INTERNET-DRAFT Iljitsch van Beijnum
Jul 11, 2005

Shim6 Reachability Detection
draft-ietf-shim6-reach-detect-01.txt

Abstract

The shim6 working group is developing a mechanism that allows multihoming by using multiple addresses. When communication between the initially chosen addresses for a transport session is no longer possible, a "shim" layer makes it possible to switch to a different set of addresses without breaking current transport protocol assumptions. This draft discusses the issues of detecting failures in a currently used address pair between two hosts and picking a new address pair to be used when a failure occurs. The input for these processes are ordered lists of local and remote addresses that are reasonably likely to work. (I.e., not include addresses that are known to be unreachable for local reasons.) These lists

must be available at both ends of the communication, although the ordering may differ. Building these address lists from locally available information and synchronizing them with the remote end are outside the scope of this document.

This text is for the most part based on discussions on the multi6 list, several multi6 design team lists and the shim6 list, with notable contributions from Erik Nordmark, Marcelo Bagnulo and Jari Arkko. Suggestions and additions are more than welcome.

1 Introduction

A naive implementation of an (un)reachability detection mechanism could just probe all possible paths between two hosts periodically. A "path" is defined as a combination of a source address for host A and a destination address for host B. In hop-by-hop forwarding the source address doesn't have any effect on reachability, but in the presence of filters or source address based routing, it may. And although links almost always work in two directions, routing protocols and filters only work in one direction so unidirectional reachability can happen. Without additional mechanisms, the practice of ingress filtering by ISPs makes unidirectional connectivity likely. Being able to use the working leg in a unidirectional path is useful, it's not an essential requirement. It is essential, however, to avoid assuming bidirectional connectivity when there is in fact a unidirectional failure.

Exploring the full set of communication options between two hosts that both have two or more addresses is an expensive operation as the number of combinations to be explored increases very quickly with the number of addresses. For instance, with two addresses on both sides, there are four possible address pairs. Since we can't assume that reachability in one direction automatically means reachability for the complement pair in the other direction, the total number of two-way combinations is eight. (Combinations = nA * nB * 2.)

An important observation in multihoming is that failures are relatively infrequent, so that a path that worked a few seconds ago is very likely to work now as well. So it makes sense to have a light-weight protocol that confirms existing reachability, and only invoke the much heavier protocol that can determine full reachability when a there is a suspected failure.

2 Determining reachability for the current pair

Reachability for the currently used address pair in a shim context is determined by making sure that whenever there is data traffic in one direction, there is also traffic in the other direction. This can be data traffic as well, but also transport layer acknowledgments or a shim reachability keepalive if there is no

other traffic. This way, it is no longer possible to have traffic in only one direction, so whenever there is data traffic going out, but there are no return packets, there must be a failure, so the full path exploration mechanism is started.

A more detailed description of the current pair reachability evaluation mechanism:

1. The base timing unit for this mechanism is named ShimKeepT. Until a negotiation mechanism to negotiate different values for ShimKeepT becomes available, a value of 10 for ShimKeepT MUST be used.

2. Whenever outgoing packets are generated that are part of a shim context, one of two timestamps belonging to the shim context is updated: the timestamp for outgoing data packets, or the timestamp for outgoing non-data packets. The difference between the two is that data packets are packets that should generate return traffic. The host should use the information available to it to determine whether a packet is a data or a non-data packet. Examples of non-data packets are TCP ACKs and shim keepalive packets. If there is any doubt, a packet should be considered a data packet.

3. Whenever incoming packets are received that are part of a shim context, one of two timestamps belonging to the shim context is updated: the timestamp for incoming data packets, or the timestamp of incoming non-data packets. For incoming packets, it's less critical that packets are labeled as data or non-data correctly. In the absence of better information, hosts may assume that any IPv6 packet with a total length field with a value of 20 or lower is a non-data packet.

4. ShimKeepT seconds after the last data packet has been received for a context, and if no other packet has been sent within this context since the data packet has been received, a shim keepalive packet is generated for the context in question and transmitted to the correspondent. The shim keepalive packet consists of an IPv6 header and a shim header containing the context tag, but no subsequent headers. Intermediate headers may be present between the IPv6 and shim headers. A host may send the shim keepalive after fewer than ShimKeepT seconds if implementation considerations warrant this. The average time after which shim keepalives are sent must be at least ShimKeepT / 2 seconds. After potentially sending a single shim keepalive, no additional shim keepalives are sent until a data packet is received within this shim context. If the shim keepalive wasn't sent because a data or non-data packet was sent since the last received data packet, no shim keepalives are sent.

5. When after a timeout period since the last transmission of a data packet no packets were received from the correspondent within this context, a full reachability exploration is started. The

timeout period is ShimKeepT seconds plus additional time to accommodate for a round trip and regular variations in network-related functions. In the absence of better information, a timeout of at least ShimKeepT + 2 seconds but no more than ShimKeepT + 5 seconds is recommended.

3 Address pair exploration

In its essence, address pair exploration is very simple: just send probes using every possible address pair, wait for something to come back and possibly consider the round trip time. In practice, testing the full combination of all source addresses and all destination addresses is very undesirable because of the large number of packets involved. This can be especially harmful when a lot of hosts on a link start doing this for many of their correspondents at the same time when there is a failure further upstream.

In order to arrive at a desired outcome more quickly and with less packets, and also to accommodate traffic engineering needs, we'll assume a model where each address (source or destination) has two preference values: p1 and p2. Addresses within the same set (source or destination) are ranked by their p1 value, where a higher p1 means that the address is more preferred. When there are multiple addresses with the same p1 value, an address is selected at random from the group with the same p1 value, where the likelihood of selecting any given address is relative to its p2 value compared to the sum of all p2 values. So if addresses A, B and C have the same p1 value and p2 values of 10, 30 and 60 for a total of 100, the chance that A is selected is 10%, the chance that B is selected is 30% and the chance that C is selected is 60%.

Note that preference information may be related to type of service. So different context with different type of service requirements may see different p1 and p2 values for a given address.

When a host suspects that there is a failure for a context, it gathers the set of possible source addresses and the set of possible destination addresses. Both sets are ordered such that each next address has an equal or lower p1 value. Addresses with the same p1 value are further ordered as per any heuristics that the host may employ, such as longest prefix matches on known working and/or known not working addresses along with the p2 value. The p2 value is considered relatively weak, and breaking p2 ordering is allowed if there is a sufficient reason for this. However, in the absence of other information, p2 ordering should be used. P1 ordering overrules any other information except a recent reachability failure for the address in question. In addition to this, the most recently used address is put in front of the list.

From the lists of eligible source and destination addresses, the

host creates a list of source/destination address pairs, along with a combined preference value for this address pair. The calculation of the preference value is implementation specific, with the only requirement being that when one address pair has a higher p1 for both the source and destination address than another pair, the pair with the higher p1 values also has a higher combined pair preference value.

The list of address pairs from different contexts is combined into a host-wide list of address pairs. The preference values are updated to take into consideration the number of contexts that is interested in the pair. The specifics of calculating the resulting host-wide preference value are left upto the implementation, but implementations SHOULD try, within reason, to avoid using address pairs with lower p1 values when pairs with higher p1 values are available for a context. Context-specific address pair preferences may be normalized prior to calculating host-wide address pair preference values. (So when context A has pairs P and Q with p1 values 10 and 1, while context B has pairs R and S with p1 values 7 and 4, the values for P and R are changed to 2 and the values for Q and S to 1.)

The host now starts probing address pairs, in order from the pair with the highest pair preference to the pair with the lowest pair preference. When all address pairs have been tested, testing restarts from the pair with the highest preference. New pairs that become available are put in the list before pairs that have been probed already, regardless of the preference values. However, both the group of address pairs that haven't been probed and the group of address pairs that have may be reordered to reflect the preference values, as long as reordering is done such that starvation doesn't occur.

When a probe is answered by the correspondent, the context that use the address pair in question are informed so they can start remapping address is outgoing packets to the pair in question. (All of this also happens when there is a working pair but an address pair with at least one address with a higher preference is determined to work.) At this point, the context updates its list of address pairs to probe by removing all pairs where either the source address has a lower p1 value than the p1 value of the now working source address, or the destination address has a lower p1 value than the p1 value of the now working destination address. Additionally, all address pairs where the p1 values for the source and destination addresses match the respective p1 values of the source and destination addresses in the now working pair are removed from the list. The host-wide list of address pair to probe is updated to reflect the removal of lower or equal priority addresses, so probing will only continue for pairs where at least one address has a higher p1 than the currently working pair.

The time between probes (ShimProbeT) must be chosen such that the number of probes is limited to 60 per 300 second period. When no probes have been sent for some time, an implementation may send the initial group of probes at a fairly aggressive rate. For instance, when no probes have been sent for 60 seconds, a host may send a second probe 200 ms after the first one, and increase the ShimProbeT by a factor 1.25 after every probe, until ShimProbeT reaches 5 seconds. This results in sending 5 probes in the first 2 seconds and/or 14 probes within the first 20 seconds after a failure. After that, there is one probe every 5 seconds.

When a context didn't see any outgoing data packets (see section 2) for four minutes, it removes all its address pairs from the host-wide list of address pairs.

4 Address pair exploration packet format

The address pair exploration packet may be encapsulated in different ways. An obvious way is inside a shim header. The address pair exploration packet contains the following information:

- A type field that is at least 8 bits long
- An 8 bit "number of probes sent" field
- An 8 bit "number of probes received" field
- An 8 bit "options length" field
- One or more sent probes (see below)
- Zero or more received probes (see below)
- Zero or more bytes of option data

There is currently one bit in the type field defined: the reply requested bit. If this bit is set, the other side should send a probe in reply to this probe.

The option data contains zero or more options in the following format:

- An 8 bit option type
- An 8 bit option length
- Zero or more bytes of data in this option

Sent and received probes contain data in the following format:

- Source locator/address (128 bits)
- Destination locator/address (128 bits)
- Sent timestamp (32 bits in ms resolution relative to private epoch)
- Time between reception and retransmission (32 bits in ms resolution, 0 on first transmission)
- Nonce (32 bits)
- Sequence number (32 bits)

The first and only mandatory sent probe structure contains the

addresses that are present in the current IPv6 packet along with a timestamp for the current time. Additional probe structures contain copies of earlier probes, presumably toward different addresses, with the appropriate field indicating how long ago the probe in question was sent. The received probes are copies of the last seen probes from the other side.

Note that an application must be able to infer which addresses belong to the same host in order to perform this probing correctly

5 NAT and firewall considerations

Since shim6 is chartered for IPv6 solutions only, and NAT compatibility is not expected, and by most people, not desired in IPv6, there is no requirement for this protocol to pass through Network Address Translation devices. However, the protocol may be applicable outside shim6, making NAT compatibility desirable.

It is absolutely essential that the shim6 negotiations and the reachability detection packets are passed through filters or firewalls wherever application packets are passed through. If the shim6 negotiation and reachability detection packets are filtered out, shim6 can't be used.

A more complex situation arises when the shim6 negotiation packets pass through a firewall, but the reachability detection packets are blocked. To avoid this complexity, it's highly desirable to make the shim6 negotiation and reachability detection part of the same protocol, so either both are allowed through or both are blocked. However, the same is true if this reachability detection mechanism is used in other protocols. This makes it desirable to define the reachability detection protocol such that it can be embedded in other protocols.

Since firewalls are in wide use, it's important to consider whether a new protocol will be able to pass through most firewalls without requiring changes to the filter configuration. On the other hand, it may not be possible to come up with a protocol that would be allowed through a large percentage of all firewalls without changes, so extra effort in this area may produce limited results. Also, in the long run firewall configuration will presumably be changed, so any compromises would only have short term benefits but long term downsides.

6 Security considerations

To avoid exposing information (even if it's just the fact that an address is reachable), hosts will probably want to limit themselves to taking part in reachability detection with known correspondents. This means that there must be identifying information and a nonce that is at least hard to guess but easy to check in all

reachability detection packets.

4 Document and author information

This document expires April, 2006. The latest version will always be available at http://www.muada.com/drafts/. Comments are welcome at:

Iljitsch van Beijnum

Email: iljitsch@muada.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.