CESNET Technical Report 4/2005

Combining Routing and ARP to a Single Lookup Operation

David Antoš

antos (at) fi (dot) muni (dot) cz

July 26, 2005

1 Introduction

In order to process a packet, a router must be able to decide *where* to send the packet and *how* to send the packet through the underlying network. The hardware accelerator COMBO6 [Novotný et al., 2002], developed in scope of the Liberouter project [Novotný, 2002], [Novotný et al., 2003b] uses a lookup engine that supports a single lookup operation to decide how to handle the packet [Antoš et al., 2003], [Antoš and Kořenek, 2004], [Antoš et al., 2003]. To make the behaviour of the accelerator equivalent to the host computer we have to combine routing, link-layer addressing, and packet filtering to the only lookup structure. As the task itself requires to build a bridge between languages of routing tables, ARP caches, and packet filters in the operating system and a completely different language of the lookup processor instructions. As this is obviously too complex to handle at once, we divided the process into two steps. First, we combine routing and L2 addressing (called ARP in this report), creating a structure named RA table. Afterwards, we add the packet filter.

This report is concerned into computation of RA table only. Handling routing table and ARP in a single entity is not a new idea. Some *BSD systems use a single table approach (although they tend to split them [Rizzo, 2004]). To the author's knowledge, the implementations, although well tested in practice, have not been formally shown correct.

We formalise routing and ARP, combine them into a RA table, and show that this table is equivalent to the original sources. Finally, we show properties of the resulting structure that allow to make the computation efficient using concise data structures.

2 Routing

As the principles of routing are equal for both IPv4 and IPv6 protocols, we do not distinguish between them unless necessary. Results of this chapter are applicable for both of them. We must keep in mind that routing tables for the protocols are distinct and they are processed separately. We also use the terminology of IPv4 as it is widely known.

2.1 Routing Table

Routing table is a function of IP addresses returning a record denoting where to send the packet

$$R\colon IP \to Interfaces \times IP$$

where *IP* is the set of IP addresses. The *Interfaces* is a finite set of network interfaces of a particular router. The result of routing is a pair consisting of the output interface and the IP address of the next hop.

The formal definition might suggest that number of routing rules to keep would be enormous. Fortunately, the way of address allocation reduces the number of records to keep significantly. Network interfaces (usually connected to a single physical link or several of them) are connected to a network. The network shares the most significant bits of the address, the common part is called a *network prefix* and its length is a *network mask*. Usual notation is `147.251.54.0/24` which means that the network prefix is 24 bits long. Other common notation uses a *mask*, a number having ones on positions that belong to the prefix, e.g., `147.251.54.0/255.255.255.0`.

2.2 Subnetting and Address Hierarchy

Nowadays, Classless Interdomain Routing (CIDR) is standardised [Fuller et al., 1993]. It allows network masks of an arbitrary length. Networks consist of smaller networks. Network prefixes are divided into sets of internal networks using a procedure called *subnetting*. Their addresses are organised in a hierarchical manner. As a result of this addressing scheme, a single network prefix may address a set of networks. A single network prefix is used to advertise the whole set of networks to the outside world.

Basic principles of sending packets over a subnetted network have been stated in [Mogul and Postel, 1985].[1] To handle a packet, a router checks its destination

Destination	Gateway	Genmask	Flags	Iface
147.251.54.0	0.0.0.0	255.255.255.0	U	eth0
0.0.0.0	147.251.54.1	0.0.0.0	UG	eth0

Figure 1 Example of a routing table

address and finds the *longest matching prefix* in its routing table. An example of routing table is in Figure 1. Packets destined to the `147.251.54.0/24` network are sent through `eth0` interface. The address of the next hop will be directly the final destination of the packet. Any other packet goes to the router that connects this network to the outside Internet, called a *gateway*. Its next hop IP address will be the address of the gateway.

As we can see in the example, the rows of the routing table are essentially of two kinds, local and global. We will characterise them with several equivalent conditions.

Local route is a route leading to the network the router is directly connected to. A packet sent to this network is destined to a host connected to the same subnet as the output interface of the router, sharing the same network prefix with the interface. Its next hop address is equal to its final destination.

Global route is a route to the network that is accessed via an intermediate router, a gateway. The gateway is the next hop of the packet. Global route is a route that is not local.

Routing is essentially finding the longest matching prefix of the destination address. On the other hand, Content Addressable Memories support first match lookups and don't-care bits. We can easily simulate longest matching prefix lookups using CAM. We only order the routes with the length of mask non-increasingly (as the prefixes with the same lengths are disjoint).

2.3 Formal Notation

Although we have described routing tables as longest matching prefix structures, we define it as first match lists. We will show that both representations can be converted. To describe routing tables, we use the following notation.[2]

A routing table R consists of $\text{size}(R)$ rules. Route R_i is the i-th rule for $1 \leq i \leq \text{size}(R)$ and $\text{Len}(R_i)$ is the length of the prefix in that rule.

[1] An interested reader can also learn about the motivations in [Mogul, 1984].
[2] The syntax is strongly inspired by the work of Frantzen [Frantzen, 2003].

Let $[R_i]$ (which is a subset of IP) stand for the packets that i-th rule talks about. If the routing table is represented with a trie structure, $[R_i]$ corresponds to the subtrie addressed by the prefix of R_i. $\mathrm{Len}(R_i)$ is then the length of the path to the prefix in the trie. The prefix is said to have *full length* if it contains a complete IP address (i.e., $\mathrm{Len}(R_i) = 32$ for IPv4 and $\mathrm{Len}(R_i) = 128$ for IPv6).

Route R_i is called *default* if $\mathrm{Len}(R_i) = 0$ (note that $[R_i] = IP$ for the default route). On the "output side," $\mathrm{IP}(R_i)$ is the IP address of the next hop and $\mathrm{Int}(R_i)$ is the output interface of the i-th rule.

The routing table does not have to cover the complete address space. If a destination address is not found in the routing table, the router drops the packet and informs the sender of the packet with "no route to host" error message.

2.4 Properties of Routing Tables

To make further processing easier, we require that the routing table satisfies several conditions. We have two types of conditions. First of them serves to restrict to real routing tables, the other to make further processing easier and/or possible. The requirements go without loss of generality.

1. The rules are sorted in non-increasing prefix lengths, therefore more special rules precede more general ones. Formally, for $1 \leq i < j \leq \mathrm{size}(R)$, condition $\mathrm{Len}(R_i) \geq \mathrm{Len}(R_j)$ holds. Note that this ordering can be easily obtained traversing the trie with the routing table representation in post-order manner.

2. Moreover, prefixes of the same lengths are disjoint, i.e., for all i and j such that $1 \leq i < j \leq \mathrm{size}(R)$ and $\mathrm{Len}(R_i) = \mathrm{Len}(R_j)$ we have $[R_i] \cap [R_j] = \emptyset$. Hence at most one longest matching prefix exists for each destination address.

The ordering allows us to define the semantics. To obtain the result of routing for a destination address $p \in IP$, we find $R(p) = R_i$ where i is the smallest index satisfying $p \in [R_i]$. Due to the ordering, it corresponds to finding the longest matching prefix of the destination address p.

In practice, a routing table is usually kept in a variant of trie data structure. In trie structure, the longest matching prefix has to be found. We will use two representations in this thesis: a longest-match trie and a first-match list that corresponds directly to our definitions. It is necessary to show that both the representations can be converted.

A first-match list equivalent to a given trie structure can be obtained by means of traversing the trie in post-order manner. It sorts the prefixes from the trie in non-increasing way. The routing table contains at most one outcome for a particular prefix, therefore the Properties are satisfied. In the opposite way, we just insert prefixes in the list into a trie; it is again possible due to the Properties.

We will need an extra Property to be satisfied by the routing table. This Property is not related to the routing table representation.

3. Local networks do not contain global networks as their subnets with the exception of full length entries. Formally, for each j such that $1 \leq j \leq \text{size}(R)$ and R_j is local we require that if R_i exists for an i such that $1 \leq i < j$ and $[R_i] \cap [R_j] \neq \emptyset$ then R_i is also local or R_i is a full-length entry. (Note that $[R_i] \subset [R_j]$ in both cases.)

Although this property seems completely artificial, we suggest that it will turn out necessary to allow us to re-arrange the order of records in Section 4.2.

Usage of this requirement would be easier if we omitted the full length entries. It would be sufficient for our purposes. On the other hand, full length entries that violate the condition may be quite common, therefore we tried to find as weak condition as possible.

This requirement does not have to be satisfied by real-world routing tables but routing tables can be converted to comply with it without change of semantics. We will show the method of conversion.

2.5 Prefix Expansion

The easiest way may be to expand the problematic global prefix to a set of full length records. This leads to an increase of routing records exponential to the address length, more precisely to the address length minus the length of the expanded prefix. It makes this method feasible only for very long global prefixes and absolutely unacceptable for IPv6 where the lowest 64 bits are expected to be an interface address.

A practically usable method is to expand the prefix of the local network so as the conflicting non-local entry is no longer in its subnet. Because of the prefix expansion, it is easier to formulate and prove the method using trie representation of the table. Then, the Property 3 can be re-written as follows. We require that if the routing table contains a local prefix a and a prefix b such that a is a prefix of b then b is either local or b has full length.

We describe how to solve a single occurrence of such situation by means of prefix expansion. Then we show how this relationship of prefixes can be efficiently detected.

Let us have binary prefixes $a = x_1 \ldots x_k$ and $b = x_1 \ldots x_k y_1 \ldots y_l$ where $x_i \in \{0, 1\}$ and $y_i \in \{0, 1\}$. The prefixes represent address spaces. Length of prefix a is k, length of b is $k + l$. The task is to expand the prefix a into a set of prefixes that cover the same address space and they are not prefixes of b. The expansion is based on the fact that a can be expressed as $a_0 = x_1 \ldots x_k 0$ and $a_1 = x_1 \ldots x_k 1$ (this relationship is usually called Shannon expansion). We take the one of them that does not interfere with b, put it into the table instead of a and continue

Input: routing table R, prefixes $a = x_1 \dots x_k$ and $b = x_1 \dots x_k y_1 \dots y_l$ (present in R) such that $l \geq 1$.
We construct prefixes

$$\begin{aligned} a_1 &= x_1 \dots x_k \neg y_1 \\ a_2 &= x_1 \dots x_k \;\; y_1 \neg y_2 \\ &\vdots \\ a_l &= x_1 \dots x_k \;\; y_1 \;\; y_2 \dots \;\; y_{l-1} \neg y_l \end{aligned}$$

We construct a set $I = \{i \mid i \in \{1, \dots, l\}$ such that a_i is not in $R\}$. Routing table R' is the copy of routing table R with prefix a removed and prefixes a_i added with the same outcome as a had for all $i \in I$.

Algorithm 1 Expanding a prefix in a routing table

expanding the other. We never overwrite prefixes already present in the table during the process. The formal description is given in Algorithm 1.

We have to show that the result of the Algorithm 1 discards the unwanted relationship of the prefixes. The following Lemma is easily seen from the form of the prefixes generated.

Lemma 1
Prefixes a_i constructed in Algorithm 1 are not prefixes of b.

Proof
Prefixes a_i differ in the final negated bit from b. □

We have to show that the new routing table behaves equivalently with the old one.

Theorem 2
The result of longest matching prefix lookup for an address $p \in IP$ is the same for routing table R and for table R' generated by Algorithm 1.

Proof
Let $p \in IP$. It either is or is not routed by the prefix a in table R.

If p is not routed by the prefix a in R, it either has not prefix $x_1 \dots x_k$ at all and nothing has changed for it in R' or it has been routed by a prefix c such that a is a prefix of c, but c is left untouched in R' as the Algorithm never changes such prefixes.

The remaining case is that p is routed by the prefix $a = x_1 \dots x_k$ in R. Then p itself has prefix a and no other record with prefix a exists in R that would be a prefix of p (otherwise it would be routed by the longer prefix).

We proceed by induction to l:

- Let $l = 1$. If p had a prefix $x_1 \dots x_k y_1$ then it would be routed by b. Hence it must have prefix $x_1 \dots x_k \neg y_1$.

- Let the Theorem hold for $l-1$. Then p is routed either by some of $a_1, \dots, a_{l-1}$ or by $a_l = x_1 \dots x_k y_1 \dots y_{l-1} \neg y_l$. The remaining possibility is that it would be routed by $x_1 \dots x_k y_1 \dots y_l = b$ which is not possible as we supposed p not to be routed by b.

Therefore the result of routing is the same for both R and R' as all the prefixes a_i have the same outcome as prefix a.

□

Violations of the Property 3 are easy to detect traversing the trie containing the routing table in Depth-First Search[3] (DFS). Returning back from a non-local non-full-length node during the DFS, we just check that each node on the path back to the root is not local. If it is not the case, we have to expand the prefix using the method above.

The DFS algorithm never returns to a node it has already visited in the trie [Cormen et al., 2001]. We perform the expansion only *returning* from the node that shall be expanded. The expansion changes only the subtrie of the node. Taking those facts into account, we may continue the DFS after the node is expanded. Hence the only Depth-First Search through the whole trie is needed to find and eliminate all prefixes violating the Property 3.

We have shown that the Properties have no effect on generality of the model. From now on we suppose that routing tables satisfy the Properties unless said otherwise.

3 ARP

The nodes of the network are connected with *links*. In order to reach the destination host, a packet must be moved over each link on the path. A link layer protocol must ensure many services, e.g., framing, medium access, flow control, error detection, retransmission, and addressing.

All the services except addressing are in principle related to packet manipulation (data plane). On the opposite, addresses of communicating partners on a

[3] For a detailed description of DFS properties see [Cormen et al., 2001].

```
Address         HWtype  HWaddress           Flags Mask  Iface
147.251.54.1    ether   00:E0:81:27:DF:7B   C           eth0
147.251.54.10   ether   00:20:ED:5E:6D:98   C           eth0
```

Figure 2 Example of an ARP table

link are maintained by "control plane." An address on the link layer is usually called *physical address*, *LAN address*, in the Ethernet context also *MAC* (Media Access Control) address.

As most link layer protocols (like 802.11 wireless Ethernet, ATM, Frame Relay, etc.) use similar way of addressing, we can use Ethernet as a well-known and illustrative model and as a typical example. We will also use Ethernet terminology, although the principles cover a wide class of level 3 to level 2 address translations.

Ethernet uses 48-bit addresses, usually expressed as six pairs of hexadecimal numbers. The addresses have flat structure (as opposed to a hierarchical structure of IP addresses) and an address is permanent to a particular device[4].

3.1 Address Translation Mechanism

Network layer and link layer addresses have to be used to transfer a packet so translation mechanism between them is needed. The translation principles are described in [Plummer, 1982]. The router maintains an *ARP table* containing IP and MAC address pairs for machines on the local network. The records are kept for a specified amount of time (typically several minutes), so the table is often called an *ARP cache*. To send a packet to a host on the same network (using a local routing record, as we defined in Section 2), the router consults its ARP table. If the appropriate record is found, it uses the address to send the frame. If the ARP cache does not contain the record, it sends ARP query packet by means of Ethernet broadcast (`FF:FF:FF:FF:FF:FF`). The ARP query packet contains the requested IP address and each host on the local network checks if its IP address matches the requested one. The host with the matching address sends back an ARP response with the MAC address. The querying node updates its ARP cache and sends the IP packet.

We have discussed sending a datagram to a node on the same local network. Let us now check the situation when the packet should be sent to a node off the local network, using a route we called global, through a gateway. In that case, the host must fill in the MAC address of the gateway the packet should go through. If the address is not in the ARP cache the procedure we described above is used to query it. The ARP cache does not distinguish gateways and other nodes, moreover

[4] At least in theory. Physical addresses used to be written in ROMs. Today's adapters keep their addresses in a flash-type memory, allowing experienced users to change them.

a gateway may be accessed as a node on a local network via the same address. An example of ARP table is in Figure 2.

3.2 Formal Notation

Formally, ARP table is a function

$$A\colon IP \to MAC$$

where MAC is a finite set of physical addresses.

In order to keep the notation consistent, we denote ARP records with A_i for $1 \leq i \leq \text{size}(A)$. An ARP record A_i translates an IP address from $[A_i]$ to the hardware address. The ordering of rules is not relevant here as the set $[A_i]$ contains just a single IP address, not a portion of the address space denoted by a prefix. We require the records to be unique, i.e., for $1 \leq i < j \leq \text{size}(A)$ the condition $[A_i] \cap [A_j] = \emptyset$ holds. The result of the ARP lookup is the MAC address $\text{MAC}(A_i)$.

The result of ARP lookup for a destination address $p \in IP$ is the record $A(p) = A_i$ such that $p \in [A_i]$.

The ARP table does not have to cover the complete IP address space. If the corresponding MAC address is not found for the next hop, the technique we described in Section 3.1 is used to learn it. If it does not succeed, the packet is dropped and an error message is returned. For constructing tables for hardware lookups, only "current snapshot" of the table is necessary, as we explain in Section 4.2. The mechanism of learning ARP records is purely matter of the operating system of the router.

4 Routing and ARP Table Combination

We have described routing, ARP, and packet filtering, and how they cooperate in the operating system. In order to perform similar lookup operation in hardware lookup machine, combining them to the only lookup is needed. We start the description with the problem of combination of routing and address resolution to show the basic concepts first.

4.1 Software Cooperation

An essential principle is used in the design: If the hardware lookup cannot resolve a packet itself it can send it to software in the same way as an ordinary network adapter would.

In the algorithms, this will be denoted by a '*SW*' action that means the packet is sent to software router to be processed. The importance of having *SW* action is that *SW* acts as an "oracle" that always performs the correct action, just in the sense of hardware/software co-design ideas.

Of course, packets passing through software are processed slower than packets switched by the hardware engine. It does not have to indicate a serious problem if the portion of the software processed traffic related to the total traffic is small enough. On one extreme, we can achieve correct behaviour of the router sending all traffic to software processing. On the other hand, maximal performance would be possible only if all traffic was switched by hardware.

Nevertheless, two kinds of reasons can prevent us to reach this goal.

- We can have "design reasons" (dictated by the economy of the development) not to process some types of packets in hardware. E.g., IPv6 Routing Option requests swapping two IPv6 addresses in the output editor. This operation is not suitable to perform in FPGAs as 128-bit string swapping would consume too large area of the chip or inadequate time, depending on the design. On the opposite, packets with Routing Option are very rare in the traffic.

  This category also contains handling erroneous packets. The easiest way to create an appropriate response (an ICMP message) is to send the packets to the operating system.

- Resigning from hardware processing can be helpful in case of "runtime problems" as the last resort. In the process of lookup structure compilation, running out of available memory may occur or length of lookup branch may exceed an allocation block.

4.2 Routing and ARP

Let us abstract away from packet filtering for a while. The aim of this section is to combine routing and ARP into the only lookup operation. Although the combination of routing and ARP tables is in principle easy we will study it in detail. On the other hand, the algorithm—apparently straightforward and obvious—turns out to have its dark corners.

Combining routing and ARP tables (and even maintaining them together) is not a new idea. Open-source BSD clones (FreeBSD, NetBSD) use a single table to store both routing and ARP information. Nowadays, FreeBSD designers tend to split the tables, saying that handling the tables separately is easier and avoids unnecessary redundancies [Rizzo, 2004].

Taking into account routing only, the result of the hardware lookup operation is (a reference to) an editing program and the next hop MAC address as its parameter. While a software router usually searches for the longest matching

routing entry in order to obtain a next hop and then it resolves its MAC address, the hardware engine has to solve both steps at once.

4.3 Lazy ARP Record Removal

The operating system maintains a routing table and an ARP cache. Records in the ARP cache are kept for a specified amount of time, typically from five to twenty minutes. In order to minimise the recomputations of the lookup structure, the initial idea was to test the whole network and to build an ARP cache as complete as possible[5]. Although this would be feasible with small networks, it is not scalable very well and it is unclean how to handle validity of records in such table. Moreover, the highest principle should be to keep the behaviour of the software router untouched whenever possible, so this approach was definitely rejected.

Instead of pro-active building of the ARP cache, we use current content of the table, marking currently unresolved entries to be processed by software. The lookup structure must be updated when an ARP cache entry is added or modified. A MAC address of the next hop of an incoming packet may be resolved, in that case the packet is forwarded by hardware. Otherwise the packet is sent to software. Software emits the ARP query to learn the MAC address of the next hop and enriches the ARP cache with the response. As the content of the cache changed, the lookup structure must be recomputed. Finally, when the lookup structure propagates into hardware, incoming packets forwarded to the next hop added will be forwarded by hardware.

A delicate problem arises with removing entries from the ARP cache. To keep the principle of equivalence of hardware and software behaviours, we should recompute the lookup structure when an entry from the ARP cache is deleted. Nevertheless, we have two reasons against it. From implementation point of view it is not easy to arrange kernels to announce ARP entry deletion. Moreover, it is desirable to keep the update frequency of the hardware lookup structure as small as possible.

Let us study "observable" effects of entry removal in order to show that we may be reluctant removing the entries. The main reason for purging the cache is that a network interface may change its hardware address (e.g., when the network adapter is replaced). Removing entries ensures that the old address is switched to the new one in several minutes. Another reason is keeping the cache reasonably small, as pointed out by Malkin [Malkin, 1995] (the original RFC [Plummer, 1982] only addresses the problem stating this issue needs more thought). To sum up, we may omit updating the lookup structure in case of ARP cache entry deletion in case that updates stimulated by other causes are reasonably frequent (we can

[5] Theoretically, this can be achieved easily trying to `ping` all machines in the local network. However, it is not clean how to decide that an address in the network is unreachable.

use a timer restarted by each update). Although we slightly violate the "hardware and software equivalence principle," lazy removing of the entries just lengthens the timeout.

4.4 RA Table Definition

We have current contents of routing and ARP tables as the input. The task is to compute the table that combines routing and ARP into the only lookup operation. We call it a routing-ARP table:

$$RA\colon IP \to (Interfaces \times MAC) \cup \{SW\}$$

The RA table returns either the interface and MAC address of the next hop where the packet should be sent or it indicates that the packet must be processed by software.

We will use RA_i for routing-ARP rules (for $1 \leq i \leq \text{size}(RA)$) and $[RA_i]$ for IP addresses affected by the rule. Analogically with routing syntax, $\text{Int}(RA_i)$ and $\text{MAC}(RA_i)$ are the interface to reach the next hop and the hardware address of the next hop, respectively.

Let $\text{Len}(RA_i)$ be the length of the prefix in the rule; RA records with length zero are called *default RA rules* and records with the length of the IP address (32 for IPv4 and 128 for IPv6) are called *full-length RA records*.

To obtain a result of a destination address $p \in IP$, denoted $RA(p)$, we find RA_i for the smallest i such that $p \in [RA_i]$.

4.5 Computing First-Match RA Tables

Creating the RA table out of routing and ARP tables is more-or-less straightforward. Traversing the routing table we "inject" ARP entries inside. In case of locally connected network, we add all resolved ARP entries and finally we send the rest of the network to the operating system (in order to resolve ARP and/or emit an error). For global routes we insert the MAC address of the next hop into the table directly instead of its IP address if the MAC address is known. Otherwise the route must be processed by software.

Algorithm 2 is a pseudocode for combining routing table represented as a list (which is equivalent to traversing a trie in post-order manner) and ARP table (represented in an arbitrary manner, say a list) into the RA table represented as a list.

We have to show that the RA table computed by this algorithm behaves the same way as routing and ARP tables. It means that the RA lookup result is the same as ordinary software processing where lookups in routing table and resolving

```
1   RA = ∅
2   l = 1
3   for i = 1 to size(R) do
4       if R_i is local
5           then
6               /* go through all ARP records in this local subnet: */
7               for each j such that 1 ≤ j ≤ size(A) and [A_j] ⊆ [R_i] do
8                   [RA_l] = [A_j]
9                   RA_l = (Int(R_i), MAC(A_j))
10                  l = l + 1
11              done
12              /* the rest must get resolved in software */
13              [RA_l] = [R_i]
14              RA_l = SW
15              l = l + 1
16          else /* R_i is global */
17              if exists k such that IP(R_i) = [A_k]
18                  then
19                      [RA_l] = [R_i]
20                      RA_l = (Int(R_i), MAC(A_k))
21                      l = l + 1
22                  else
23                      [RA_l] = [R_i]
24                      RA_l = SW
25                      l = l + 1
26              fi
27      fi
28  done
29  if [RA_{l-1}] ≠ IP /* we have no default route */
30      [RA_l] = IP /* add the final default route to software */
31      RA_l = SW
32  fi
```

Algorithm 2 Combining routing and ARP into RA table

ARP are performed separately. Sending the packet to software is also a correct result—in that case the packet is processed by the original routing and ARP tables.

Lemma 3
RA table produced by Algorithm 2 is total (i.e., it is defined for each destination address $p \in IP$).

Proof
To ensure that the RA table is total, it is enough that it contains a default RA record.

If the routing table contains the default route then the default route must be the last record in the table and the default RA record is inserted into the RA table. At the end, the algorithm tests presence of the default route in the table. If the default route was not present the algorithm adds final the RA record destined in software. □

We have shown that the result of RA lookup is defined. In the following theorem, we prove that the result is correct.

Theorem 4
For each destination address $p \in IP$, routing-ARP table contains a result that is either SW or it is the same as applying routing and then ARP on the destination address, i.e., $\mathrm{Int}(RA(p)) = \mathrm{Int}(R(p))$ and $\mathrm{MAC}(RA(p)) = \mathrm{MAC}(A(\mathrm{IP}(R(p))))$.

Proof
Let us have a destination address $p \in IP$. As the RA table is total, the smallest index l exists such that $p \in [RA_l]$.

If $RA_l = SW$ then the destination address is handled by the original routing table and ARP cache in software, thus correctly.

Let us suppose the result is an interface $\mathrm{Int}(RA_l)$ together with a MAC address $\mathrm{MAC}(RA_l)$. In that case, the RA rule RA_l has been created either in lines 8–10 or lines 19–21 of the algorithm. We will study both cases.

Let us suppose that the rule RA_l was created in lines 8–10. Such an entry has a single prefix of full length in $[RA_l]$ (it originates from the ARP table that contains full addresses). We will show that the RA_l rule was created from a route R_i where i is the smallest index such that $p \in [R_i]$. Suppose the opposite. Then an index j such that $1 \leq j < i$ must exist such that $p \in [R_j]$. Length of prefix of R_i cannot be equal to $\mathrm{Len}(R_j)$ as prefixes of equal lengths are disjoint. Therefore $\mathrm{Len}(R_j) > \mathrm{Len}(R_i)$ due to the ordering of R. As each routing record produces at least one routing-ARP record with the same address space (the final software entry for local routes and/or the global route), the algorithm must have produced an entry RA_k such that $1 \leq k < l$ and $p \in [RA_k]$. This is not possible as we chose the RA_l as the first matching entry for destination address p. Hence $\mathrm{Int}(RA(p)) = \mathrm{Int}(R(p))$.

As $\mathrm{IP}(R(p)) = p$ for local routes, a unique ARP entry A_j exists such that $\mathrm{MAC}(RA_l) = \mathrm{MAC}(A_j)$.

The remaining case is that the rule RA_l was created in lines 19–21. We will show that the rule originates from the first R_i for that $p \in [R_i]$. Suppose not, precisely, suppose that a routing record such that $p \in [R_j]$ exists for $1 \leq j < i$. Then again, $\mathrm{Len}(R_j) > \mathrm{Len}(R_i)$, moreover each routing record produces at least one RA entry, therefore an RA record RA_k must exist such that $1 \leq k < l$

and $p \in [RA_k]$. That is not possible due to the choice of RA_l as the first matching record.

To sum up, $\mathrm{Int}(RA_l) = \mathrm{Int}(R_i)$ and $\mathrm{MAC}(RA_l) = \mathrm{MAC}(A(\mathrm{IP}(R_i)))$. □

We have obtained a routing-ARP table that is behaviourally equivalent to the original routing and ARP tables. The RA table is first match. It can be advantageous to have a trie representation of the table (i.e., an equivalent longest matching prefix representation) in order to obtain a concise representation.

If we were able to sort RA entries in non-increasing way and ensure that prefixes of equal lengths are disjoint we would have a table that can be converted into a trie form. To reach this goal, let us study the ordering and relations of rules produced by the algorithm and potential redundancies in the table that have to be discarded.

Lemma 5
Algorithm 2 sorts the resulting RA table in non-increasing prefix lengths with exception of prefixes of full lengths, i.e., for $1 \leq k < l \leq \mathrm{size}(RA)$ the following condition holds: $\mathrm{Len}(RA_k) \geq \mathrm{Len}(RA_l)$ or RA_l has full length.

Moreover, prefixes with equal lengths with exception of full-length prefixes are disjoint.

Proof
The RA records are copied in the same order as the original routes with the exception of resolved local ARPs that can produce full addresses.

Each routing record produces just one RA prefix of the same length and optionally a number of full-length prefixes. Note that the final default RA record destined in software is added only if no default record was present in the table. □

As we see from the preceding lemma, the only issue that remains to be solved is the structure of full-length prefixes. The full-length prefix for a particular destination address may be repeated across the table several times. In that case the first occurrence is of interest as all the following ones are unreachable and may be omitted.

Lemma 6
Let us have $1 \leq k < l \leq \mathrm{size}(RA)$ such that $[RA_k] = [RA_l]$. Let RA' be the table RA with row l omitted ($RA'_i = RA_i$ for $1 \leq i < l$ and $RA'_i = RA_{i+1}$ for $l \leq i < \mathrm{size}(RA)$). Then $RA(p) = RA'(p)$ for each destination address $p \in IP$.

Proof
Obviously, the row RA_l is unreachable in the original table. □

Full-length prefixes are mixed into prefixes of other lengths. In order to prepare the table to be converted in longest prefix semantics it would be more suitable to move the prefixes to the beginning of the table. We have to make sure that this operation does not change the meaning of the table. To ensure this, it is enough that the address space affected by the prefix is disjoint with all the preceding rules (i.e., rules of an arbitrary length). Blocks of disjoint records can be freely re-ordered.

Thanks to the preceding Lemma, we have to be interested in the *first* occurrence of the full-length prefix for an address only.

The meaning of the following result is that the first occurrence of a full length entry in the RA table is disjoint with *all* its predecessors. It can also be understood from an operational point of view that the first occurrence of a full length entry for an address is *reachable* in the routing-ARP table.

Lemma 7
Let RA_l be the first occurrence of full-length RA record for a given destination address, precisely, let RA_l be a full-length RA record satisfying that for each full-length record RA_m such that $[RA_l] = [RA_m]$ we have $l \leq m$. Then for each k such that $1 \leq k < l$ condition $[RA_k] \cap [RA_l] = \emptyset$ holds.

Proof
Let $p \in [RA_l]$ satisfying the prerequisites of the Lemma. We want to show that no index k exists such that $1 \leq k < l$ and $p \in [RA_k]$. Let us suppose the opposite, i.e. that a routing-ARP entry RA_k exists such that $p \in [RA_k]$ and $1 \leq k < l$ (saying nothing about its length). We will study all possibilities how such an entry could have been created.

Let R_j be the routing record from which RA_l was created and R_i the record that produced RA_k. First, we note that $p \in [R_j]$ and $p \in [R_i]$. If it was not so the routing-ARP entries would not contain p (the algorithm copies the address space and/or takes a full-length entry out of the address space of a local route). Moreover, $i \leq j$, otherwise the order of the produced routing-ARP entries would not be preserved. (Note that equality is possible, a routing rule can produce several routing-ARP entries.)

The route R_j can be either global or local. We will study both cases.

- Let R_j be global[6] first. Then R_j must have full length, otherwise it would not produce a full length entry RA_l. As $i \leq j$ the route R_i must have full length due to the ordering of the routing table. If $i < j$ then $[R_i]$ and $[R_j]$ would be disjoint. This is not possible as both contain p. Hence $i = j$. As global route

[6] This is a theoretical option as it makes no sense to have subnets of full lengths normally. Anyway, e.g. Linux `route` command allows to configure them, therefore we have to allow such pathological cases in the model.

produces just one routing-ARP entry, we cannot have $k < l$. Therefore R_j cannot be global.

- Let R_j be local. In that case the route R_i must be local or full-length due to the requirement 3 on page 5.

  - Let R_i be local. As R_j is local, the record RA_l must have been created either in lines 8–10 or lines 13–15 of the Algorithm 2.

    - ⋆ If RA_l was created in lines 8–10 then an ARP record for p must exist. It would be used also to expand the route R_i, hence RA_l would not be the first full-length record for p.

    - ⋆ If RA_l was created in lines 13–15 then R_j would have full length. As we have $i \leq j$ then route R_i would have full length, too.

      We show that $i = j$. If $i < j$ then $[R_i]$ and $[R_j]$ would have to be disjoint. This is not possible as both contain p. Hence $i = j$.

      It is possible that the route R_i created two (full-length) entries for p. In that case RA_l would not be the first full-length routing-ARP entry containing p.

  - The remaining case is that R_i has full length. Then R_i must have created a full-length record RA_k such that $p \in [RA_k]$. Then RA_l is not the first full-length record handling p, which is not possible.

  The record RA_k could not be created in any of the cases. □

To prove the preceding lemma, we needed the requirement that all subnets of local networks are local or have full length. If it was not so, an unreachable full length entry might be produced by the algorithm. Consider a fragment of a routing table as an example:

```
1.2.3.0/24 -> global route
1.2.0.0/16 -> local route
```

Suppose that an ARP record exists for the `1.2.3.4` destination address and the global route has just been added. The ARP record will have been occupying the ARP cache for several minutes. The algorithm would produce

```
1.2.3.0/24 -> some gateway and/or SW
1.2.3.4/32 -> resolved ARP in the local network
1.2.0.0/16 -> some gateway and/or SW
```

This algorithm is the same as Algorithm 2 with the following modifications.

- The resulting RA table is kept in a trie structure. (Hence it is sorted on-the-fly in non-increasing way and all records are unique.)
- Let us understand assignments that create rules RA_l in the following way:
  - Assignment to $[RA_l]$ is creating a path in the trie denoting the prefix of the RA_l rule.
  - Assigning the output to the rule (e.g. $RA_l = (\mathrm{Int}(R_i), \mathrm{MAC}(A_j))$) means *assign the output only if it was empty previously.*

Algorithm 3 Combining routing and ARP into RA table expressed as trie

The second record is unreachable and cannot be moved to the beginning of the table.

4.6 RA Table Properties Summary

To sum previous results up, we have observed following properties of the routing-ARP table produced by the Algorithm 2:

1. Only the first occurrence of an address space is of interest, precisely if the algorithm creates an entry RA_l and an entry RA_k such that $[RA_k] = [RA_l]$ and $k < l$ has been created before, we do not have to store RA_l into the RA table according to Lemma 6.

2. Full-length entries may be pushed to the beginning of the table thanks to Lemma 7. This way we obtain a table sorted in non-increasing prefix lengths.

4.7 Longest Prefix Representation of RA Table

Instead of post-processing the table, we may change the order of the rules immediately during the run of the algorithm with a small change of semantics of assignments. The final version of the computation is shown in Algorithm 3.

With those changes, the algorithm constructs a trie representation of the RA table which is equivalent to the original output list. Only the first rule for a given address prefix is recorded and full-length entries may be harmlessly "moved to the beginning" as we have shown above. Again, first match representation may

be obtained from the trie traversing it in post-order manner. Also note that rules sharing common prefix length are disjoint.

The RA table computed out of examples shown in Figure 1 and Figure 2 is shown in Figure 3.

```
147.251.54.1/32 -> eth0, 00:E0:81:27:DF:7B
147.251.54.10/32 -> eth0, 00:20:ED:5E:6D:98
147.251.54.0/24 -> SW
0.0.0.0/0 -> eth0, 00:E0:81:27:DF:7B
```

Figure 3 RA table computed out of R in Fig. 1 and A in Fig. 2

5 Conclusion

We have formalised a method to combine routing and ARP into a single equivalent operation. Moreover, the RA structure can be expressed as easily as the routing table itself, by a trie.

The remaining (and more complex) step is to add the packet filter and to convert the resulting structure into the lookup structure of the COMBO6's processor.

Acknowledgement I am very thankful to Vojtěch Řehák for bringing me to the idea how to formalise routing and ARP and to Petr Holub for willingly listening my proof concepts.

Bibliography

Antoš, D., Kořenek, J., Minaříková, K. and Řehák, V. (2003). Packet header matching in Combo6 IPv6 router. Technical Report, CESNET.

Antoš, D. and Kořenek, J. (2004). String Matching for IPv6 Routers. In Boas, P. V. E., Pokorný, J., Bielikova, M. and Štuller, J., editors, *SOFSEM 2004*, pages 205–210, Měřín, Czech Republic. MATFYZPRESS, Prague. ISBN 80-86732-19-3.

Antoš, D., Kořenek, J. and Řehák, V. (2003). Vyhledávání v IPv6 směrovači implementovaném v hradlovém poli. In *EurOpen, Sborník příspěvku XXIII. konference*. EurOpen. ISBN 80-86583-04-X.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2001). *Introduction to Algorithms*. MIT Press, Second edition. ISBN 0-262-03293-7.

Frantzen, L. (2003). Approaches for Analysing and Comparing Packet Filtering in Firewalls. Master's thesis, Technical University of Berlin.

Fuller, V., Li, T., Yu, J. and Varadhan, K. (1993). RFC 1519: Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy. Obsoletes RFC1338.

Malkin, G. (1995). RFC 1868: ARP Extension—UNARP. Status: experimental.

Mogul, J. C. (1984). RFC 917: Internet subnets.

Mogul, J. C. and Postel, J. (1985). RFC 950: Internet Standard Subnetting Procedure. Updates RFC0792.

Novotný, J. (2002). Projekt routeru IPv6. *Zpravodaj ÚVT MU*, 1/2002:10–12. In Czech.

Novotný, J., Fučík, O. and Antoš, D. (2003b). Project of IPv6 Router with FPGA Hardware Accelerator. In Cheung, P. Y., Constantinides, G. A. and de Sousa, J. T., editors, *Field-Programmable Logic and Applications, 13th International Conference FPL 2003*, pages 964–967. Springer Verlag.

Novotný, J., Fučík, O. and Kokotek, R. (2002). Schematics and PCB of COMBO6 card. Technical Report, CESNET.

Plummer, D. C. (1982). RFC 826: Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware.

Rizzo, L. (2004). New arp code snapshot for review.