Address Summarization for Distance Vector Protocols

Erik van Zijst
erik@marketxs.com
MarketXS
August 28th 2003

Abstract

Adaptive routing protocols such as distance vector[i] and link-state[ii] protocols maintain routing tables that contain a routing entry to each destination in the complete network. This way each packet can be forwarded directly to the neighbor that is in the shortest path toward the destination. When something changes in the network topology or available bandwidth, this may trigger changes to certain routing entries. In link-state protocols these changes need to be propagated to all nodes in the network as every host maintains an image of the complete network topology. With distance vector protocols the situation is usually slightly better, but some changes also need to be propagated throughout the entire network. The obvious problem is the limited scalability of these routing tables. When the network grows in size, so do the routing tables. This requires more bandwidth to exchange the necessary routing data between hosts. Also, the more links there are, the more changes occur in the network. These factors severely limit the scalability of these protocols. In this text we propose a technique that can significantly increase the scalability of large networks by substituting logical ranges of hosts in the routing table by one single, condensed entry. The farther a destination host is away, the more efficient it can be condensed together with other remote hosts. We call this *address summarization*. Because this text is targeted to the new MarketXS multicasting network that uses a distance vector protocol at its base, we will only discuss the proposed technique in the context of such protocol, although it can also be applied to link-state protocols.

Location Based Addressing

Traditional distance vector protocols only require host addresses to be unique. This results in routing tables similar to the one below.

dest	hop	cost
P	R	135
Q	R	137
A	A	12

Illustration 1: Routing table fragment of a typical distance vector protocol.

Given the addresses of the other nodes, there is no way to tell approximately where these hosts are located, although looking at both the distance (cost) together with the direction (hop or neighbor) might reveal that both P and Q are located close to each other at the far end of the network. If we consider the above routing table to be a fragment of the routing table of node C in the network depicted below, we can see that P, Q and R belong to the same, remote *subnet*.

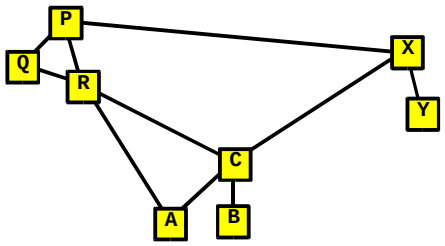

Illustration 2: Example network topology with abstract node addresses.

The paths to the individual nodes in the subnet largely overlap (for a large part of the total cost if we consider the link between C and R to be an expensive, long-distance connection). To capture the notion of subnets, we assign a name to every subnet and prepend it to the host address. We call the subnets S1, S2 and S3 and as a result the fully qualified name of node C becomes S2.C. This makes the addresses location based.

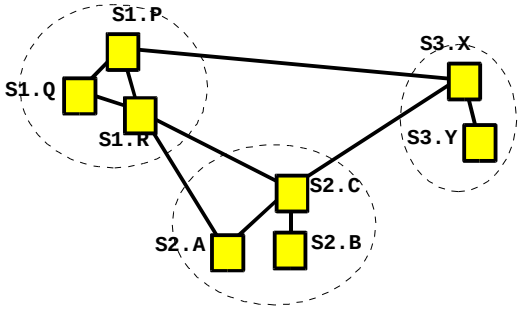

Illustration 3: Example network with hierarchical, location-based node addresses that define logical subnets or domains.

Address Summarization

Given the logical domains that now group clusters of nearby hosts, each host can treat domains other than its own as a single entity and use a wildcard address that matches every host inside the domain. This reduces the size of the routing table, as well as the amount of routing information that needs to be exchanged between the nodes when the topology changes inside a domain. When a new host is added to domain `S1`, there is no need to propagate that information to the other domains, as they already have a wildcard entry that will match the new address. In illustration 4 we applied this condensation or summarization process to a fragment of the routing table of node `S2.C`.

dest	*hop*	*cost*
S1.*	S1.R	132
S3.*	S3.X	157
S2.A	S2.A	12

Illustration 4: Routing table fragment with address summarization applied.

The cost metric that is listed in the wildcard entries is the shortest path to any host inside the domain. In our example that equals the cost to reach `S1.R` from `S2.C`. If we assume that both inter-domain links `S2.C` – `S1.R` and `S2.A` – `S1.R` have the same weight or cost, `S2.C` will route all traffic for domain `S1` through neighbor `S1.R`.

When nodes exchange distance vectors, either because of a link change or as a normal, scheduled, periodic exchange, the receiving node first summarizes the destination address of each distance vector relative to its own address. It matches the destination address with its own address field by field[1] and when a field does not match, the rest of the address is substituted by a wildcard. When `S2.C` receives a distance vector from `S1.R` that contains a path to destination `S1.P`, it is changed to `S1.*` upon arrival at node `S2.C`. This is because the first field differs from the first field of the local address and as such the rest of the fields are replaced by a wildcard. This wildcard entry is then fed to the distance vector core that checks whether the new cost (path length) is shorter than the cost of the entry that was already in the routing table. In our example there already is an `S1.*` wildcard entry in the routing table that was derived from neighbor destination `S1.R`. Since the path to `S1.P` runs through `S1.R`, the path to `S1.R` will always be shorter than the path to `S1.P`, so the entry in the routing table will not be changed.

In general, when the local address is `1.2.3.4` and an incoming distance vector advertises destination `1.2.2.2`, it will be summarized to `1.2.2.*`. Destination `2.1.6` becomes `2.*`, destination `1.2.3.4.5.6` becomes `1.2.3.4.5.*`, destination `1.2.3.5` stays `1.2.3.5` and destination `1.2.3.4.5` also stays `1.2.3.4.5`.

Note that `1.2.3` and `1.2.3.*` are two different addresses. The first only matches the exact address `1.2.3` while the second is a wildcard that matches everything that starts with `1.2.3` and has at least 4 fields. This includes `1.2.3.4` and `1.2.3.4.5.6` but not `1.2.3`.

Increased Stretch Factor

Summarization means we maintain only a single path for a range of remote nodes. The consequence of this is that packets will not always be routed according to the real shortest path between source and destination. We consider the example network of illustration 3. Node `S1.R` receives the distance vectors of both `S2.A` and `S2.C` and after summarization learns that both neighbors offer a route for wildcard `S2.*`. If we assume that both inter-domain links (`S1.R` – `S2.A` and `S1.R` – `S2.C`) have equal costs, then `S1.R` will choose `S2.A` to be the preferred hop for `S2.*` because of the fact that its address logically comes first. When `S1.R` now needs to forward a packet to `S2.B`, it uses the `S2.*` wildcard entry and forwards the packet to neighbor `S2.A`. Unfortunately this is

1 Note that a network will typically have a hierarchy of multiple nested domains. This is much like the real world where a street is a domain inside a city and a city is a domain inside a country, etc.

not the shortest path to S2.B as S2.A first has to route the packet through S2.C. Instead, S1.R should have sent the packet directly through neighbor S2.C. This ratio between the length of the actual path and the distance between the endpoints is called the *stretch factor*. The path between S1.R and S2.B *stretches* as a consequence of the address summarization[iii].

Summarization Levels

Path stretching can occur when packets are forwarded between nodes that are in different logical domains. An entire subnet is treated as a single node with several outgoing links. Because the virtual node often contains a large number of nodes, connected by its own internal network, it is sometimes better to choose a different inter-domain link when sending packets to the domain.

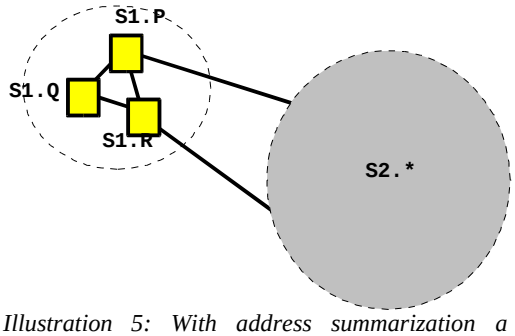

Illustration 5: With address summarization a different domain is seen as a single wildcard entity.

Consider illustration 5. When node S1.P needs to forward a packet for destination S2.F, it will send the packet directly down its inter-domain link to the S2 domain. In this example we assume both inter-domain links to have equal costs. Illustration 6 shows the inner network of the S2 domain and it becomes clear the stretch factor is quite high for packets from S1.P to S2.F, as the shortest path runs through S1.R.

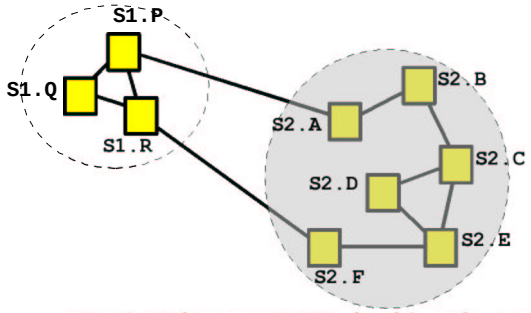

Illustration 6: With summarization level 2, nodes can look one level deep into a foreign domain. This reduces the stretch factor.

We can manipulate the stretch factor by changing the level of summarization. By default, an address is summarized after the first field that differs from the local address. However, if we make that the second field, we can look inside other domains for one level. A host with address 1.2.3.4 will then summarize 2.3.4.5 into 2.3.* and 1.3.4.5 into 1.3.4.*. Doing this at least on the border nodes that have the inter-domain links might be a good way to reduce the stretch factor. As the network was designed to be administrated by independent parties, administrators are free to experiment with different levels of summarization without jeopardizing the other subnets or domains.

Addressing Structure

In this article we use dotted strings as node addresses. The advantage of such a scheme is the large address space and flexible naming. New sub-domains can be added at any time as there is no need for a predefined or fixed number of hierarchical levels. Also, if individual address fields are not bound to a limited number of characters, they can contain geographical locations or company names. An example might be nl.amsterdam.level3.marketxs.pc3, or nl.ams.l3.mxs.pc3 to keep the address relatively short. The drawback of using this kind of unrestricted addresses is the amount of bytes they require. Because every packet always contains a source and a destination address, the amount of overhead per packet can be excessive. Early demo versions of the software could use these kind of addresses by letting the first byte of each address header specify the length of the address in bytes, followed by the address itself as a normal 8 bit ASCII string with a maximum of, say, 256 bytes. Null-terminated strings might be an alternative. For the final product it will probably be desirable to come up with a different, more lightweight way of representing hierarchical addresses. A straightforward way of assigning addresses could be to use a sufficiently large address space and divide it into logical subsets, much like IPv4 and IPv6 addresses. If we use a 64 bit address space and divide it into 8 bit fields (identical to IPv6) used to identify logical subnets, we get addresses like 120-23-61-201-43-146-128-132. To make distinct addresses, IPv4 uses the dot "." for separation, IPv6 uses the colon ":" and we use a hyphen "-". When each address field represents exactly one logical subnet level, this scheme provides a simple, but at the same time somewhat limited way of addressing. It means that the complete network can address as many

individual hosts as IPv6, while grouping them in at most 8 levels of sub domains, where each sub domain can contain a maximum of 256 hosts or sub domains.
We can clarify the numerical addressing scheme with illustration 7.

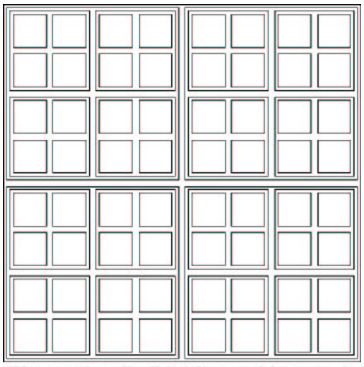

Illustration 7: Dividing addresses in logical domains and sub domains.

In this figure we have taken the whole address range and divided it in 4 subnets or domains. Each of these domains have again been divided into 4 smaller domains, until every individual host address has its own private domain. If we do this for 4 levels deep, we can address a total of $4^4 = 256$ hosts. If we apply normal address summarization on a host's routing table, that would lead to a maximum of 13 routing entries in total if we include an entry to ourselves. This is 3 entries to identify the 3 remote subnets on the highest level, another 3 to identify the remote subnets on the next level, etc. The deepest level includes our own address, which makes up for 4 entries. We can see that the number of routing entries is a function of the size of each domain and the total depth of the hierarchy:

$$(ds-1)*hd+1=se$$

while

$$ds^{hd}=h$$

Where *ds* represents the number of hosts or sub domains in a domain, *hd* stands for the maximum depth of the domain hierarchy and *se* is the total number of routing entries after summarization. In the second formula, *h* stands for the total number of hosts. In our example *se* would be (4 – 1) * 4 + 1 = 13.
If we increase the domain size *ds* to 16, while decreasing the hierarchy depth *hd* to 2, then the total number of hosts *h* still adds up to 256 ($16^2 = 256$), but the number of routing entries *se* increases to (16 – 1) * 2 + 1 = 31. If we work the other way around and decrease *ds* to 2, while increasing the hierarchy depth *hd* to 8, we can still address the same 256 hosts, but we would need only (2 – 1) * 8 + 1 = 9 routing entries. Obviously the stretch factor is inversely proportional to the size of the routing table, so a small *se* is not necessarily a good thing. With a 64 bit address space and 8 bit domains, the largest routing table a host would ever have to manage is (256 – 1) * 8 + 1 = 2041 which seems like a reasonable maximum. If we make the domain size 4 bit (16 hosts) and the depth 16 levels, the maximum routing table size is only (16 – 1) * 16 + 1 = 241 while we can still reach 2^{64} individual hosts.

Conclusions

In this text we have identified the scalability problem of standard distance vector protocols. Because each host maintains some information on every other host in the entire network, the overhead introduced to keep all routing tables in sync grows proportionally to the number of hosts in the network. This includes increased memory and processor capacity to store and search the larger routing tables and an increased number of routing updates to propagate all network changes to relevant hosts. When the network grows beyond thousands or maybe even millions of hosts, this overhead quickly grows out of proportions, making the network unusable.
We have proposed a technique we call address summarization in which we group ranges of addresses into logical domains and sub domains, while using wildcard addresses to refer to domains instead of their individual hosts. The advantage of this scheme is the increased scalability. We have proposed two hierarchical addressing structures, a flexible one that uses human readable dotted strings but suffers from a high overhead factor and a numerical addressing scheme that is more likely to be useable in a production environment.
Depending on the configuration of the numerical scheme, we could let our distance vector protocol dynamically manage 2^{64} hosts with only hundreds to thousands routing table entries. With appropriate sorting and search algorithms, together with modern computers, we consider routing tables with these sizes manageable. We also identified the main drawbacks of our approach. Increased scalability means that packets often do not travel according to their real shortest path. We call this deficiency the stretch factor. Another downside is the less flexible address allocation for new hosts and domains.

---

i L.R. Ford and D.R. Fulkerson, *Flows in*

Networks, Princeton University Press, 1962

ii J.Moy, *OSPF Version 2*, Network Working Group Internet Draft, November 1992

iii D. Peleg and E. Upfal, *A Trade-Off between Space and Efficiency for Routing Tables*, Journal of the ACM, Vol. 36, No. 3, July 1989, pp. 510-530