This is a description of the new zephyr protocol.

The original protocol did not consider the overhead of scaling to multiple realms. So, these needs have been reconsidered, and these are the recommended protocol changes to support this.

1 Contents of Packets

A message is the conceptual entity which gets broken into packets. The *unfragmented packet* is the fragmentation which occurs when you have no limit on data size. When packets are fragmented, packet zero (the first one sent) has a full set of headers, and the rest of the packets have a reduced set.

The following fields appear at the top of every packet:

Block 0	
name	description
version	string containing version
# of header fields	number of header fields, including the first two
unique id	unique id for the notice (not packet)
sequence id	ordering of the packets for this notice

These fields appear only in a full packet (with a sequence id of 0), not in the fragments (with a sequence id other than 0) :

Block 1	
name	description
auth type	authentication type
auth length	length of auth info. (optional. See below)
auth data	auth data. (optional. See below.)
notice kind	general type of the notice
fragment count	number of fragments
class	string
slass instance	string
recipient	string
opcode	string
sender	string
sending machine	IP address of sender (may be 0)
response port	port at sending addr to use for responses
data total	number of data bytes in total message

The following fields also appear in every packet. In the full packet, they appear after block 1. In the fragment packets, they appear after block 0.

Block 2	
name	description
frag offset	offset into notice data of the data in this packet.
data length	number of data bytes in this packet
data	actual data

Notes:

- The encoding scheme chosen for the data will affect the sizes of the various fields, and will directly affect the amount of fragmentation occurring. Two candidates for encoding are ASN.1 and a tightly-packed binary protocol.
- The notice kind will likely change from ZEPH0.2.
- Sending and receiving realms are determined by the realms in the sender and recipient, respectively.
- In this version of the protocol, more levels of authentication than Yes/No/Forged may be necessary. I recommend that when the client

requests authentication, the answer be specific to the type of authentication taking place.

- It it possible that more than one type of encryption may be desired. If the authentication type is 0, then the next field should be the notice kind. The authentication list should be terminated by a null authentication type, that is, the tuple {auth type, auth length, auth data} should be repeated, with a 0 at the end. Just a zero symbolizes lack of authentication.

2 Packet Routing

Although keeping the same notice/packet format is useful in order to simplify parsing and routing, transmitting them along the same channels all the time seems incorrect. So, there should be several different paths of communication: *It is not at all clear what this paragraph means.*

- sending client → hostmanager
- hostmanager → local server
- local server → remote server
- remote server → receiving client

3 Authentication

3.1 Trust authentication

This the really basic authentication you use when you have no better way of doing it. It simply consists of one byte of data: 0 = unauthentic and 1 = authentic. Wasn't that easy?

3.2 End to End Authentication

End to end authentication ensures that the message came from the user it says it came from, but is susceptible to packet retransmission, as it doesn't check the routing of the packet. In general, however, since the entire contents of the packet are checksummed, retransmission is a fairly useless attack. Note that this authentication can't work for broadcast messages. The algorithm is as follows:

The authentication data consists of a Kerberos authenticator for the service "zephyr.zephyr@RECIPIENT_REALM" and an unencrypted checksum of the unfragmented packet.

1. The sending client generates a checksum on the unfragmented packet.
2. The sending client generates an authenticator for the service "zephyr.zephyr@RECIPIENT_REALM" and encrypts the checksum in the shared key with the remote server.
3. The sending client sends the packets to the hostmanager
4. The packets are forwarded unchanged to the local server
5. The packets are forwarded unchanged to the remote server
6. The remote server opens the authenticator and verifies the sender in the packet with the sender in the authenticator. Assuming it is valid, it uses the key in the authenticator to decrypt the checksum, and then encrypts the checksum in the key it shares with the receiving client. The modified packet is forwarded to the receiving client
7. The receiving client reassembles the fragments and computes a checksum. It then decrypts the checksum in the packet and compares the two. If they match, the authentication succeeds.

3.3 Path Authentication

Path authentication uses encryption of a checksum in order to prove that the zgram came from a zephyr server. This is susceptible to untrustable site

administrators.

The authentication consists of a Kerberos authenticator for the service "zephyr.zephyr@SENDER_REALM" and an unencrypted checksum of the unfragmented packet.

1. The sending client generates a checksum on the unfragmented packet.
2. The sending client generates an authenticator for the service "zephyr.zephyr@SENDER_REALM" and encrypts the checksum in the shared key with the local server.
3. The sending client sends the packets to the hostmanager
4. The packets are forwarded unchanged to the local server
5. The local intrarealm server verifies the identity of the sender (including the sender's IP address from the kernel). If it is verified, the checksum is decrypted with the key in the message and encrypted in the shared key with the local interrealm server. The packet is then forwarded to the remote server.
6. The remote server decrypts the checksum, encrypts it in the shared key for the client, and forwards the packets to the remote interrealm server for the appropriate realm.
7. The receiving client reassembles the fragments and computes a checksum. It then decrypts the checksum in the packet and compares the two. If they match, the authentication succeeds.

3.4 Other

Other possible encryption mechanisms are a user-to-user authentication based on Kerberos V5 or a public-key authentication based on RSA.

4 Subscription

Subscriptions take the form of messages sent to the special recipient "zephyr.zephyr" in the appropriate realm on some magic class. The server should snarf the packet and deal with it itself. The data area of the message may vary. Authentication types assume only the above authentication methods have been implemented. Other methods of authentication may be accepted in the future.

Subscription possibilities:

- User subscribes to messages from local realm.

  The body will contain the class, instance, and recipient. It will be sent to the local server using end-to-end encryption.

- User subscribes to messages from remote realm directly.

  The body will contain the class, instance, and recipient. It will be sent to the remote server using end-to-end encryption.

- User subscribes to messages from remote realm through proxy.

  The body will contain the class, instance, recipient, and an authenticator for the remote realm. It will be sent to the local server using end-to-end encryption. The local server will make a proxy subscription, using the authenticator in the body to prove that the proxy is authorized. This is most useful to reduce long-range network load for a broadcast instance across realms.