

Advanced Computer Networks
Professor Doctor Neminath Hubballi
Department of Computer Science Engineering
Indian Institute of Technology, Indore

Lecture 8

IP Table Lookup- Optimized Trie based Data Structures - Part 2

(Refer Slide Time: 0:19)



So, with that background, let us summarize what did we discuss? We said that we want to do the lookup operation; the lookup operation needs to be fast enough because the routers are getting millions of packets per second, particularly if the router is in the backbone of the network, and in order to do that, we have got a bunch of options, one option was to implement that in the software and that software implementation turned out to be a trie structure, we started the binary and then binary trie structure and then couple few optimizations that you can do in order to minimize the height of the trie structure.

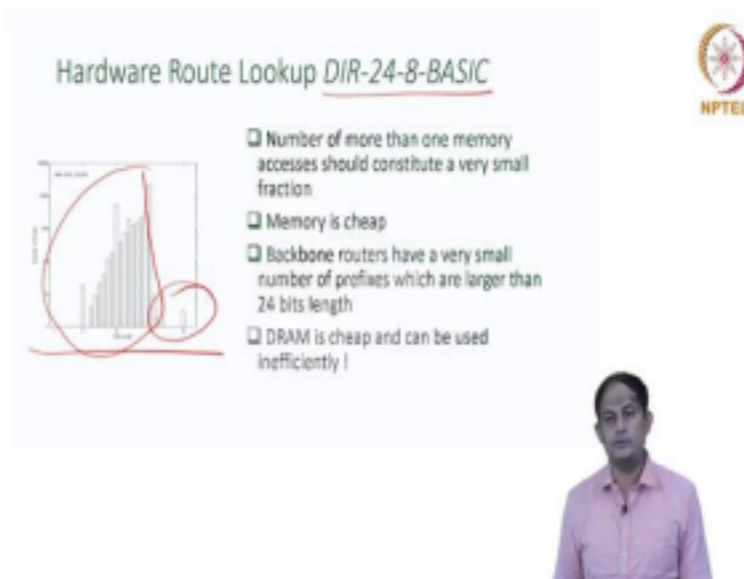
But in spite of that software implementation, whatever optimization that I do are still slower in operation, and in the worst case, if the entire trie structure is completely full, then you cannot bring any optimization, particularly, in the binary trie level. So, you can do the optimization that is, if all 2^{32} entries are there in the routing table, then if you use the binary structure that will be full, you will have the worst-case performance of 2^{32} number of entries, and that is the memory space and also the 32 number of the memory references are required.

Because you have to traverse from one pointer to another pointer, that is going to be quite complex or tedious in nature and that probably will not scale particularly if you go to the IP version 6, then the IP address length is 128 bits in the worst case 2^{128} is the memory size requirement and then 128 memory pointer references are required in the worst case. So, that is going to be quite slow, and it is not going to scale up.

And now the question that we ask is what is the other thing that you can do if software implementations are not scalable then we turned into hardware? So, the question that we ask is, what kind of hardware lookup can I do to do the faster route lookup? So, some of the example that we see at least a couple of examples that I am going to discuss in this class, and the goal is to do the lookup operation in order of one time .

So, one memory access should be good enough to do the forwarding operation, so that is the overall goal. So, let us see whether that is possible or not using the hardware. So, what kind of hardware and memory access mechanism can we do for doing the lookup operation?

(Refer Slide Time: 3:03)



The slide is titled "Hardware Route Lookup DIR-24-8-BASIC". It features a histogram on the left showing the distribution of prefix lengths, with a red circle highlighting the peak at 24 bits. To the right of the histogram is a list of four points:

- ❑ Number of more than one memory accesses should constitute a very small fraction
- ❑ Memory is cheap
- ❑ Backbone routers have a very small number of prefixes which are larger than 24 bits length
- ❑ DRAM is cheap and can be used inefficiently !

In the top right corner, there is a logo for NPTEL (National Programme on Technology Enhanced Learning).

So, first, I will discuss a simpler architecture of the hardware-based route lookup for this lookup which is called the direct 24-8-basic route lookup architecture. And this is a research effort at the Stanford University and it is a very simple lookup operation that is done in the hardware. And what it does is it divides the prefixes into two parts, one is the 24-bit up to length 24, 0 to 23, and from 24 to 32 that is the complete address. It has got 2 tables the first table holds the

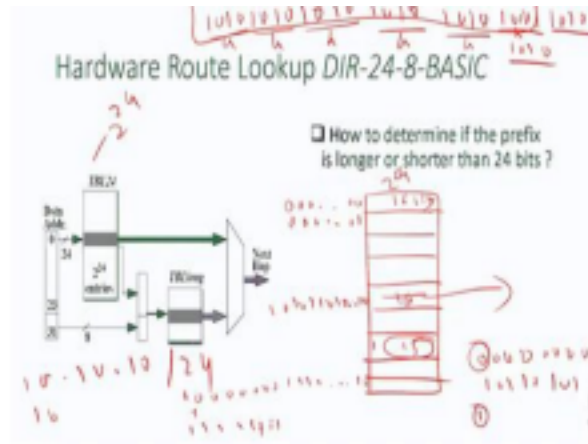
prefixes up to length 24 and the second level table holds the prefixes which are longer than the 24 bits including the 24 bits 0 to 23 and 24 onwards it is in the second table.

So, why this kind of structure, this is a very simple architecture, what they did is they took some of the practical router forwarding tables and looked at the distribution of the forwarding table, and what kind of prefixes are there in those forwarding tables. So, it turned out that the majority of the prefixes are concentrated towards or the maximum number of the prefixes are up to length 24 or less than that and beyond 24 there are only a few numbers of the prefixes and it makes sense to have concentrated all of these prefixes up to length 24 in one table and the remaining ones into another table without wasting the memory size that is the motivation for this kind of the architecture.

So, the idea is having number of more than one memory reference, so if I have multiple tables then multiple memory references are required. So, you remember in the multi-bit trie one table 3-bit second level 3 bits and so forth it was going, so more number of this structure I got more number of the memory references are required, so that is going to be slower the operation. So, here in this case I am going to do one level operation up to 24 bits, 0 to 23 and then 24 onwards to the second, only two tables are there and I am going to store the entries in such a way that one lookup operation or one memory reference in the first level is sufficient and one memory left references in the second level is sufficient.

And why this is the motivation for having this kind of structure is the they came up with argument that memory is getting cheaper when I say the memory this is an inexpensive DRAM memory that is available and DRAM is cheap and can be used inefficiently, why inefficiently, because we are going to expand if any prefixes which is less than 24 bit lengths that way I am going to expand and then fill it in the first level table.

(Refer Slide Time: 6:15)



So, in effect, what I am trying to say is that the first-level table will have 2^{24} number of the entries. And any prefix which is less than 24 bits, one lookup in the first-level table is sufficient, any prefix which is longer than that, then the second-level reference is required. In the worst case, two memory lookups, one in the first level and the other in the second level, would be sufficient to find out where to forward the packet or if the lookup operation is complete.

So, what it does? It does the following, the first level table would look something like this, the table itself has these slots and in the first level, all 24 0 bits are the first entry, the first 23 0 and then 1 is the next entry, and all 24 bits 1 is the last entry that is there in the table, this is how the 2^{24} bits the table is taken. And the width of each entry is 16 bits.

So, when I reference, let us say the destination IP address is something like this 10101010101010101010101010101010. So, what I am going to do is take the first 24 bits, up to bits from starting to this one, and then I go to that memory location, meaning whatever the memory location, which is all 101010 that memory location I go let us say this is the location that I get through for that one.

And this value is whatever the entry that I get up referring to this is the 16 bits and I take that and depending upon whether this is actually indicating or representing a prefix which is lesser than 24 bits or greater than or equal to 24 bits, I decide whether the second level memory reference is required or not.

So, how do I determine whether the second-level references are required or not? by looking at

the entry. If let us say the 16 bits are something like this, I get 00000000000000000000000010110101. So, if the first bit out of the 16 bits is 0, it represents prefix which is less than or equal to less than 24 bits, and I do not require the second level reference and then I go whatever is it is indicating, on which port number I need to forward this packet, I am going to just forward that.

If instead of 0, the first bit is actually 1, then it requires that this is the prefix which is longer than prefix is longer than or equal to 24 bits, second level reference is required. I go to the second level and then decide. So, depending upon whether the first bit out of 16 bits is 0 or 1, I am going to decide whether the second level lookup reference is required or not.

Now, what does the second-level table contain? Second level table contains 256 entries for every possible prefix that is there in the first level entry, if the prefix is longer than or equal to 24 bits, what I am trying to say is, if let us say the entry is 10.10.10/24 and now the first 24 bits or indexed in the first table and let us say this is entry 1 followed by something, and these 15 bits indicate where exactly the 256 entries correspond to the remaining possibilities in the second level table, what is that 10.10.10/24, so, these are indicating 24 bits, and the remaining eight bits can be anything from all 00000000 to all 11111111.

So, for all these possible 256 cases, 8 bits are there, the remaining 8 bits I do not care, if I have a plus /24 prefix then I do not care about the remaining 8 bits. So, for one entry in the table 1, if the prefix is of the length larger than or equal to 24 bits, you will have 256 expanded a number of the entries in the secondary level table and the index of the starting index of that entry is stored in the first table, so that is given by these 15 bits. So, I take these 15 bits and then multiply it by 256 I get the index number, I go to that index number and starting from that is the remaining entries that I am going to get.

(Refer Slide Time: 12:09)

Hardware Route Lookup DIR-24-8-BASIC

☐ Prefixes shorter than 24 bits are expanded

Handwritten notes on the slide:

- 24
- 10-10/16
- 10-10.0
- 10/8
- 256
- 10-0.0
- 256
- 256



So, let us take a look in an example that should be clear. So, any prefix which is smaller than 24 bits is expanded up to 24 bits, let us say I have got an entry which is 10.10/16 this is not 24 bits. For 256 possibilities what are those? 10.10. it can be 0 it can be 1 so on it can be 256.

So, for all these 256 entries, one entry which is having /16 is expanded to 256 entries and then stored. If you have something 10/8 then 256 possibilities for the second position and 256 possibilities for the third position that is, the 256 x 256 times the prefix is expanded and then stored in the first level table, so that is what is done.

So, the first level table is always full, so all 2^{24} entries are always there in the first level table. So, if any prefix is less than that you need to bring it into the 24-bit format so in this the example that I showed. So, similarly we have 10/8 then for these possible cases 256 cases for this one and 256 cases for this one, you expand it and then store it as one single entry in the table.

(Refer Slide Time: 13:47)

Hardware Route Lookup DIR-24-8-BASIC

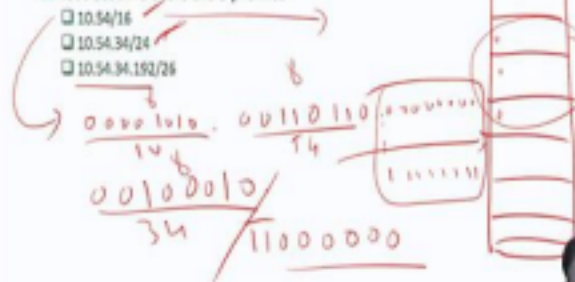
- Each route prefix which has at least one entry larger than 24 bits is allocated 256 slots in the Table Long



Hardware Route Lookup DIR-24-8-BASIC

- Let's assume there are 3 prefixes

- 10.54/16
- 10.54.34/24
- 10.54.34.192/26



So, let us take an example to see how this will look like, let us assume these are three prefixes that I have got one prefix is of the /16, 16 bits prefix 10.54 and 10.54.34/24 bit prefix and then the third one I have got is 26 bit prefix, because this is the CIDR entry.

Now, what are all kind of entries that I got for these many cases in the table1 and table 2. So, 10.54 is the prefix, first is I convert this into binary. I got a table, the first level table entry and I convert this 10.54 into binary, so what is that look like, the first one 00001010 the 10 is over and 54 when you convert it to binary you get to 00110110.

And what are the other third bit? Because I need to bring it into these cases, so for all entries starting with all 0s to all 1s. So, I am going to have these many 256 entries are going to be there in the first level table 24. So, I am going to make these many number of the entries for this prefix in table number 1 and for all of this, the first bit is going to be 00 and the remaining bits indicate the indexing to the forwarding table entry on which port number they need to be forwarded, next hop information.

Similarly, I have got one entry which is 10.54.34, so 10.54 is common, and then out of that 34 is one possibility among these many combinations from all 0s to all 1s, so 34, if you convert it into binary, you will get to 00100010 and one among these entries that I have created is going to be reserved for this 10.54.34 and that is the case.

And then in the second level secondary level table, I am going to expand into 256 entries because this is the prefix of length 24 and a subset of that which is the series 10.54.34.192/26, if you convert this number 10.54.34.192. So, this is the my 10, this is my 54, this is my 34 and then the last one 192 when you convert it into binary, you get to 11000000.

And so, what it means is: 8 bits here, 8 bits here, 8 bits here and these 2 bits are fixed and the remaining 6 bits I do not care. So, when I examined, I already made entries for these cases /34 is already there, 256 entries are created in the first level table and some portion of the remaining 256 entries in level number 2 are now reserved for the third portion, so that is how the table looks like.

(Refer Slide Time: 17:37)



And 10.54.34, the third column is actually for the cases where the direct forwarding can be done by consulting this table only the third column actually has the port number information, here the port A is one port for 10.54/16 series, port B is for 10.54.34/24 series and port C is for 10.54.34.192 series.

So, what this third column for this entry indicates is, it is an index to the starting position of the second table. So, the entries corresponding to the expansion of this entry into the second table is starting at the index number 120, that is what it means. So, the 120 is multiplied by 256 because every expansion from the first table to the second table will have 256 entries, so what it means is 119 entries starting from the address are already there in this table.

So, those are here 256×119 entries this is the starting position of the expansion corresponding to 10.54.34. So, this index is that number and starting at this one, so this indicates that it should forward this with the port number B. So, all these entries will have port number B marked, so 120 multiplied by 256 and the next entry 256 plus 1 plus 2 and so forth.

So, up to how long you need to do that, so the third series is actually a subset of this series 10.54.34.192, up to the point it is 191 you forward it to port number B, the moment you see 192 the entry you need to forward it to port number C. So, how long till the all the remaining entries what it means is 192 if you convert it into binary, so it will be something like the 11000000 starting with I do not care what 192/26 the 24 bits are actually covered in the previous case and the 2 bits are kept constant and from this all 0s to all 1s, the entries needs to be forwarded to C what it means is, I do not care what is the value of the remaining 6 bits, so just a 192, 193, 194, everything else till there 255 need to be forwarded to a port number C, so that is how the table is actually constructed.

(Refer Slide Time: 23:15)

Hardware Route Lookup DIR-24-8-BASIC



- ❑ Example lookup
- ❑ 10.54.22.147
- ❑ 10.54.34.23
- ❑ 10.54.34.194

24
2



Hardware Route Lookup DIR-24-8-BASIC



- ❑ Let's assume there are 3 prefixes
- ❑ 10.54/16 - A
- ❑ 10.54.34/24 - B
- ❑ 10.54.34.192/26 - C



So, let us take an example of a particular IP address and how exactly the lookup operation or the search operation in this two table structure is actually done. The first IP address that I get is a 10.54.22.147, so we need to look into in which series this is actually falling. So, the IP address is always 32 bits, but the way you do the comparison is based on whether it falls under the first series, second series, or the third series.

So, if you look at this diagram, 10.54.34 is the place where the second level memory reference is required, but what we have is 10.54.22 that entry is somewhere here, so the 10.54.22 is somewhere here, so that is actually falling into the first series. So, the entry says that the best match that you can find is 10.54/16 prefix. So, you forward it to port number A.

Similarly, the second 10.54.34.23 is falling under the second case, and we need to examine further whether it is going into the third range or not. So, 23 is somewhere here 1 2 3 4 and somewhere here is $120 \times 256 + 23$, and remaining up to 191 it is falling here, so you need to forward it to port number B using the second prefix series which is 10.54.34. And the third one, 10.54.34.194 that, is actually falling here the next entry to this one is $120 \times 256 + 194$ you need to forward it to port number C. So, that is how this table lookup is done.

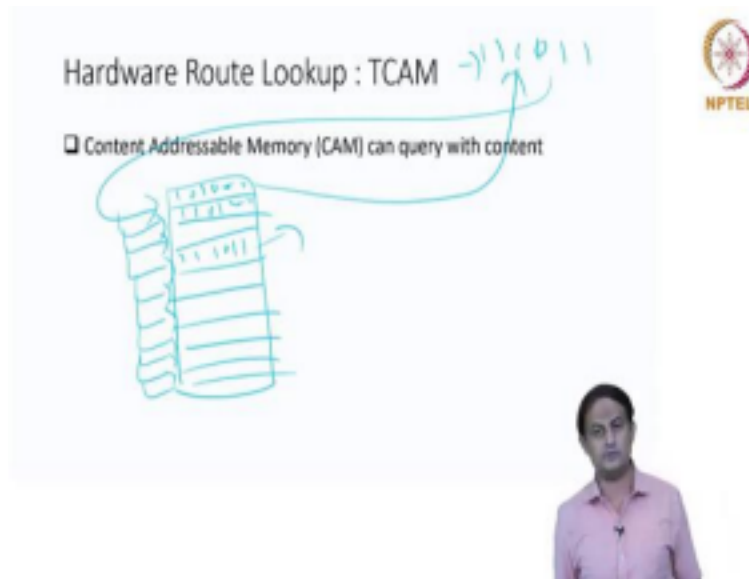
So, let me summarize what we discussed for this 2-level memory-based hardware lookup operation. The first memory is holding the entries for the 2^{24} number of the cases, so if any prefix is falling short of the 24 bits, then you do an expansion and create entries for all of them, and whatever the expansion that you do, for all of them you need to write the port number of the same series, which you have done the expansion.

And if the prefix is more than 24 bits, then the second level table reference is required or the memory reference is required and that is indicated by writing 1 in the first column and then you for every entry in the first table memory reference, you have got 256 entries in the second level table. So, if the entries represent less than 24 bits, then the port number information is kept in the third column, otherwise, that stores the index to the starting point for the expansion of the series which is indicated in that particular IP prefix.

So, that is how with the two memory references, first one you find out in 24 bit table and then the next one in the 8-bit table. So, that is how this lookup is done. So, although this lookup is able to do the quick search operation by having two level memory structure, so the number of the entries 2^{24} entries in the first table itself is quite large in number.

So, 2^{24} entries are quite large in number, so by virtue of that, sometimes this table itself will become bulky also if you go to the IP version 6 then surely this 24-bit table will not work it works only for the IP version 4 address, so that is why this is not used in practice, nevertheless this is the first attempt which is meant to construct the route lookup operation inside the hardware. So, if this is not the best way to do the lookup then what else I can do?

(Refer Slide Time: 28:06)



So, if you can recollect our previous discussion on the ternary content addressable memory, we saw that TCAM can actually do the comparisons by taking prefixes into account, basically some of the bits you can omit and then do the comparison. So, TCAM is actually having the entries, it is a table and it has got entries for some prefix position, so I can say that 1010* is the first entry 1101* is the second entry, and so forth.

So, what it means is that given your IP addresses still can be 32 bits, for the sake of convenience I will write 6 bits here, you can give an IP address of 6 bits ignoring the remaining 6 bits, you can still do the comparison. Here, in this case, 111011 is not matching to any one of them maybe 111011 and something like this is there done, it can find out. So, this is the best match that I can find for this particular IP address. So, one of the key features of this TCAM is although it is doing the content search in the memory location, the beauty of this TCAM is it can do this search operation in $O(1)$ time or in constant time.

So, what it means is the hardware of the TCAM memory has been designed to do a parallel comparison. So, you take the IP address that is given and the hardware circuitry parallelly initiates the comparison operation by taking contents from each of these memory locations. So, whether this IP address matches to the what is the entry in the first row, what is the entry in the second row and what is the entry in the third row and so forth.

So, a parallel comparison is done immediately after this comparison, so it takes a deterministic amount of time to do one comparison. So, taking the entry of this one and with the comparison

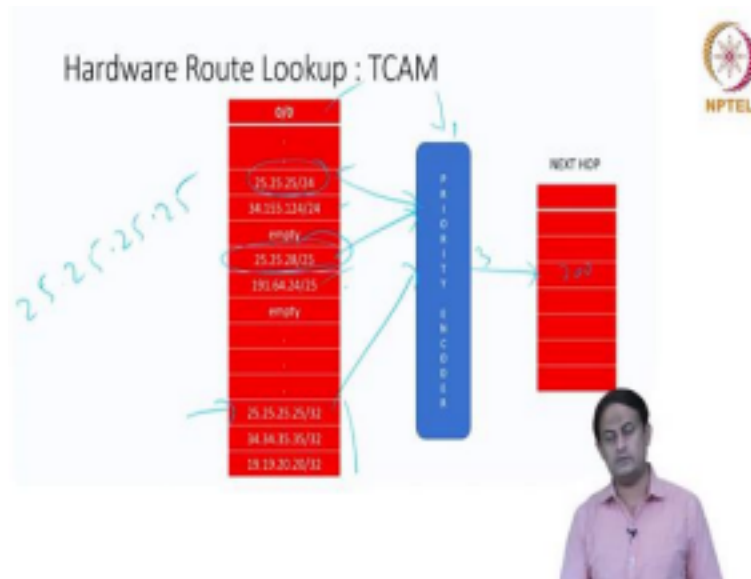
of this one, it will take the deterministic amount of time. And similarly, the second entry will also take a similar amount of time, in effect with the deterministic amount of time taken for one comparison would be sufficient enough and that is the time taken part of the overall search operation because these operations are done in the parallel comparisons.

(Refer Slide Time: 30:48)



So, that is why TCAM is actually a preferred way of implementing the lookup operation and moreover, it works with the prefix ignoring some of the bits that perfectly work for our case as well.

(Refer Slide Time: 31:06)



So, here is an example of how a TCAM entry might look like, so I might have one prefix indicating 0/0 that is the default route entry so anything that does not match the rest of the entries matches with this one and then somewhere here I might have a prefix 25.25.25/24 that is a 24 bit prefix and something else 25.25.28/25, 191.164.24/25 here is the next entry and so forth. I might have a 32-bit prefix I might have 24 bit prefix, I might have 25, 24 whatever number of the prefixes, I might have all of them and I will be able to store it in the TCAM memory.

So, when I see /25 what it means first 25 bits, I will consider, and the remaining bits I will ignore out of the 32 bits, assuming this is the IP version 4 addresses. And then what might happen is if I get an IP address, more than one entry can find a match. So, for example, if I get an IP address 25.25.25.25, so in this case so this is the entry that is going to match 25.25.25.25/32 this is the exact 32-bit address and interestingly, you might also find a match here 25.25.28/25 and you might find a match here 25.25.25/24.

So, the TCAM will come back and say I found a match here, I found a match here, and I found a match here. Now, the question is which entry I need to take, so there are three prefixes which are matching for this particular IP address, so whether to go with the third one, second one, or the first one, so the resolution of whether I should go with the first match or whether I should go with the second match or the third match is resolved by this priority encoder.

So, every match that is found is actually packed into this priority encoder and that will resolve the conflict. So, whether the first entry, second entry, or the third entry, so here in this case, the third entry is the longest, the priority encoder will come back and then tell that the third entry is the best longest prefix that is matched here and you need to go to this particular location and find out what is the port number information that is written there and you need to forward this particular packet to that port number.

So, here, in this case, all the multiple entries match, the third one gets the preference, and the index that is indicated by that third prefix entry in this table would take the preference, and the packet is actually forwarded to that, so that is how actually this TCAM actually works.

So, I repeat, the first level entry is basically in which a variable number length prefixes are stored in the TCAM memory given an IP address that IP address is spread into this TCAM memory it does the parallel comparisons and it might happen that more than one entry will match for the given IP address and all those entries will be fed into priority encoder and that encoder will figure out which is the longest prefix that is matched and then it will indicate this is the index number of the highest priority entry in the table which you need to make the routing decision or forwarding decision.

(Refer Slide Time: 35:10)

The slide is titled "Hardware Route Lookup : TCAM". It features a list of limitations under the heading "Limitations":

- ❑ Lookup Performance: $O(1)$
- ❑ Limitations
 - ❑ Large size
 - ❑ Power consumption
 - ❑ Heat dissipation
 - ❑ Limited capacity

In the bottom right corner, there is a video inset showing a man in a pink shirt speaking. The HPTEL logo is visible in the top right corner of the slide.

So, this is a nice feature, so the lookup operation is done in constant time and it is of the order of

one now the problem is solved, but unfortunately, using the TCAM memory will not solve the lookup operation completely. There are certain practical limitations of the TCAM memory, one is it is of large size, because of the parallel comparison that the TCAM memory is doing the hardware circuitry itself is bulky in nature, so it might have a large number of flip-flops embedded in it and by virtue of it, its physical size, the hardware chip size actually increases and if the chip size increases then you require larger space for the router and if the router size increases then the other realistic cost and others will come, it will occupy a larger space, the larger lag is required and so forth.

So, it requires a lot of investment and one of the objectives of the hardware Engineers is to minimize the size of the device so that in a compact way, it does not occupy as much space and also there are other advantages as well. So, the second major disadvantage of the TCAM is it consumes a lot of power, so as the number of the Hardware's elements inside the chip increases, then the larger amount of the power is consumed, and that will actually dissipate a lot amount of the heat, so on one side it consumes the hardware and the chip itself consumes a lot of power and in effect, if it generates more it dissipates more heat then you require larger cooling equipment, larger capacity cooling equipment to keep this device cool.

So, then that cooling equipment itself will take draw energy or electricity, and the larger the capacity of the cooling equipment, the larger the power consumption you have got, and then the runtime cost of using this TCAM-based memory itself will multiply. So, first of all, it requires a large size and then it is actually drawing more power and then in addition to that it is actually adding more power consumption in terms of the cooling equipment that is why because of these three reasons, the practical chipsets that are available for the TCAM memory is very limited in capacity.

So, you may not be able to find TCAM chips that can accommodate all the prefixes that are available inside your routing table. So, now the question is, your routing table might have entries, but because owing to the limitation of the TCAM memory size, I may not be able to fit all entire entries or all entries in my routing table into TCAM memory.

(Refer Slide Time: 38:27)

Hardware Route Lookup : TCAM Table Pruning



S.No	Prefix	Netmask	Next Hop
P1	10001000	11111000	4
P2	11011000	11110000	2
P3	11000110	11110000	2
P4	11100000	11110000	3
P5	11111100	11110000	3
P6	10101000	11110000	5
P7	10000110	11111000	4

S.No	Prefix	Next Hop
P1	10001*	4
P2	1101*	2
P3	1100*	2
P4	1110*	3
P5	1111*	3
P6	1010*	5
P7	1000*	4

S.No	Prefix	Next Hop
P1 and P7	1000*	4
P2 and P3	110*	2
P4 and P5	111*	3
P6	1010*	5



Now, what do I do? So, that is where the opportunities for TCAM pruning actually come into the picture where we look for opportunities where multiple entries on the routing table can be merged and one single prefix can be created. So, here is a table which is actually having 7 prefixes, and for the sake of convenience I have taken the 8-bit prefixes, and the third column is actually showing the netmask. The fourth column indicates the port number on which this particular packet needs to be forwarded. So, as usual, the first thing is to convert these entries into prefix format, taking the netmask into account.

So, for example, the next mask for the first prefix says you take the first five entries and ignore the next three bits, so I got a prefix of 10001*. Similarly, the second entry is that take the first four bits and ignore the next four bits I get the prefix of 1101*, so the netmask basically tells you how many 1s are there in the netmask those many bits you take, and the remaining bits whichever are 0s you ignore that.

So, that is how you convert the entries in the routing table into prefix format, that is the first exercise, and retain the fourth column as it is. So, the netmask now disappears from the picture. And now, what I do is for the same Next Hop information, same port number here, the port number 4 is there and this is port number 4, any two prefixes which are having the same output port number or the next hop information you see whether these two can be merged and get one entry.

So, I can merge two entries only when they differ by 1 bit so 1000 is there and 1000 is there, and the fifth bit is here is 1, and the fifth bit here is 0, I can merge these two entries and then get one

entry here. So, it does not matter whether the fifth bit is 0 or 1, I still need to forward it to port number 4.

So, still going by the same argument, I can also merge P_2 and P_3 as they are also differing by one bit, so I create one entry and both of them are saying that you need to forward it to port number 2, I create one entry here and P_4 and P_5 by the same argument can also be merged as both of them are saying you need to forward it to port number 3 P_6 cannot be merged with anyone, so, I will retain that as it is.

So, like this, I look for opportunities for merging, and by merging those entries, I can reduce the number of the entries in my routing table, and if I can reduce the number of the entries in the routing table I can probably fit it into the limited size TCAM memory that I have got. So, that is how the routing tables constructed by the routing algorithms are taken and then offline exercise for merging or pruning is done and you get a reduced table size, and you load it into the TCAM memory for doing the lookup operation in real-time.

And remember every time you change the routing information, the routing table structure changes, and then you need to rerun this pruning algorithm again one more time to accommodate the changed prefixes in the routing table every time there is some change in the routing table you need to run this.