DebTik Network Appliance

Routing Filters, BGP Policies & Traffic Engineering Manual

Software Version: 1.3.11+ | Routing Engine: BIRD 2 (v2.17+) | Syntax Layer: MikroTik RouterOS v7 Scripting Layer | Date: November 2026

1. Introduction & Core Architecture

Routing Filters in DebTik serve as the central policy enforcement mechanism for dynamic routing protocols (principally BGP and OSPF). The underlying implementation is powered by the industry-standard, high-performance BIRD 2 routing daemon, coupled with an intuitive graphical interface and a syntax compatibility layer inspired by MikroTik RouterOS v7.

The policy engine operates according to the following foundational principles:

- Chains: Filter rules are organized into logical chains (e.g., Internal, bgp-in, bgp-out). Each chain compiles in BIRD 2 into an procedural function function fn_filter_<Chain>() and a standalone filter filter filter_<Chain>.
- Sequential Evaluation (Rule Order): Rules execute strictly in ascending numerical order of their sequence number (#, e.g., 10, 20, 30). Lower numbers execute first.
- Terminal Actions: An explicit accept; or reject; action terminates filter evaluation immediately for that route. The decision is final.
- Non-Terminal Action (Passthrough): Allows modifying route attributes (Local Preference, Communities, Prepending, Metric) without halting evaluation, letting subsequent rules inspect the route.
- Default Deny Policy: If a candidate route traverses an entire chain without matching any rule (unmatched), it is automatically rejected at the end of the filter (reject;).

ARCHITECTURAL SAFEGUARD: Automated Export Protection (v1.3.11+): When an Out-Filter is attached to a BGP peer, DebTik ensures that explicitly declared Advertised Networks (Output Networks / Static Blackhole prefixes) and redistribution sources (Connected/Static) are safely preserved and never dropped by the terminal default reject, unless a user rule explicitly discards them.

DebTik Policy Engine Core Capabilities:

Feature / Capability	Description & Implementation Details
v7 Procedural Scripting	Native if (condition) { actions; } syntax supporting composite logical operators (&&, ||, !).
CIDR & Subnet Ranges	Prefix matching with variable length bounds (e.g., dst in 10.0.0.0/8{24,28}) and exact mask checks (net.len = 24).
BGP Traffic Engineering	Full control over BGP Local Preference, MED Metric, AS-Path Prepending, and BGP Community tagging.
Named Lists	Reusable named BGP Community Lists and AS-Path Lists to simplify complex filtering rules into concise definitions.
Visual Rule Wizard	Built-in modal form assistant with one-click Quick Templates for instant, error-free rule generation.

2. Web UI Structure & Management (Policy Filters)

Routing Filters are managed via the DebTik Web Dashboard at Routing → Policy Filters. The workspace consists of 3 dedicated tabs, a global management toolbar, and modal editors:

Tab Name	RouterOS Path	Description & BIRD 2 Backend Role
Rules	`/routing filter rule`	Executable policy rules. Defines match criteria and actions (Accept/Reject/Set). Compiles directly into BIRD 2 filter blocks.
Community Lists	`/routing filter community-list`	Named sets of BGP Communities (e.g., 65000:100). Compiles into define comm_<Name> = [...].
AS Lists	`/routing filter num-list`	Named lists of Autonomous System Numbers (ASNs / Ranges). Compiles into define aslist_<Name> = [...].

2.1 Toolbar Controls

- + New: Opens the modal creation dialog corresponding to the active tab.
- Enable / Disable: Instantly toggles the active state of selected rows. Marked with flag A (Active - Cyan) or X (Disabled - Rose).
- Remove: Permanently deletes the selected rule or list item.
- Chain Filter Dropdown: Filters table rows to display only rules belonging to a specific chain (e.g., Internal).
- Find... (Live Search): Instant text filtering matching across IP prefixes, script syntax, comments, and flags.
- Reorder Buttons (▲/▼): Directly shifts the sequence number (Rule Order) up or down to adjust evaluation priority.

2.2 Rule Modal Parameters

- Chain (Required): Alphanumeric identifier of the policy chain (e.g., bgp-in, bgp-out, Internal, isp-primary).
- Rule Order (Required): Integer execution sequence (1 - 65535). A step of 10 (10, 20, 30...) is recommended for future insertions.
- Comment (Optional): Descriptive note documenting the intent of the rule (flagged with C in table view).
- Disabled (Checkbox): Temporarily suspends rule evaluation without deleting configuration.

2.3 Visual Rule Wizard & Quick Templates

For ease of use, each rule editor features a collapsible Visual Rule Wizard that builds valid script code automatically:

- Prefix (CIDR): Network IP prefix (e.g., 192.168.0.0/16 or 0.0.0.0/0).
- Prefix Length: Netmask length boundary (e.g., 24-32 or ge 24).
- Protocol: Origin protocol selector: any, bgp, ospf, connected, static.
- Action: Policy disposition: accept (permit), reject (deny), passthrough (modify & fall through).
- Set Local-Pref / Prepend: Numeric inputs for BGP Local Preference and AS-Path Prepend count.
- Generate Script Expression ↑: Transcribes the visual form selections directly into formatted v7 script syntax.

Available Quick Templates:

- 0.0.0.0/0: Matches default route • Drop RFC1918: Drops 10/8, 172.16/12, 192.168/16 • Len ≤ /24: Enforces standard ≤ /24 global prefix limit • Loc-Pref 150: Sets local preference 150 • AS-Prepend 2x: Prepends local AS twice • Reject: Unconditional drop.

3. Script Syntax Reference & Parameter Catalog

DebTik dynamically compiles MikroTik RouterOS v7 procedural rule expressions into native BIRD 2 filter syntax. The following tables provide an exhaustive reference for all supported match conditions and actions.

3.1 Match Conditions (Criteria)

RouterOS v7 Syntax	BIRD 2 Compiled Output	Description & Usage Example
`dst in A.B.C.D/M`	`net ~ [ A.B.C.D/M+ ]`	Tests if prefix belongs to subnet. E.g., dst in 10.0.0.0/8 (matches /8 through /32).
`dst in A.B.C.D/M{L,H}`	`net ~ [ A.B.C.D/M{L,H} ]`	Subnet match bounded by prefix lengths. E.g., dst in 192.168.0.0/16{24,28} (/24 to /28 only).
`dst == A.B.C.D/M` `or dst = A.B.C.D/M`	`net = A.B.C.D/M`	Exact prefix and netmask match. E.g., dst == 0.0.0.0/0 (default route only).
`dst != A.B.C.D/M`	`net != A.B.C.D/M`	Negative match; excludes a specific prefix.
`dst-len == N` `dst-len <= N, >= N`	`net.len = N` `net.len <= N, >= N`	Prefix length evaluation. E.g., net.len <= 24 prevents routes more specific than /24.
`protocol <name>`	`source = RTS_<NAME>`	Route origin check: bgp (RTS_BGP), ospf (RTS_OSPF), connected (RTS_DEVICE), static (RTS_STATIC).
`bgp-communities includes A:B`	`(A, B) ~ bgp_community`	Evaluates if a route contains a specific community. E.g., bgp-communities includes 65000:100.
`bgp-communities in ListName`	`bgp_community ~ comm_ListName`	Evaluates if any route community matches an entry in a defined Community List.
`bgp-as-path in ListName`	`bgp_path.last ~ aslist_ListName`	Evaluates if neighbor AS belongs to a defined AS List.
`bgp-as-path "pattern"`	`bgp_path ~ [= * pattern * =]`	Regular expression matching across the AS-Path. E.g., bgp-as-path "^65000_".
`&&, ||, !`	`&&, ||, !`	Logical conjunctions: AND (&&), OR (||), NOT (!).

3.2 Actions & Attribute Modifications

Action / Set Syntax	BIRD 2 Output	Routing Effect & Description
`accept;`	`accept;`	Terminal permit. Route is accepted and filter evaluation terminates immediately.
`reject; / discard;`	`reject;`	Terminal deny. Route is dropped and filter evaluation terminates immediately.
`passthrough;`	(non-terminal)	Non-terminal. Executes modification statements and falls through to next rule.
`set bgp-local-pref <N>;`	`bgp_local_pref = <N>;`	BGP Local Preference (default 100). Higher value = preferred outbound exit path.
`set bgp-med <N>;`	`bgp_med = <N>;`	BGP Multi-Exit Discriminator (MED). Lower value = suggested inbound path for neighbor AS.
`set bgp-path-prepend <N>;`	`bgp_path.prepend(...);`	Prepends local AS 1 to 10 times to discourage traffic over backup transit links.
`set bgp-communities.add <A:B>;`	`bgp_community.add((A,B));`	Appends a BGP Community to the route without overwriting existing tags.
`set bgp-communities.delete <A:B>;`	`bgp_community.delete((A,B));`	Removes a specific BGP Community from the route attribute set.
`set distance <N>;`	`preference = <N>;`	Administrative Distance / Preference for local routing table selection.
`set gw <IP>;`	`gw = <IP>;`	Forcibly overrides the Next-Hop IP gateway for the route.

4. Named Lists (Community Lists & AS Lists)

Named lists allow defining reusable sets of BGP Communities and Autonomous System numbers. Using named lists ensures policy rules remain compact, maintainable, and readable across multiple peers.

4.1 BGP Community Lists (/routing filter community-list)

BGP Communities are 32-bit values attached to routes to convey policy intents such as peer classification, traffic engineering, blackholing, or geographic scope.

- List Name: Unique identifier (e.g., AWMN_COMM, UPSTREAM_BLACKHOLE).
- Communities: Comma-separated list of community tags in AS:Value format or well-known keywords (no-export, blackhole). E.g., 65000:100, 6610:200, no-export.
- RegExp: Optional regular expression pattern for advanced matching.
- Comment & Disabled: Descriptive note and administrative state toggle.

Configuration & BIRD 2 Compilation Example:

```
# 1. Configured in DebTik Web UI (Community List: AWMN_COMM):
#    Name: AWMN_COMM | Communities: 65000:100, 6610:200
#
# 2. Generated BIRD 2 global definition in /etc/bird/bird.conf:
define comm_AWMN_COMM = [ (65000, 100), (6610, 200) ];

# 3. Usage inside filter rules (RouterOS v7 syntax):
if (bgp-communities in AWMN_COMM) {
    set bgp-local-pref 150;
    accept;
}
```

4.2 AS Number Lists (/routing filter num-list)

AS Lists define sets of Autonomous System Numbers (ASNs). They are commonly used to verify the origin AS or detect specific upstream transit carriers in an AS-Path:

- List Name: Unique list identifier (e.g., TIER1_ISPS, GRNET_PEERS).
- Range: Hyphenated ranges (e.g., 65000-65010) and discrete ASNs separated by commas (e.g., 6610, 17244, 3356).
- Comment & Disabled: Descriptive annotation and administrative state toggle.

Configuration & BIRD 2 Compilation Example:

```
# 1. Configured in DebTik Web UI (AS List: TIER1_ISPS):
#    Name: TIER1_ISPS | Range: 65000-65010, 6610, 17244, 3356
#
# 2. Generated BIRD 2 global definition in /etc/bird/bird.conf:
define aslist_TIER1_ISPS = [ 65000..65010, 6610, 17244, 3356 ];

# 3. Usage inside filter rules (checks neighbor/last AS in path):
if (bgp-as-path in TIER1_ISPS) {
    set bgp-local-pref 200;
    accept;
}
```

TECHNICAL NOTE: In BIRD 2, list constants (comm_... and aslist_...) are emitted at the top of /etc/bird/bird.conf before any filter functions or protocol definitions. This ensures they are universally accessible to all filter chains.

5. BGP Peering Integration (InFilter, OutFilter & Export Policy)

Policy chains are associated with BGP sessions via the peer configuration form (Routing → BGP Routing → Peers), using the In-Filter and Out-Filter fields.

5.1 In-Filter (Import Policy)

- Function: Evaluates incoming routes received from the peer before installation into master4.
- Processing: Every incoming BGP update is sequentially processed by the rules of the selected In-Filter chain.
- Decision: Routes hitting accept; are installed in the routing table. Routes hitting reject; or unmatched routes are discarded (Default Deny).
- Best Practice: In-Filters should always reject bogon/martian subnets and validate prefix lengths (≤ /24).

5.2 Out-Filter (Export Policy) & 3-Tier Safeguard Architecture

In BIRD 2, an Out-Filter processes all candidate routes in the routing table. To prevent unintentional route leaks while protecting locally originated networks, DebTik (v1.3.11+) employs a 3-tier evaluation hierarchy:

Priority Level	BIRD 2 Mechanism	Description & Behavioral Guarantee
Tier 1: User Policy	`fn_filter_<Chain>();`	Highest priority. User-defined chain rules execute first. An explicit reject; drops the route immediately. An explicit accept; exports the route immediately with all modifications applied.
Tier 2: Advertised Nets	`if proto = "bgp_nets_..." then accept;`	Automated protection. If a route was not explicitly matched by user rules, but was declared in the peer's Output Networks or global BGP Networks (Static blackhole), it is automatically accepted for advertisement.
Tier 3: Redistribution	`if source = RTS_DEVICE then accept;` `if source ~ [ RTS_STATIC ]...`	Redistribution. If the peer has Redistribute Connected or Redistribute Static checked, those routes are automatically accepted unless rejected in Tier 1.
Default Deny	`reject;`	Any remaining unmatched route is safely discarded, preventing unintended transit leaks.

Generated BIRD 2 Peer Export Configuration:

```
# Generated BIRD 2 block for peer with OutFilter: "Internal"
protocol bgp bgp_bgp_peer_1 {
    description "ebgp-internal-Primcast";
    local 192.168.129.226 as 65001;
    neighbor 192.168.129.225 as 65555;
    ipv4 {
        import filter filter_Internal;
        export filter {
            # Tier 1: User policy rules evaluation
            fn_filter_Internal();
            # Tier 2: Automated protection for declared peer & static networks
            if proto = "bgp_nets_bgp_bgp_peer_1" then accept;
            if proto = "bgp_static_nets" then accept;
            # Tier 3: Connected & Static route redistribution (if enabled)
            if source = RTS_DEVICE then accept;
            if source ~ [ RTS_STATIC, RTS_INHERIT ] then accept;
            # Default Deny
            reject;
        };
    };
}
```

6. Practical Recipes & Configuration Examples

The following battle-tested configuration patterns illustrate common real-world routing scenarios.

Recipe 1: Secure eBGP Inbound Transit Filter (Edge Hardening)

Goal: Accept internet routes, discard RFC1918/bogon subnets, drop over-specifics (> /24), and accept default route:

```
# Chain: isp-in | Rule 10: Accept Default Route (0.0.0.0/0)
if (dst == 0.0.0.0/0) { accept; }

# Chain: isp-in | Rule 20: Drop private RFC1918 & martian IP ranges
if (dst in 10.0.0.0/8 || dst in 172.16.0.0/12 || dst in 192.168.0.0/16 || dst in 127.0.0.0/8) { reject; }

# Chain: isp-in | Rule 30: Reject prefixes more specific than /24
if (net.len > 24) { reject; }

# Chain: isp-in | Rule 40: Accept all other legitimate public routes with standard Local-Pref
if (true) { set bgp-local-pref 100; accept; }
```

Recipe 2: Traffic Engineering (Active / Backup Multi-Homing)

Goal: Prefer ISP-A for all traffic. Use ISP-B as backup by boosting ISP-A Local-Pref and prepending on ISP-B:

```
# ISP-A Inbound (Chain: isp-a-in): High Local-Pref 200 (Preferred Exit)
if (true) { set bgp-local-pref 200; accept; }

# ISP-B Inbound (Chain: isp-b-in): Standard Local-Pref 100 (Backup Exit)
if (true) { set bgp-local-pref 100; accept; }

# ISP-B Outbound (Chain: isp-b-out): AS-Prepend 2x (Discourages Inbound Traffic on ISP-B)
if (dst in 192.168.130.0/24 || dst in 192.168.131.0/24) {
    set bgp-path-prepend 2;
    accept;
}
```

Recipe 3: Community Tagging with Passthrough

Goal: Tag customer routes with a BGP Community without interrupting sequential rule processing:

```
# Chain: customer-out | Rule 10: Non-terminal community tag
if (dst in 100.64.0.0/16) {
    set bgp-communities.add 65000:500;
    passthrough;
}

# Chain: customer-out | Rule 20: Terminal permit for specific block
if (dst in 100.64.64.0/24) { accept; }
```

Recipe 4: Remote Triggered Blackhole (RTBH / DDoS Mitigation)

Goal: Drop traffic to an internal IP under attack by signaling upstream provider blackholing:

```
# Chain: isp-out | Rule 5: Signal RTBH /32 to upstream carrier
if (dst == 192.168.131.100/32) {
    set bgp-communities.add 65535:666;  # Well-known Blackhole Community
    set gw 192.0.2.1;                   # Discard Next-Hop
    accept;
}
```

7. Diagnostics, CLI Troubleshooting & FAQ

DebTik provides complete operational visibility into active route policies via both the Web Dashboard and the BIRD 2 command-line interface (birdc).

7.1 BIRD CLI (birdc) Command Reference

CLI Command	Diagnostic Purpose & Usage
`birdc show protocols`	Summarizes status of all dynamic routing protocols (BGP, OSPF, Static), session uptime, and state (Established/Active/Idle).
`birdc show protocols all <peer>`	Primary diagnostic command. Displays detailed filter counters: Import updates, rejected, filtered, accepted and Export updates, rejected, accepted.
`birdc show route export <peer>`	Displays exactly which routes are currently being advertised to the peer after Out-Filter policy evaluation.
`birdc show route import <peer>`	Displays all candidate routes imported from the neighbor after In-Filter processing.
`birdc show route filtered <peer>`	Lists routes that were rejected or filtered out by policy rules.
`birdc show route for <prefix> all`	Inspects detailed route attributes for a specific prefix (AS-Path, Communities, Local Preference, Next-Hop).
`birdc restart <peer>`	Instantly resets the BGP session with the peer for immediate policy re-evaluation and route refresh.

7.2 Frequently Asked Questions (FAQ)

TROUBLESHOOTING GUIDE: Q1: The peer is Established but my static/blackhole networks are not being advertised. Why?
Answer: Verify whether an Out-Filter is attached to the peer. In release 1.3.11+, declared Advertised Networks (Output Networks / Static Blackhole) are exported automatically. Confirm by running birdc show route export <peer> to inspect the active outbound route list.

Q2: I wrote a rule using '==' and BIRD 2 rejected it with a syntax error.
Answer: BIRD 2 strictly requires single = for equality tests. In release 1.3.11+, DebTik automatically sanitizes double equals (==) to single equals (=) upon save, preventing syntax failures regardless of script style.

Q3: How do I know if my In-Filter is dropping routes I need?
Answer: Run birdc show protocols all <peer> and check the Route change stats → filtered counter. If this counter is incrementing, execute birdc show route filtered <peer> to identify the specific dropped prefixes and adjust your filter rules.

Q4: What happened to the 'Select Rules' tab?
Answer: Select Rules was a RouterOS v7 legacy concept with no equivalent in BIRD 2. In release 1.3.11, it was completely removed to ensure platform reliability and eliminate confusion, as all routing policies are cleanly achieved using standard Filter Rules.