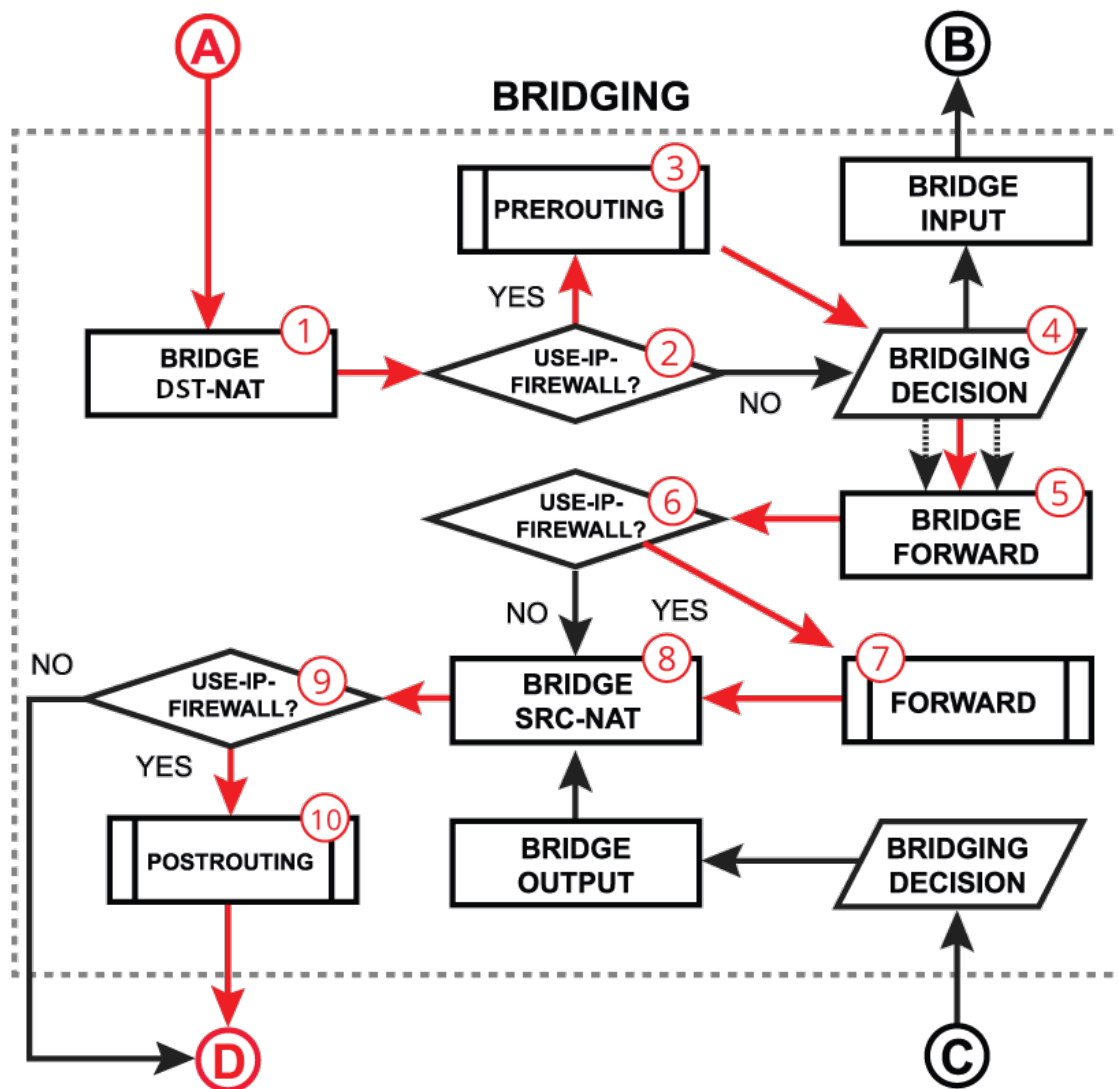


Forward With Firewall Enabled

In certain network configurations, you might need to enable additional processing on routing chains for bridged traffic, for example, to use simple queues or an IP firewall. This can be done when the use-ip-firewall is enabled under the bridge settings. Note that additional processing will consume more CPU resources to handle these packets. All the steps were already discussed in previous points, below is a recap:

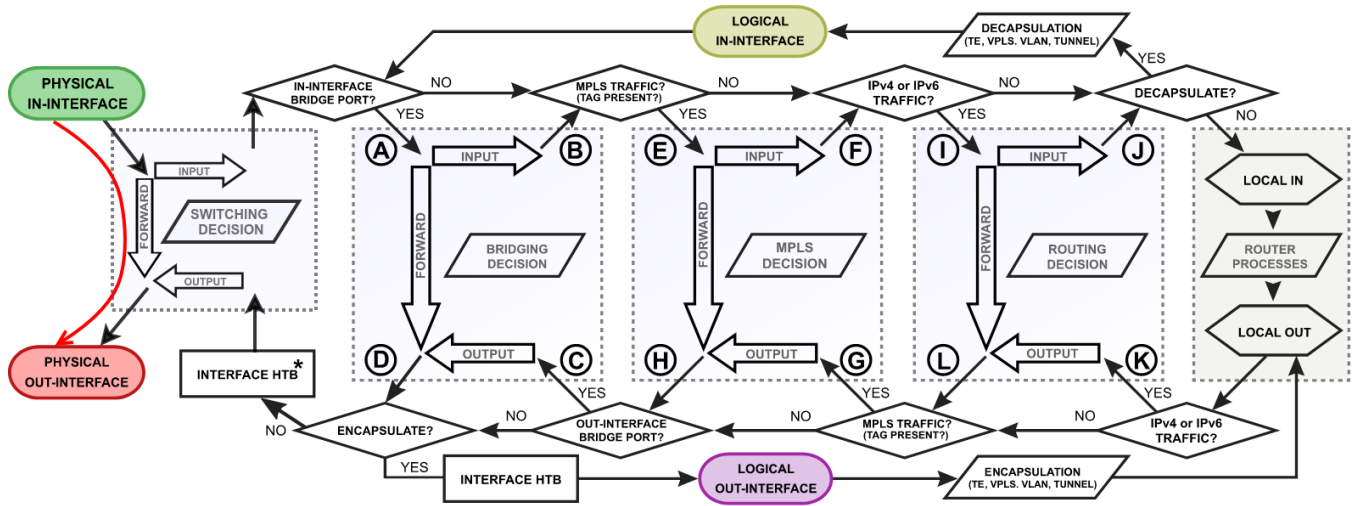
1. A packet goes through the bridge NAT dst-nat chain;
2. With the use-ip-firewall option enabled, the packet will be further processed in the prerouting chain;
3. A packet enters prerouting processing;
4. Run packet through the bridge host table to make forwarding decision;
5. A packet goes through the bridge filter forward chain;

6. With the use-ip-firewall option enabled, the packet will be further processed in the routing forward chain;
7. A packet enters routing forward processing;
8. A packet goes through the bridge NAT src-nat chain;
9. With the use-ip-firewall option enabled, the packet will be further processed in the postrouting chain;
10. A packet enters postrouting processing;

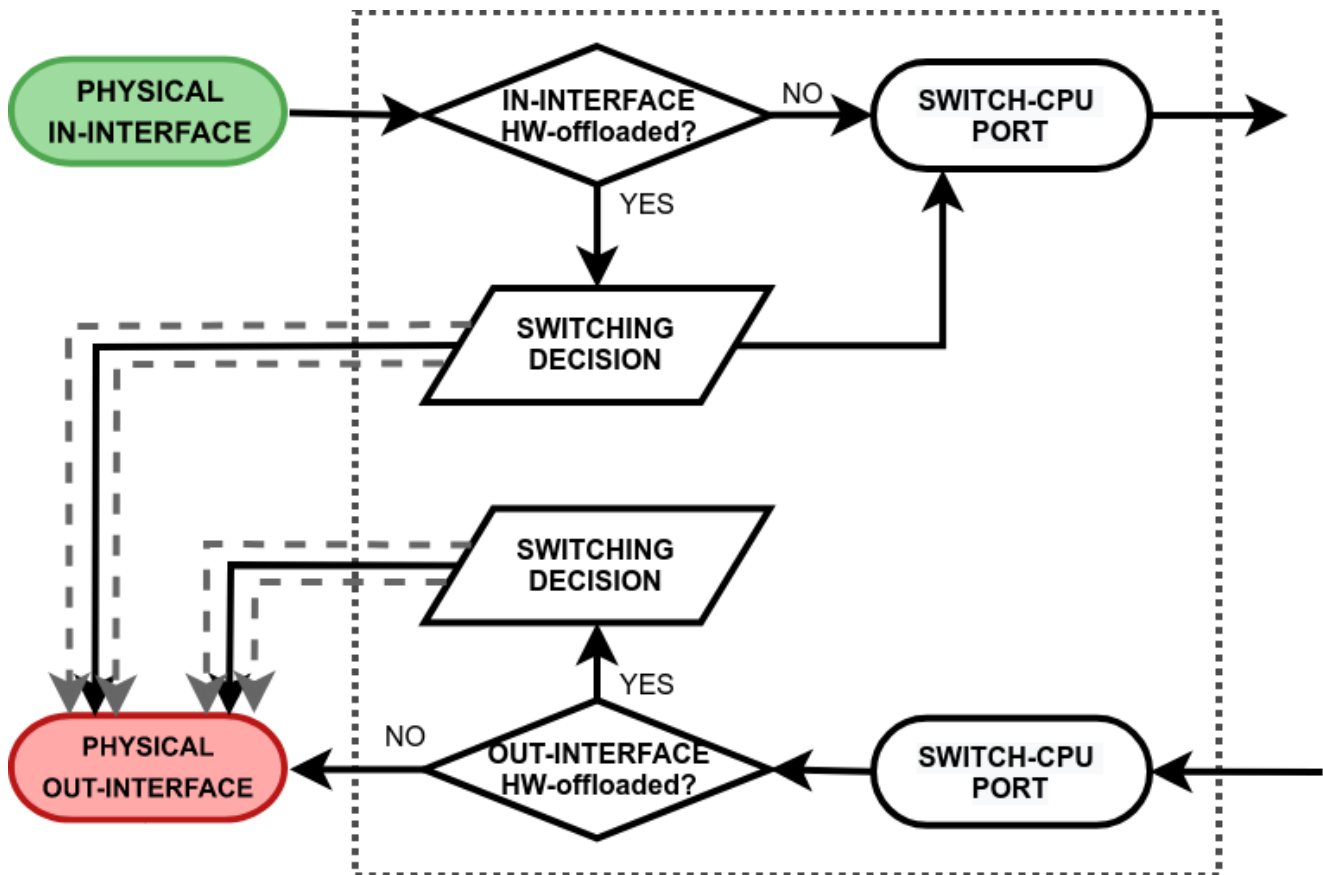


Flow of Hardware Offloaded Packet

On the previous topic, we solely discussed a software bridging that requires the main CPU processing to forward packets through the correct bridge port. Most of the MikroTik devices are equipped with dedicated switching hardware, the so-called switch chip or switch ASIC. This allows us to offload some of the bridging functions, like packet forwarding between bridge ports or packet filtering, to this specialized hardware chip without consuming any CPU resources. In RouterOS, we have named this function Bridge Hardware (HW) Offloading. Different MikroTik devices might have different switch chips and each chip has a different set of features available, so make sure to visit this article to get more details - [Bridge Hardware Offloading](#).



Interface HTB will not work correctly when the out-interface is hardware offloaded and the bridge Fast Path is not active.



- **switching decision** - widely depends on the switch model. This block controls all the switching-related tasks, like host learning, packet forwarding, filtering, VLAN tagging/untagging, etc. Certain switch configurations can alter the packet flow;
- **switch-cpu port** - a special purpose switch port for communication between the main CPU and other switch ports. Note that the switch-cpu port does not show up anywhere on RouterOS except for the switch menu, none of the software-related configurations (e.g. interface-list) can be applied to this port. Packets that reach the CPU are automatically associated with the physical in-interface.

The hardware offloading, however, does not restrict a device to only hardware limited features, rather it is possible to take advantage of the hardware and software processing at the same time. This does require a profound understanding of how packets travel through the switch chip and when exactly they are passed to the main CPU.

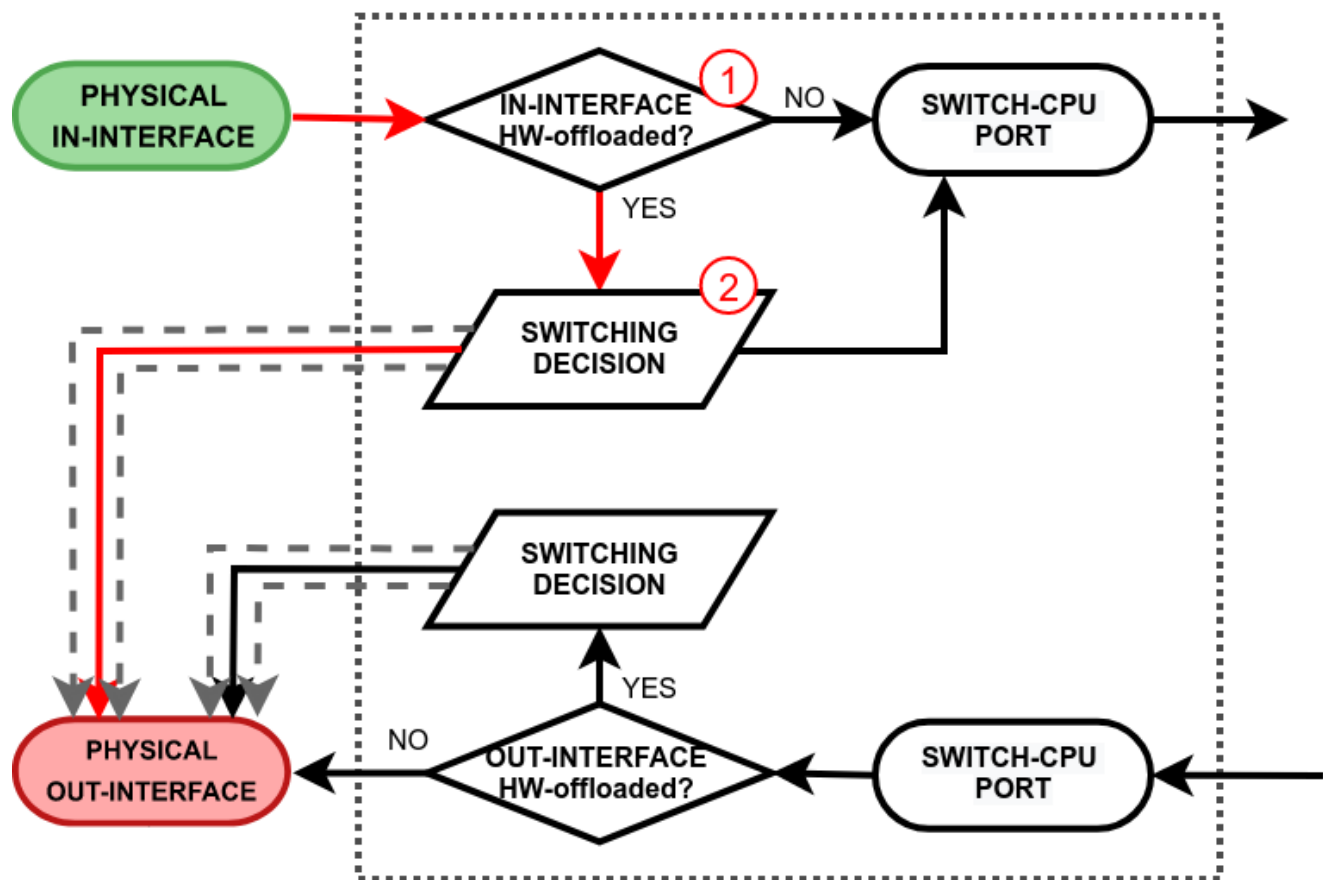


Switch features found in the "/interface/ethernet/switch" menu and its sub-menus, like ACL rules, mirroring, ingress/egress rate limiters, QoS, and L3HW (except inter-VLAN routing) may not rely on bridge hardware offloading. Therefore, they can **potentially** be applied to interfaces not configured within a hardware-offloaded bridge.

Switch Forward

We will further discuss a packet flow when bridge hardware offloading is enabled and a packet is forwarded between two switched ports on a single switch chip. This is the most common and also the simplest example:

1. The switch checks whether the in-interface is a hardware offloaded interface;
2. Run a packet through the switch host table to make a forwarding decision. If the switch finds a match for the destination MAC address, the packet is sent out through the physical interface. A packet that ends up being flooded (e.g. broadcast, multicast, unknown unicast traffic) gets multiplied and sent out to every hardware offloaded switch port.



Switch to CPU Input

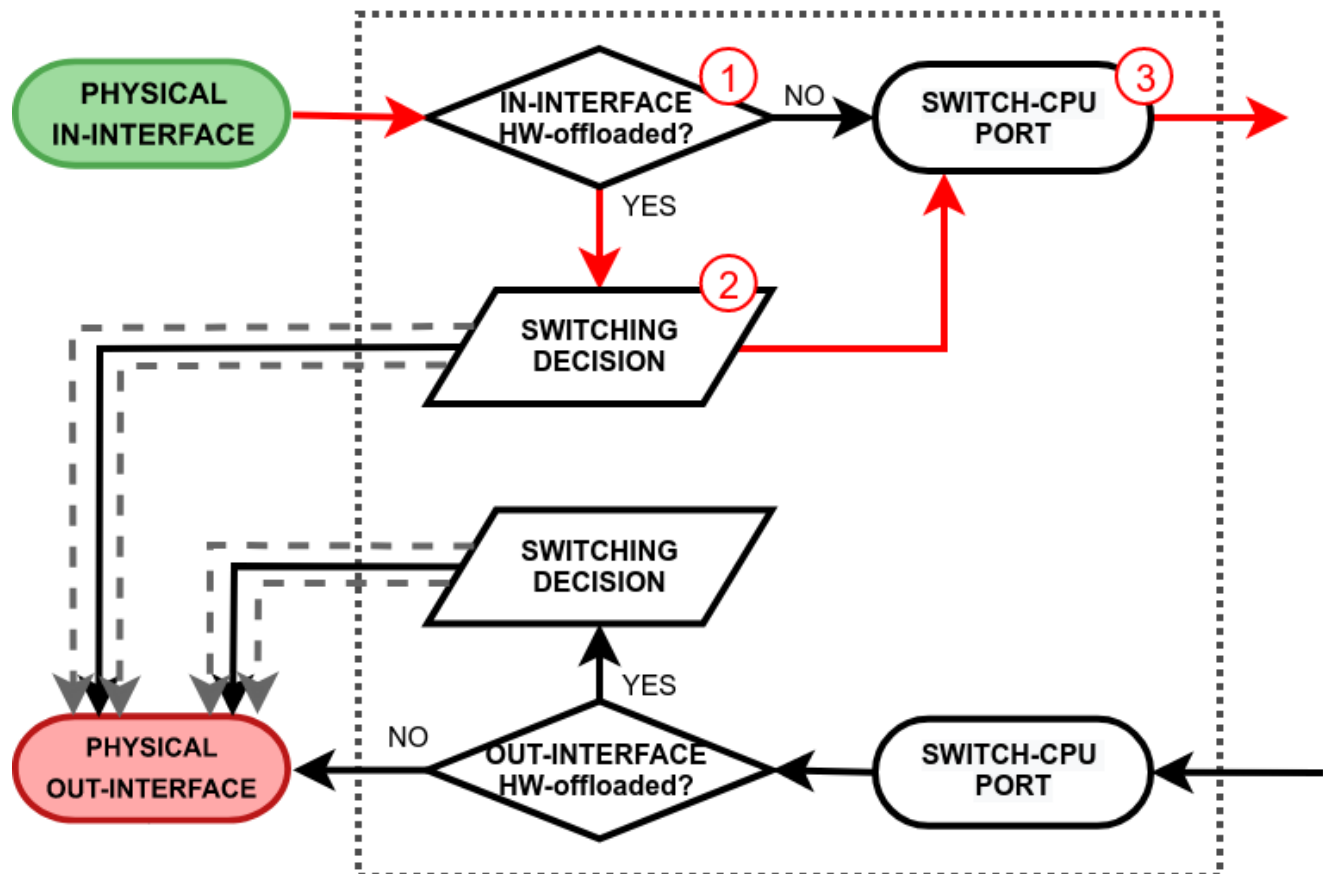
This process takes place when a packet is received on a physical interface and it is destined to switch-cpu port for further software processing. There are two paths to the switch-cpu. One where hardware offloading and switching is not even used (e.g. a standalone interface for routing or a bridged interface but with deliberately disabled HW offloading), so the packet is simply passed further for software processing. Another path is taken when hardware offloading is active on the in-interface. This will cause the packet to pass through the switching decision and there are various reasons why the switch might forward the packet to the switch-cpu port:

- a packet's destination MAC address match with a local MAC address, e.g. when a packet is destined to a local bridge interface;
- a packet might get flooded to all switch ports including the switch-cpu port, e.g. when broadcast, multicast, or unknown unicast traffic is received;

- a switch might have learned that some hosts can only be reached through the CPU (switch-cpu port learning is discussed in the next section), e.g. when a bridge contains HW and non-HW offloaded interfaces, such as wireless, EoIP, and even Ethernet interfaces;
- a packet is intentionally copied to the switch-cpu, e.g. for a packet inspection;
- a packet is triggered by the switch configuration and should be processed in software, e.g. a DHCP or IGMP snooping.

See the packet walkthrough when an in-interface is hardware offloaded:

1. The switch checks whether the in-interface is a hardware offloaded interface;
2. Run a packet through the switch host table to make a forwarding decision. In case any of the above-mentioned points are true, the packet gets forwarded to the switch-cpu port.
3. The packet exits through the switch-cpu port and it will be further processed by the RouterOS packet flow.



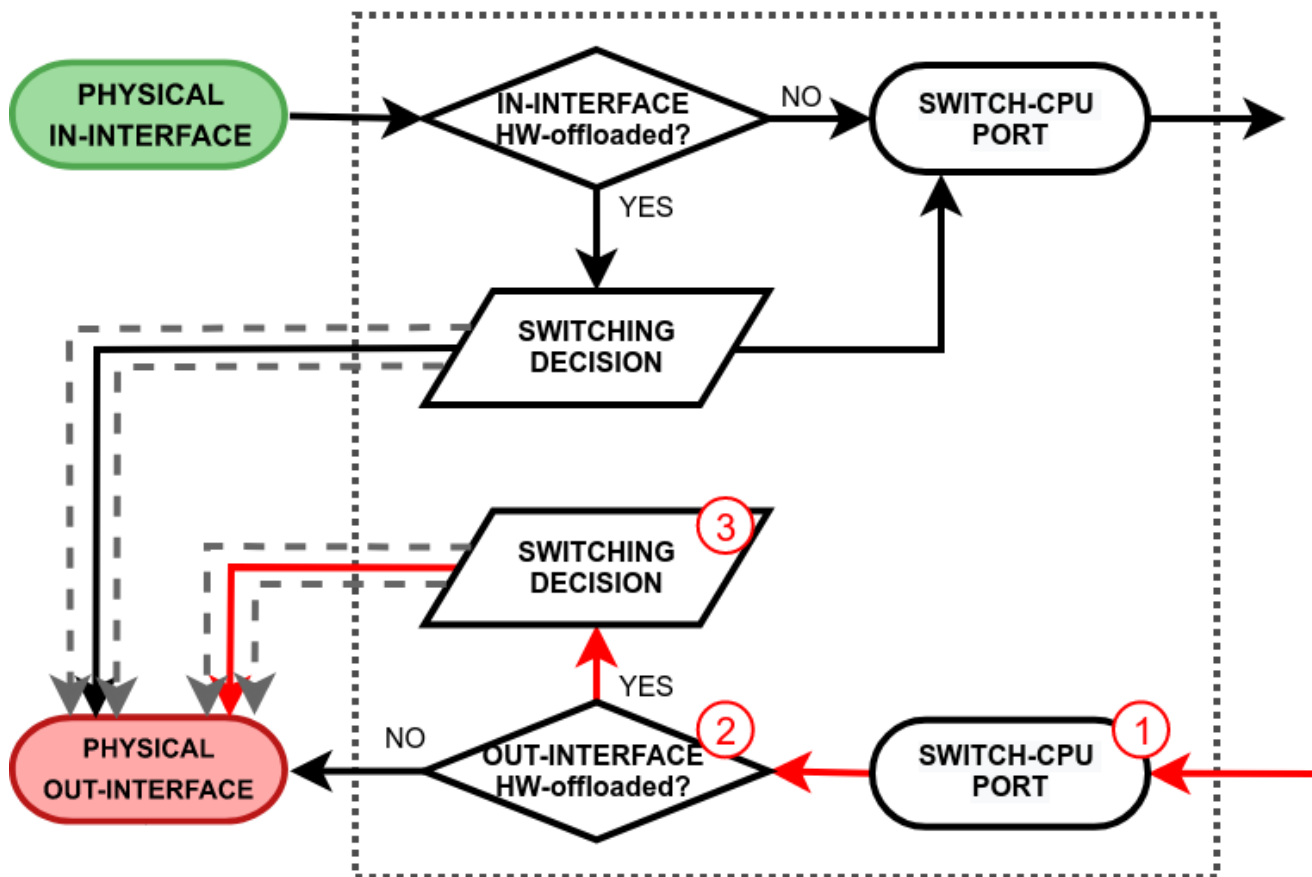
Any received packet that was flooded by the switch chip will not get flooded again by the software bridge to the same HW offloaded switch group. This prevents the formation of duplicate packets.

CPU Output to Switch

This process takes place when a packet exits the RouterOS software processing and is received on the switch-cpu port. Again, there are two paths the packet can take. One where hardware offloading and switching are not even used (e.g. a standalone interface for routing or a bridged interface but with deliberately disabled HW offloading), so the packet is simply sent out through the physical out-interface. Another path is taken when hardware offloading is active on the out-interface. This will cause the packet to pass through the switching decision. Just like any other switch port, the switch will learn the source MAC addresses from packets that are received on the switch-cpu port. This does come in handy when a bridge contains HW and non-HW offloaded interfaces, so the switch can learn which frames should be forwarded to the CPU. See the packet walkthrough when an out-interface is hardware offloaded:

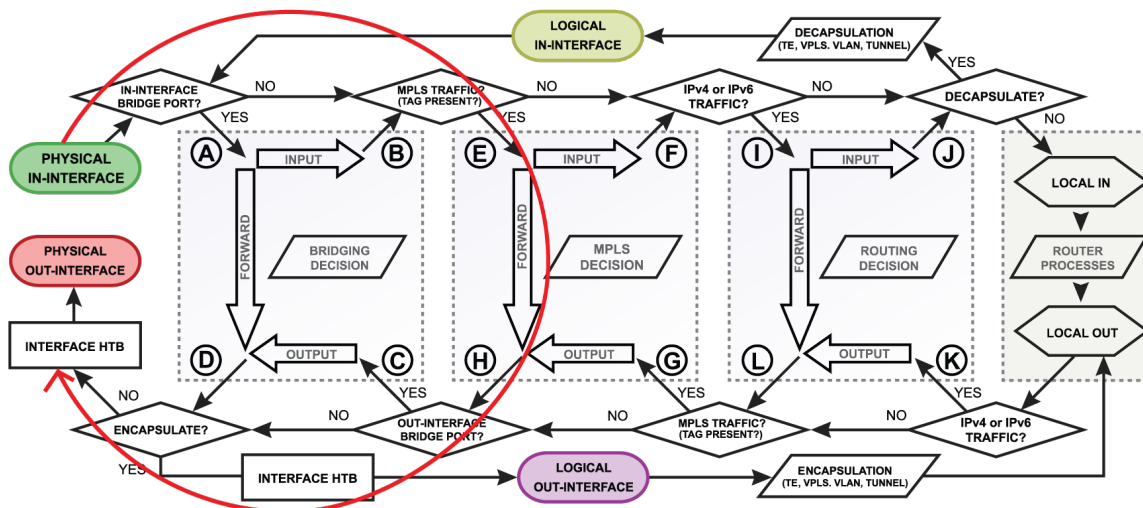
1. A packet that exits the RouterOS software processing is received on the switch-cpu port;
2. The switch checks whether the out-interface is a hardware offloaded interface;
3. Run a packet through the switch host table to make a forwarding decision. If the switch finds a match for the destination MAC

address, the packet is sent out through the physical interface. A packet that ends up being flooded (e.g. broadcast, multicast, unknown unicast traffic) gets multiplied and sent out to every hardware offloaded switch port.

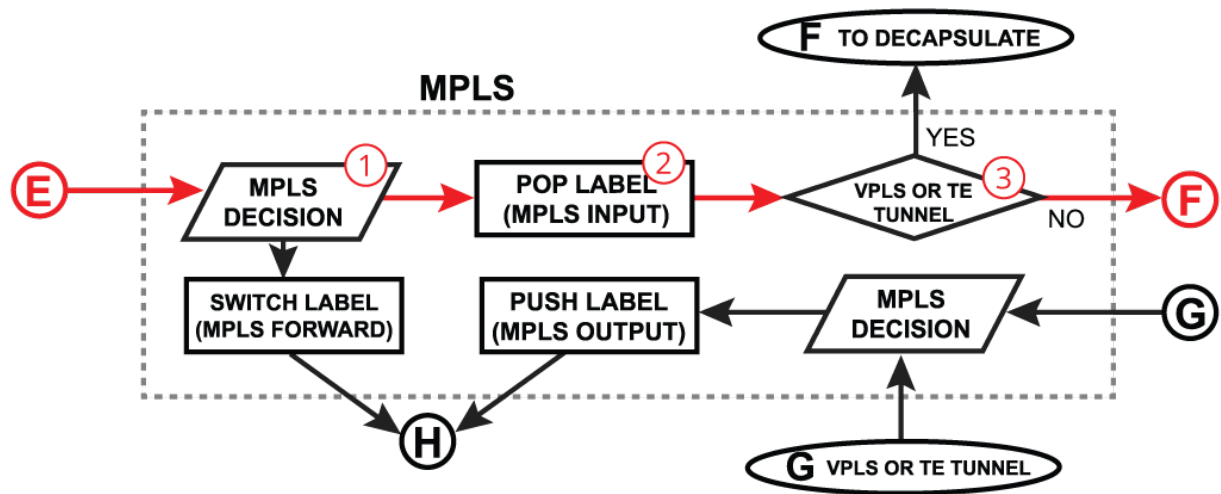


A software bridge that sends a flooded packet through HW offloaded interfaces, will only send a single packet copy per HW offloaded switch group rather than per HW offloaded interface. The actual flooding will be done by the switch chip, this prevents the formation of duplicate packets.

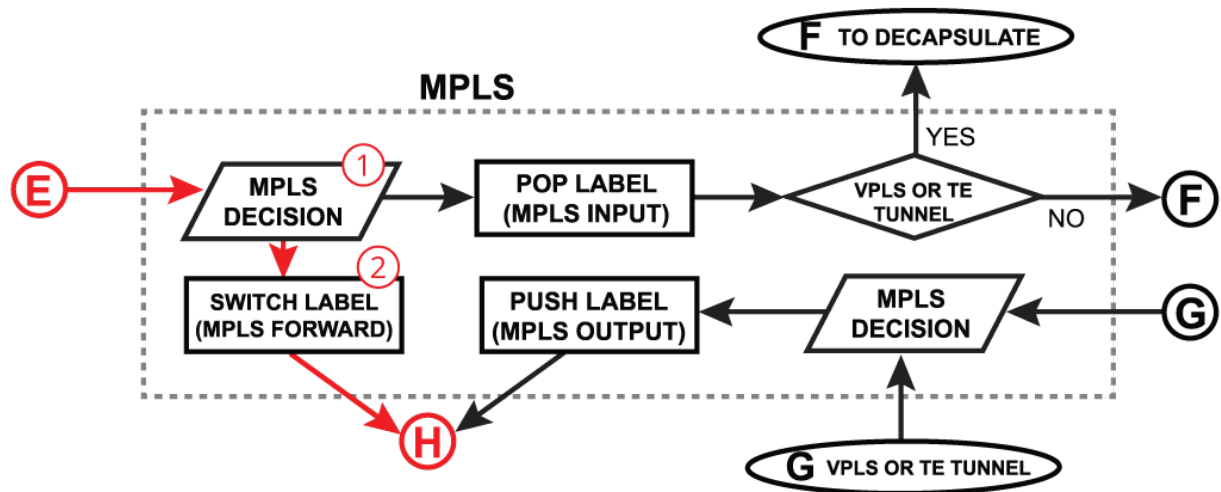
Flow of MPLS Packet



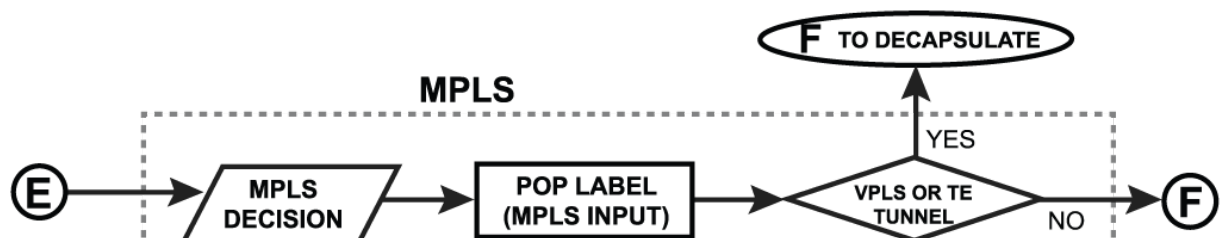
Pop Label

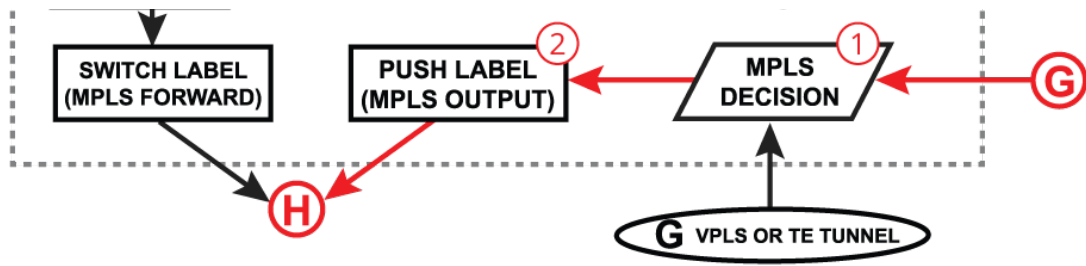


Switch Label



Push Label





MPLS IP VPN

In VPNv4 setups packet arriving at PE router that needs to be forwarded to the CE router is not a typical "forward".

If incoming label and destination is bound to the VRF Then after MPLS label is popped and:

- destination address is local to the router, then packet is moved to LOCAL_IN
- destination address is in the CE network, then packet is moved to LOCAL_OUT

⚠ Forwarded packets from MPLS cloud to the CE network will not show up in the forward.

For example traffic from src:111.15.0.1 to dst:111.13.0.1

```
[admin@CCR2004_2XS] /mpls/forwarding-table> print
Flags: L - LDP, P - VPN
Columns: LABEL, VRF, PREFIX, NEXTHOPS
# LABEL VRF PREFIX
NEXTHOPS

0 P 17 myVrf 111.13.0.0/24
4 L 20 main 203.0.113.2 { label=impl-null; nh=111.11.0.1; interface=sfp-sfpplus1 }
[admin@CCR2004_2XS] /mpls/forwarding-table>
...

[admin@CCR2004_2XS] /ip/route> print detail
Flags: D - dynamic; X - disabled, I - inactive, A - active; c - connect, s - static, r - rip, b - bgp, o -
ospf, i - is-is, d - dhcp, v - vpn, m - modem, y - bgp-mpls-vpn;
H - hw-offloaded; + - ecmp

DAc dst-address=111.11.0.0/24 routing-table=main gateway=sfp-sfpplus1 immediate-gw=sfp-sfpplus1 distance=0
scope=10 suppress-hw-offload=no
local-address=111.11.0.2@sfp-sfpplus1
DAc dst-address=203.0.113.1/32 routing-table=main gateway=lo immediate-gw=lo distance=0 scope=10 suppress-
hw-offload=no local-address=203.0.113.1%lo
DAo dst-address=203.0.113.2/32 routing-table=main gateway=111.11.0.1@sfp-sfpplus1 immediate-gw=111.11.0.1%
sfp-sfpplus1 distance=110 scope=20 target-scope=10
suppress-hw-offload=no
DAc dst-address=111.13.0.0/24 routing-table=myVrf gateway=sfp-sfpplus2@myVrf immediate-gw=sfp-sfpplus2
distance=0 scope=10 suppress-hw-offload=no
local-address=111.13.0.2@sfp-sfpplus2@myVrf
DAy dst-address=111.15.0.0/24 routing-table=myVrf gateway=203.0.113.2 immediate-gw=111.11.0.1@sfp-sfpplus1
distance=200 scope=40 target-scope=30 suppress-hw-offload=no [admin@CCR2004_2XS] /ip/route>
```

Packet will be seen in the output and postrouting chains, because now it is locally originated packet with source MAC address equal to vrfInterface:

```
08:10:55 firewall,info output: in:(unknown 0) out:sfp-sfpplus2, connection-state:established src-mac f2:b5:e9:
17:18:3b, proto ICMP (type 8, code 0), 111.15.0.1->111.13.0.1, len 56
08:10:55 firewall,info postrouting: in:myVrf out:sfp-sfpplus2, connection-state:established src-mac f2:b5:e9:17:
18:3b, proto ICMP (type 8, code 0), 111.15.0.1->111.13.0.1, len 56
```

```
08:10:55 firewall,info prerouting: in:sfp-sfpplus2 out:(unknown 0), connection-state:established src-mac dc:2c:6e:46:f8:93, proto ICMP (type 0, code 0), 111.13.0.1->111.15.0.1, len 56
08:10:55 firewall,info forward: in:sfp-sfpplus2 out:sfp-sfpplus1, connection-state:established src-mac dc:2c:6e:46:f8:93, proto ICMP (type 0, code 0), 111.13.0.1->111.15.0.1, len 56
08:10:55 firewall,info postrouting: in:sfp-sfpplus2 out:sfp-sfpplus1, connection-state:established src-mac dc:2c:6e:46:f8:93, proto ICMP (type 0, code 0), 111.13.0.1->111.15.0.1, len 56
```

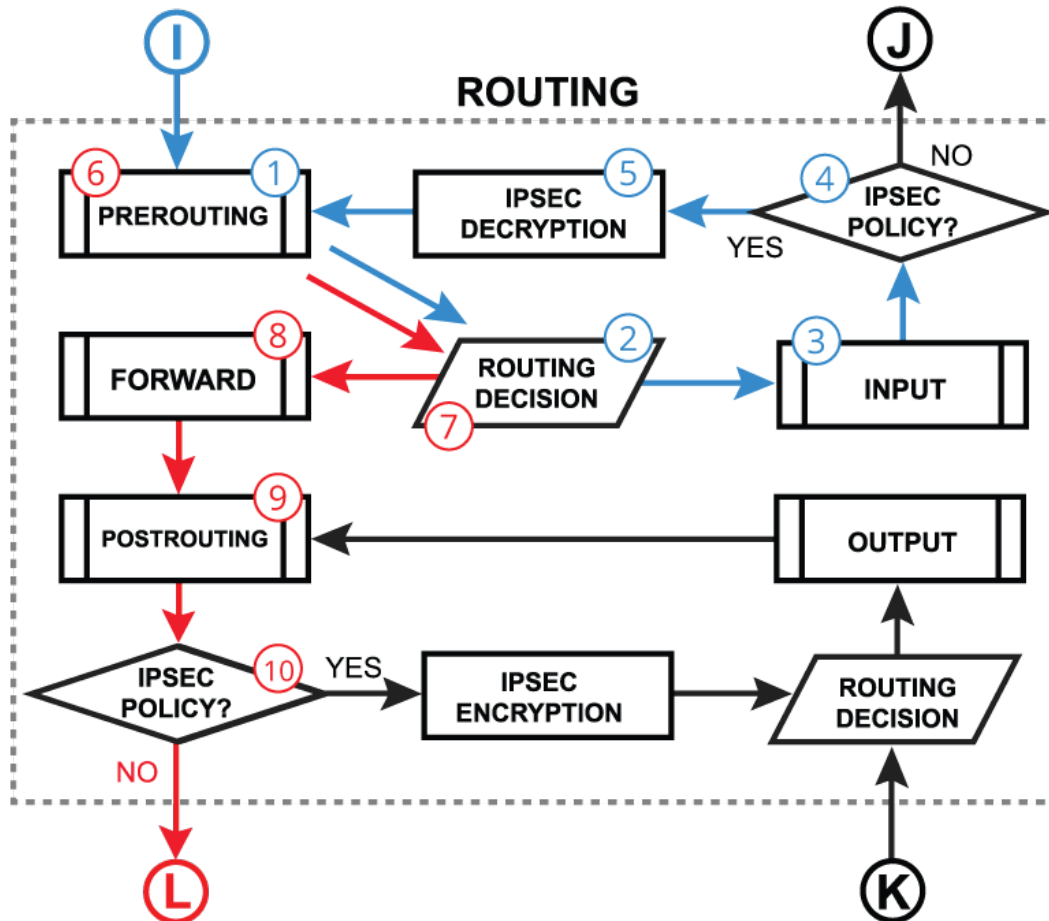
Logical Interfaces

[illegible]

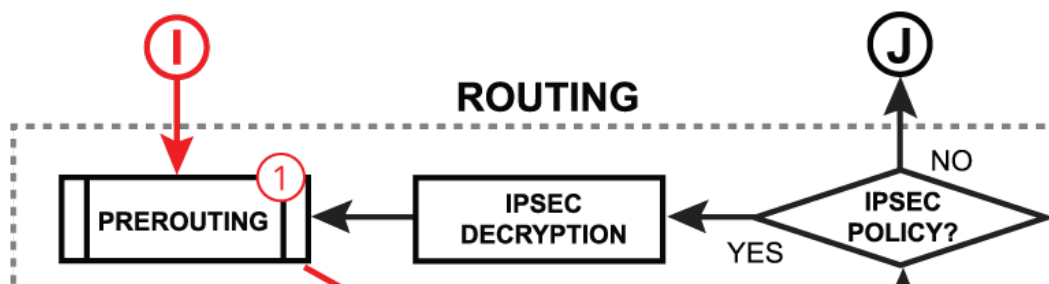
 Packet encapsulation and decapsulation using a bridge with enabled `vlan-filtering` do not relate to logical interfaces. See more details in the bridging section.

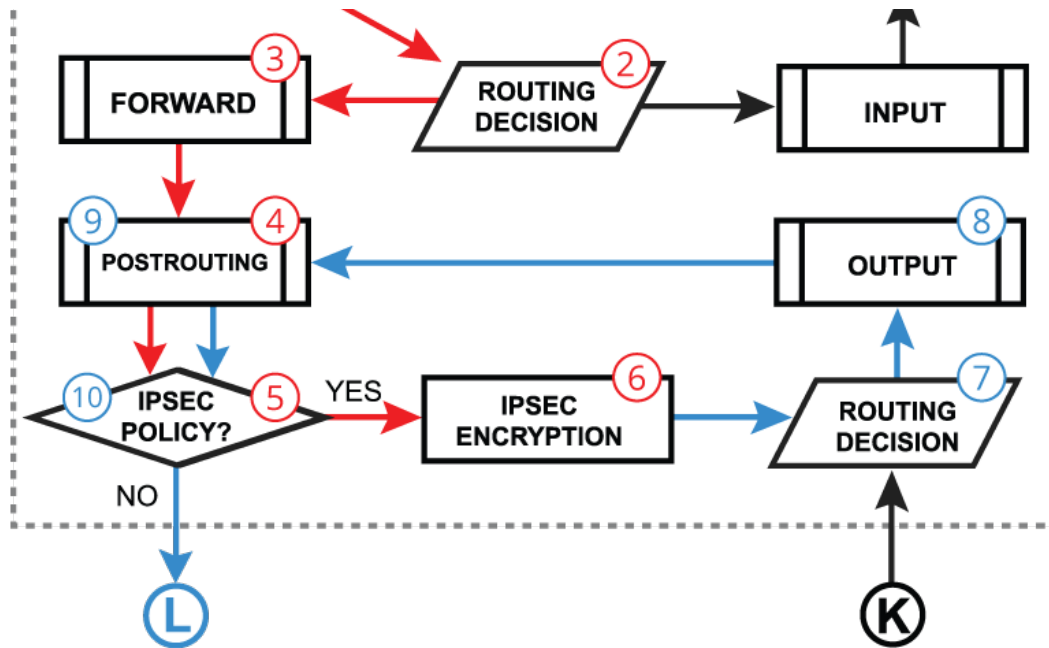
Instead of logical interfaces packets are processed through IPSec policies. After routing decision (2) and input firewall processing (3), the router tries to match the source and

The same process is with encapsulation but in reverse order. The first IP packet gets processed through facilities, then matched against IPsec policies (5), encapsulated (6), and then sent to processing on the second loop (7-10).



The same process is with encapsulation but in reverse order. The first IP packet gets processed through facilities, then matched against IPsec policies (5), encapsulated (6), and then sent to processing on the second loop (7-10).





Fast Path

From what we learned so far, it is quite obvious that such packet processing takes a lot of CPU resources. To fast things up FastPath was introduced in the first RouterOS v6. What it does is it skips processing in the Linux kernel, basically trading some RouterOS functionality for performance. For FastPath to work, interface driver support and specific configuration conditions are required.

How Fast Path Works

FastPath is an interface driver extension, that allows a driver to talk directly to specific RouterOS facilities and skip all others.



The packet can be forwarded by a fast path handler only if at least the source interface supports a fast path. For complete fast-forwarding, destination interface support is also required.


Currently, RouterOS has the following FastPath handlers:

- IP traffic processing
- FastTrack
- Traffic Generator
- MPLS
- Bridge

FastPath handler is used if the following conditions are met:

- firewall rules are not configured;
- simple queue or queue trees with *parent=global* are not configured;
- no mesh, metarouter interface configuration;
- sniffer or torch is not running;
- connection tracking is not active;
- IP accounting is disabled;
- VRFs are not configured (`/ip route vrf` is empty);

- A hotspot is not used (/ip hotspot has no interfaces);
- IPSec policies are not configured;
- /tool mac-scan is not actively used;
- /tool ip-scan is not actively used.


 Packets will travel the FastPath way if FastTrack is used no matter if the above conditions are met.

Traffic Generator automatically use FastPath if the interface supports this feature.

Currently, MPLS fast-path applies to MPLS switched traffic (frames that enter router as MPLS and must leave router as MPLS) and VPLS endpoint that do VPLS encap/decap. Other MPLS ingress and egress will operate as before.

A Bridge handler is used if the following conditions are met:

- there are no bridge Calea, filter, NAT rules;
- use-ip-firewall is disabled;
- no mesh, MetaRouter interface configuration;
- sniffer, torch, and traffic generator are not running;
- bridge vlan-filtering is disabled (condition is removed since RouterOS 7.2 version);
- bridge dhcp-snooping is disabled.

 FastPath on the vlan-filtering bridge does NOT support priority-tagged packets (packets with VLAN header but VLAN ID = 0). Those packets are redirected via a slow path.

Interfaces that support FastPath:

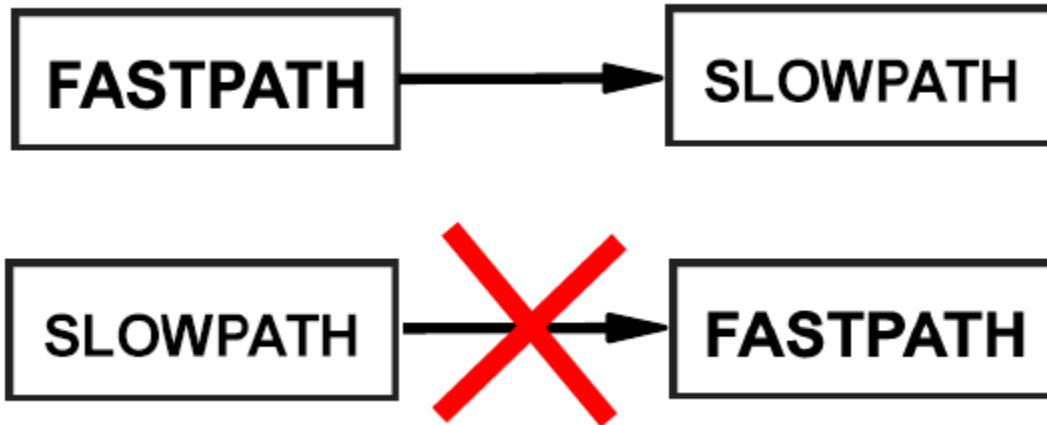
RouterBoard	Interfaces
RB6xx series	ether1,2
RB800	ether1,2
RB1100 series	ether1-11
All devices	Ethernet interfaces
	wireless interfaces
	bridge interfaces
	VLAN, VRRP interfaces
	bonding interfaces (RX only)
	PPPoE, L2TP interfaces
	EoIP, GRE, IPIP, VXLAN interfaces.
	VPLS (starting from v7.17)

EoIP, Gre, IPIP, VXLAN and L2TP interfaces have per-interface setting *allow-fast-path*. Allowing a fast path on these interfaces has a side effect of bypassing firewall, connection tracking, simple queues, queue tree with parent=global, IP accounting, IPSec, hotspot universal client, vrf assignment for encapsulated packets that go through a fast-path. Also, packet fragments cannot be received in FastPath.

 Whether FastPath is being used can be verified with `/interface print stats-detail`

Only interface queue that guarantees FastPath is only-hardware-queue. If you need an interface queue other than hardware then the packet will not go fully FastPath, but there is not a big impact on performance, as "interface queue" is the last step in the packet flow.

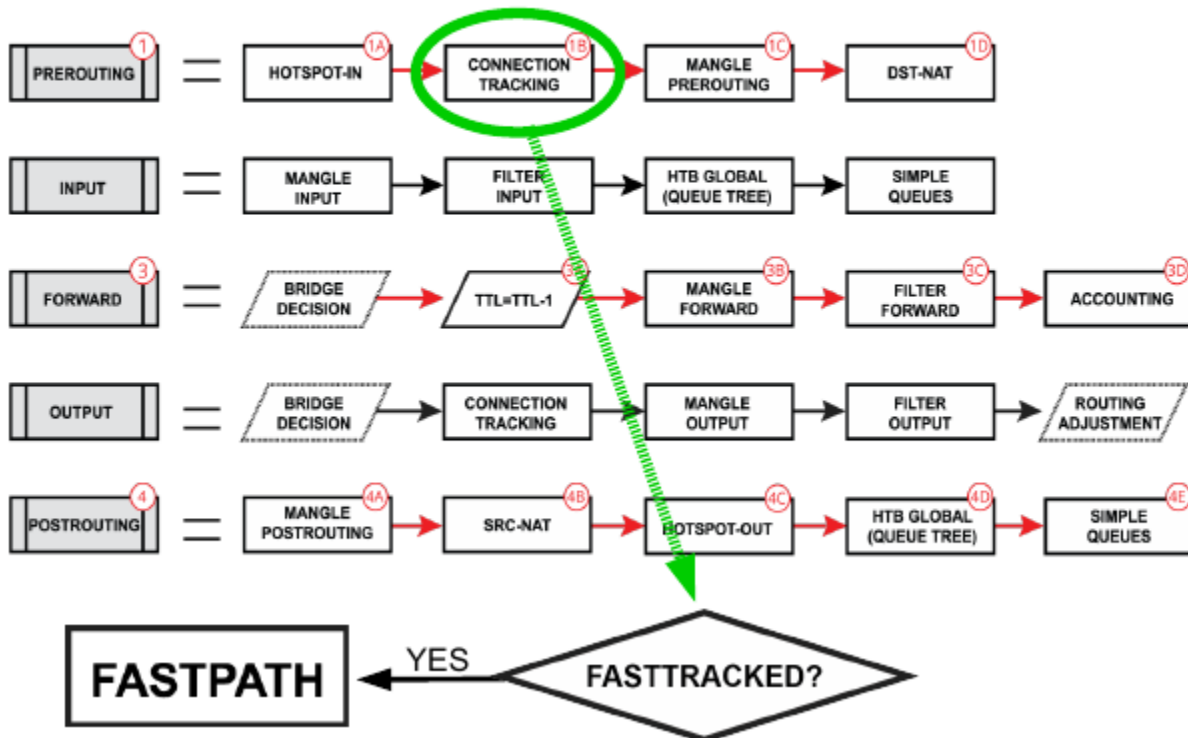
The packet may go Half-FastPath by switching from FastPath to SlowPath, but not the other way around. So, for example, if the receiving interface has FastPath support, but the out interface does not, then the router will process the packet by FastPath handlers as far as it can and then proceed with SlowPath. If the receiving interface does not support FastPath but the out interface does, the packet will be processed by SlowPath all the way through the router.



FastTrack

Fasttrack can be decoded as Fast Path + Connection Tracking. It allows marking connections as "fast-tracked", marking packets that belong to fast-tracked connection will be sent fast-path way. The connection table entry for such a connection now will have a fast-tracked flag.


Routing Forwarding FastPath



FastTrack packets bypass firewall, connection tracking, simple queues, queue tree with parent=global, ip traffic-flow, IP accounting, IPSec, hotspot universal client, VRF assignment, so it is up to the administrator to make sure FastTrack does not interfere with other configuration!

To mark a connection as fast-tracked new action was implemented "*fasttrack-connection*" for firewall filter and mangle. Currently, only TCP and UDP connections can be fast-tracked and to maintain connection tracking entries some random packets will still be sent to a slow path. This must be taken into consideration when designing firewalls with enabled "fasttrack".

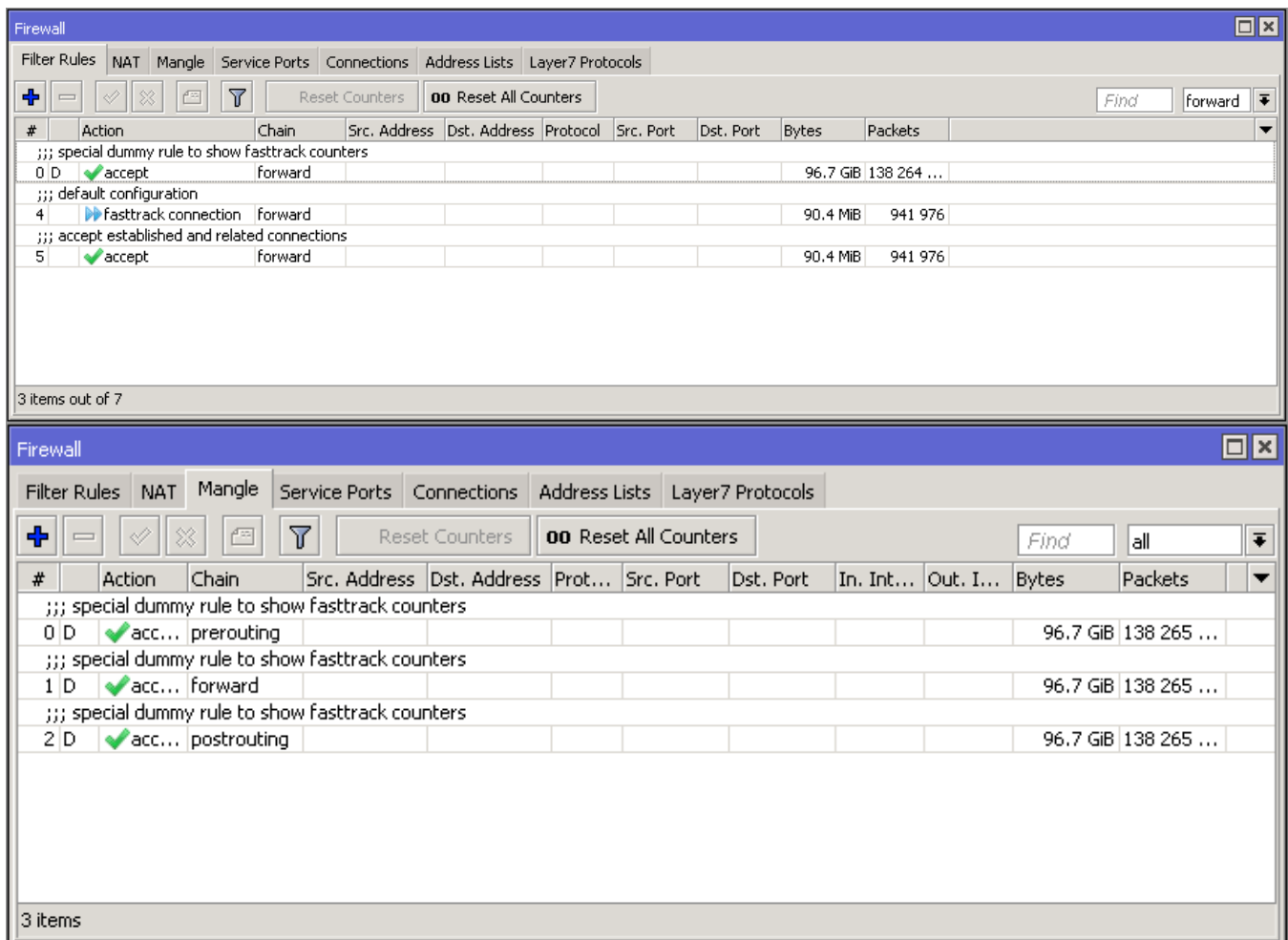
FastTrack handler also supports source and destination NAT, so special exceptions for NATed connections are not required.

 Traffic that belongs to a fast-tracked connection travels in FastPath, which means that it will not be visible by other router L3 facilities (firewall, queues, IPsec, IP accounting, VRF assignment, etc). Fasttrack lookups route before routing marks have been set, so it works only with the main routing table.

The easiest way to start using this feature on home routers is to enable "fasttrack" for all *established, related* connections:

```
/ip firewall filter
add chain=forward action=fasttrack-connection connection-state=established,related \
    comment="fasttrack established/related"
add chain=forward action=accept connection-state=established,related \
    comment="accept established/related"
```

Notice that the first rule marks established/related connections as fast-tracked, the second rule is still required to accept packets belonging to those connections. The reason for this is that, as was mentioned earlier, some random packets from fast-tracked connections are still sent the slow pathway and only UDP and TCP are fast-tracked, but we still want to accept packets for other protocols.



#	Action	Chain	Src. Address	Dst. Address	Protocol	Src. Port	Dst. Port	Bytes	Packets
;;; special dummy rule to show fasttrack counters									
0	D ✓ accept	forward						96.7 GiB	138 264 ...
;;; default configuration									
4	Fasttrack connection	forward						90.4 MiB	941 976
;;; accept established and related connections									
5	✓ accept	forward						90.4 MiB	941 976

#	Action	Chain	Src. Address	Dst. Address	Prot...	Src. Port	Dst. Port	In. Int...	Out. I...	Bytes	Packets
;;; special dummy rule to show fasttrack counters											
0	D ✓ acc...	prerouting								96.7 GiB	138 265 ...
;;; special dummy rule to show fasttrack counters											
1	D ✓ acc...	forward								96.7 GiB	138 265 ...
;;; special dummy rule to show fasttrack counters											
2	D ✓ acc...	postrouting								96.7 GiB	138 265 ...

After adding the "FastTrack" rule special dummy rule appeared at the top of the list. This is not an actual rule, it is for visual information showing that some of the traffic is traveling FastPath and will not reach other firewall rules.



FastTrack can process packets only in the main routing table so it is the system administrator duty to not FastTrack connections that are going through non-main routing table (thus connections that are processed with mangle action=mark-routing rules). Otherwise packets might be misrouted though the main routing table.

These rules appear as soon as there is at least one fast-tracked connection tracking entry and will disappear after the last fast-tracked connection times out in the connection table.



The connection is FastTracked until a connection is closed, timed out or the router is rebooted.

Configuration example: excluding specific host, from being Fast-Trackd

```
/ip firewall filter
add action=accept chain=forward connection-state=established,related src-address=192.168.88.111
add action=accept chain=forward connection-state=established,related dst-address=192.168.88.111
add action=fasttrack-connection chain=forward connection-state=established,related hw-offload=no
add action=accept chain=forward connection-state=established,related
```

In this example, we exclude host 192.168.88.111, from being Fast-tracked, by first accepting it with the firewall rule, both for source and destination. The idea is - not allowing the traffic to reach the FastTrack action.

Note: the "exclusion" rules, must be placed before fasttrack filters, order is important.

Requirements

FastTrack is active if the following conditions are met:

- no mesh, metarouter interface configuration;
- sniffer, torch, and traffic generator are not running;
- "/tool mac-scan" is not actively used;
- "/tool ip-scan" is not actively used;
- FastPath and Route cache are enabled under IP/Settings (route cache condition does not apply to RouterOS v7 or newer);
- bridge FastPath is enabled if a connection is going over the bridge interface;

Packet flow for the visually impaired

The following document in DOCX format describes the diagram in a way optimized for visually impaired people. The descriptions are by Apex CoVantage care of Benetech. They are not being updated.

- [Packet flow, optimized document.](#)