Network Working Group
Request for Comments: 1326

P. Tsuchiya
Bellcore

May 1992

Mutual Encapsulation Considered Dangerous

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Abstract

This memo describes a packet explosion problem that can occur with mutual encapsulation of protocols (A encapsulates B and B encapsulates A).

The Current Environment

In spite of international standardization efforts to the contrary, we are these days seeing a plethora of different protocols, both standard and proprietary, each designed to fill a technical or marketing niche. The end result is that they eventually butt up against each other and are expected to interwork in some fashion.

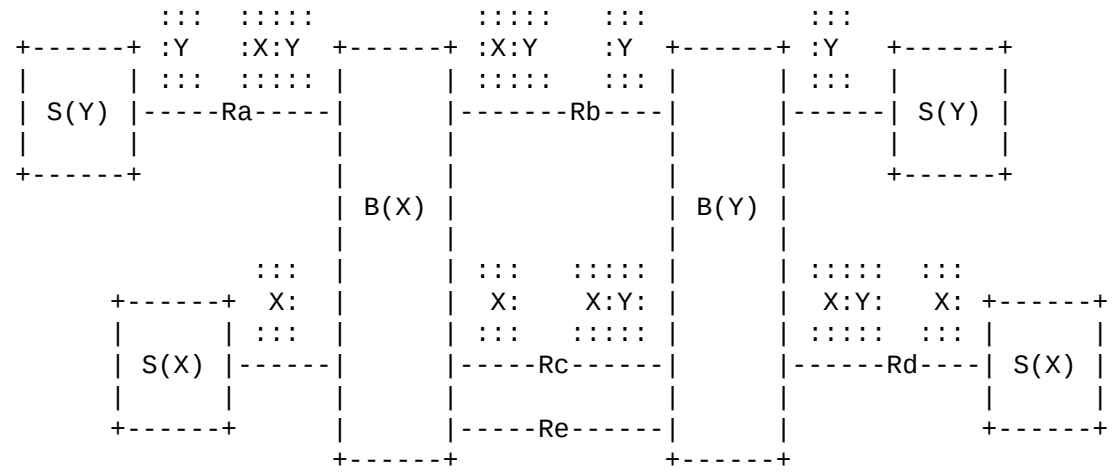
One approach to this interworking is to encapsulate one protocol within another. This has resulted in cases of mutual encapsulation, where protocol A runs over protocol B in some cases, and protocol B runs over protocol A in other cases. For example, there exists cases of both IP over AppleTalk and AppleTalk over IP. (The term mutual encapsulation comes from the paper by Shoch, Cohen, and Taft, called Mutual Encapsulation of Internetwork Protocols", Computer Networks 5, North-Holland, 1981, 287-300. The problem identified in this RFC is not mentioned in the Shoch et. al. paper.)

If there are not already other instances of mutual encapsulation, there will likely be more in the future. This is particularly true with respect to the various internet protocols, such as IP, CLNP, AppleTalk, IPX, DECNET, and so on.

The Problem

The problem with mutual encapsulation is the following. Consider the topology shown in Figure 1. We see two backbones and four stubs. Backbone B(X) uses a native protocol of X (that is, it expects to receive packets with a header for protocol X). B(Y) uses a native

protocol of Y. Likewise, the right and left S(Y) stubs use protocol Y, and the right and left S(X) stubs use protocol X.



LEGEND:

::::: X:Y: ::::: A packet with protocol X encapsulated in protocol Y, moving left to right

Rx Router x

S(Y) A stub network whose native protocol is protocol Y

B(X) A backbone network whose native protocol is protocol X

FIGURE 1: MUTUAL ENCAPSULATION

Figure 1 shows how packets would travel from left S(X) to right S(X), and from right S(Y) to left S(Y). Consider a packet from left S(X) to right S(X). The packet from left S(X) has just a header of X up to the point where it reaches router Rc. Since B(Y) cannot forward header X, Rc encapsulates the packet into a Y header with a destination address of Rd. When Rd receives the packet from B(Y), it strips off the Y header and forwards the X header packet to right S(X). The reverse situation exists for packets from right S(Y) to left S(Y).

In this example Rc and Rd treat B(Y) as a lower-level subnetwork in exactly the same way that an IP router currently treats an Ethernet as a lower-level subnetwork. Note that Rc considers Rd to be the

appropriate "exit router" for packets destined for right S(X), and Rb considers Ra to be the appropriate "exit router" for packets destined for left S(Y).

Now, assume that somehow a routing loop forms such that routers in B(Y) think that Rd is reachable via Rb, Rb thinks that Rd is reachable via Re, and routers in B(X) think that Re is reachable via Rc. (This could result as a transient condition in the routing algorithm if Rd and Re crashed at the same time.) When the initial packet from left S(X) reaches Rc, it is encapsulated with Y and sent to B(Y), which forwards it onto Rb. (The notation for this packet is Y<X>, meaning that X in encapsulated in Y.)

When Rb receives Y<X> from B(Y), it encapsulates the packet in an X header to get it to Re through B(X). Now the packet has headers X<Y<X>>. In other words, the packet has two X encapsulates. When Rc receives X<Y<X>>, it again encapsulates the packet, resulting in Y<X<Y<X>>>. The packet is growing with each encapsulation.

Now, if we assume that each successive encapsulation does not preserve the hop count information in the previous header, then the packet will never expire. Worse, the packet will eventually reach the Maximum Transmission Unit (MTU) size, and will fragment. Each fragment will continue around the loop, getting successively larger until those fragments also fragment. The result is an exponential explosion in the number of looping packets!

The explosion will persist until the links are saturated, and the links will remain saturated until the loop is broken. If the looping packets dominate the link to the point where other packets, such as routing update packets or management packets, are thrown away, then the loop may not automatically break itself, thus requiring manual intervention. Once the loop is broken, the packets will quickly be flushed from the network.

Potential Fixes

The first potential fix that comes to mind is to always preserve the hop count information in the new header. Since hop count information is preserved in fragments, the explosion will not occur even if some fragmentation occurs before the hop count expires. Not all headers, however, have hop count information in them (for instance, X.25 and SMDS).

And the hop counts ranges for different protocols are different, making direct translation not always possible. For instance, AppleTalk has a maximum hop count of 16, whereas IP has up to 256. One could define a mapping whereby the hop count is lowered to fit

into the smaller range when necessary. This, however, might often result in unnecessary black holes because of overly small hop counts. There are for instance many IP paths that are longer than 16 hops.

It is worth noting that the current IP over AppleTalk Internet Draft does not preserve hop counts ("A Standard for the Transmission of Internet Packets Over AppleTalk Networks").

Another potential fix is to have routers peek into network layer headers to see if the planned encapsulation already exists. For instance, in the example of Figure 1, when Rb receives Y<X>, it would see what Y had encapsulated (for instance by looking at the protocol id field of X's header), notice that X has already been encapsulated, and throw away the packet. If the encapsulation loop involves more than two protocols, then the router may have to peek into successive network layer headers. It would quit when it finally got to a transport layer header.

There are several pitfalls with this approach. First, it is always possible that a network layer protocol is being encapsulated within a transport layer protocol, thus I suppose requiring that the router continue to peek even above the transport layer.

Second, the router may not recognize one of the network layer headers, thus preventing it from peeking any further. For instance, consider a loop involving three routers Rxy, Ryz, and Rzx, and three protocols X, Y, and Z (the subscripts on the routers R denote which protocols the router recognizes). After the first loop, Rxy receives X<Z<Y<X>>>. Since Rxy does not recognize Z, it cannot peek beyond Z to discover the embedded Y header.

Third, a router may be encrypting the packet that it sends to its peer, such as is done with Blacker routers. For instance, Rc might be encrypting packets that it encapsulates for Rd, expecting Rd to decrypt it. When Rb receives this packet (because of the loop), it cannot peek beyond the Y header.

Finally, there may be situations where it is appropriate to have multiple instances of the same header. For instance, in the nested mutual encapsulation of Figure 2, Ra will encapsulate Y in X to get it to Rd, but Rb will encapsulate X<Y> in Y to get it to Rc. In this case, it is appropriate for Rb to transmit a packet with two Y headers.

A third (somewhat hybrid) solution is to outlaw nested mutual encapsulation, employ both hop count preservation and header peeking where appropriate, and generally discourage the use of mutual encapsulation (or at least adopt the attitude that those who engage

in mutual encapsulation deserve what they get).

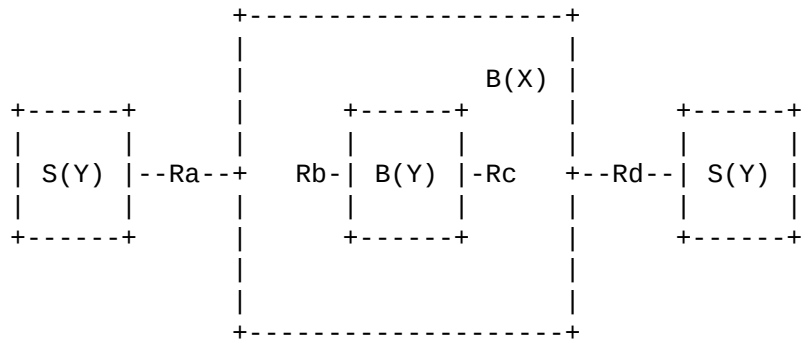


FIGURE 2: NESTED MUTUAL ENCAPSULATION

Security Considerations

Security issues are not discussed in this memo.

Author's Address

Paul Tsuchiya
Bellcore
435 South St.
MRE 2L-281
Morristown, NJ 07960

Phone: (908) 829-4484
EMail: tsuchiya@thumper.bellcore.com