Performance Analysis of the REAchability Protocol for IPv6 Multihoming

Antonio de la Oliva[1], Marcelo Bagnulo[2], Alberto García-Martínez[1], and Ignacio Soto[1]

[1] Universidad Carlos III de Madrid*
[2] Huawei Lab at Universidad Carlos III de Madrid
{aoliva,marcelo,alberto,isoto}@it.uc3m.es

Abstract. There is ongoing work on the IETF aimed to provide support for different flavors of multihoming configurations, such as SHIM6 for multihomed sites, multiple CoAs support in MIP for multihomed mobile nodes and HIP for multihomed nodes and sites. A critical aspect for all the resulting multihoming protocols is to detect failures and gain information related with the paths available between two hosts. The Failure Detection and Locator Path Exploration Protocol (in short REAchability Protocol, REAP) being defined in the SHIM6 WG of the IETF is a good candidate to be included as a reachability detection component on protocols requiring this functionality. Performance study is performed by combining analytical estimations and simulations to evaluate its behavior and tune its main parameters.

Keywords: multihoming, failure detection, SHIM6, REAP.

1 Introduction

So far, IPv4 has failed to provide a scalable solution to preserve established communications for arbitrarily small-size sites connected to the Internet through different providers after an outage occurs. This is so because the current IPv4 multihoming solution, based on the injection of BGP [BGPMULT] routes in order to make a prefix reachable through different paths, would collapse if the number of managed routing entries would increase to accommodate small sites and even hosts. In particular, this restriction prevents end hosts equipped with different access interfaces, such as IEEE 802.11, UMTS, etc, connected to different providers, to benefit from fault tolerance, traffic engineering, etc. On the other hand, the huge address space provided by IPv6 has enabled the configuration of public addresses from each of the providers of an end host. A step further is been taken in the SHIM6 WG of the IETF to develop a framework to manage in an end-to-end fashion the use of the different addresses between a communication held by two hosts. To achieve this, a SHIM6 layer is included

* This work was supported by IST FP6 Project OneLab and by the Spanish MEC through Project CAPITAL (TEC2004-05622-C04-03).

inside the IP layer to assure that the same IP address pair is provided to the upper layers to identify a given communication, while the packets flowing in the network can use different IP addresses (locators) to be able to enforce different paths. The SHIM6 [SHIM] [SHIMAPP] layers of two communicating nodes that want to benefit from the multihoming capabilities first execute a four-way handshake to exchange in a secure way the relevant information for managing the IP addresses that play the roles of identifiers and locators. Once this exchange has been performed, both SHIM6 layers use the REAP (REAchability Protocol) protocol to timely detect failures in the currently used path, and once a failure is detected to select a new path through which the communication could be continued. Note that while the REAP protocol is being defined as a component of the SHIM6 multihoming solution, it is envisioned that such a protocol could became a generic component in different scenarios in which end-to-end path validation is of paramount concern, such as HIP [HIP] or Mobile IPv6 with registration of multiple CoAs [MONAMI].It is straightforward to conclude from the framework presented above that the REAP protocol determines the performance that upper layers perceive when an outage occurs in the communication path. The path failure detection function of the REAP protocol relies on timers driven by the inspection of upper layer traffic, and by specific Keep Alive probe packets when upper layer traffic is too sparse. The current specification of the REAP protocol lacks of experimental support to properly configure the timers of the protocol and to fully understand the interaction with transport layers such as UDP or TCP. In this paper we simulate the protocol with the OPNET[1] tool and we analyze the configuration of these timers and the impact on different type of applications. The remainder of the paper is organized as follows: A description of the REAP protocol is given in section 2. The scenario used in the simulation, as well as the details of the simulation environment are shown in section 3. Section 4 presents the results obtained regarding the behavior of UDP (section 4.1) and TCP (section 4.2) protocols, providing an analysis of several application types. Finally in section 5 we provide the conclusions of our work.

2 Failure Detection and Path Exploration in the SHIM6 Architecture

The SHIM6 architecture is currently being defined by the IETF to provide multihoming between hosts with multiple provider independent addresses. The architecture defines a shim sublayer, placed in the IP layer, which is responsible for ensuring that the same local and remote addresses are provided to the upper layers for the peers involved in a given communication, while at the same time different addresses can be used to allow the usage of different paths. As a consequence, two roles are assumed by the IP addresses. The term identifier is used for addresses passed to transport and application layers, and the term locator is reserved for the actual addresses used for IP forwarding. SHIM6 defines

[1] OPNET University Program, http://www.opnet.com/services/university/

two components to manage the identifier/locator relationship in two communicating peers: the secure exchange between the peers of information related with identifiers and locators, performed by the SHIM6 protocol [SHIM], and the identification of communication failures and the exploration of alternative paths.Failure detection does not need specific tools if traffic is flowing between two hosts. On the other hand, when a node has no packets to send, it is irrelevant for the node if the locator pair is properly working or not, since it has no information to transmit. So, a potential relevant failure situation occurs when a node is sending packets but it is not receiving incoming packets. Such situation does not necessarily imply a failure, since a unidirectional flow may be being received, but this is indistinguishable from a failure without additional tests. In this case, the node needs to perform an explicit exchange of probe packets to discover if the current locator pair is properly working. This exchange is described in the REAchability Protocol, REAP [REAP] specification. The REAP protocol relies on two timers, the Keep Alive Timer and the Send Timer, and a probe message, namely the Keepalive message. The Keep Alive Timer T_{KA} is started each time a node receives a data packet from its peer, and stopped and reset, each time the node sends a packet to the peer. When the Keep Alive Timer expires, a Keep Alive message is sent to the peer. The Send Timer T_{Send}, defined roughly as three times the Keep Alive Timer plus a deviation to accommodate the Round Trip Time, is started each time the node sends a packet and stopped each time the node receives a packet from the peer. If no answer (either a Keep Alive or data packet) is received in the Send time period a failure is assumed and a locator path exploration is started. Consequently, the Send timer reflects the requirement that when a node sends a payload packet there should be some return traffic within Send Timeout seconds. On the other hand, the Keepalive timer reflects the requirement that when a node receives a payload packet there should a similar response towards the peer within Keepalive seconds (if no traffic is interchanged, there is no Keep Alive signaling). As a consequence, there is a tight relationship between the values of the timers defined by the REAP protocol and the time required by REAP to detect a failure. The current specifications suggest a value of 3 seconds for the Keepalive Timer, and of 10 seconds for the Send Timer, although these values are supported by neither analytical studies nor experimental data.Once a node detects a failure, it starts the path exploration mechanism. A Probe message is sent to test the current locator pair, and if no responses are obtained during a period of time called Retransmission Timer T_{RTx}, the nodes start sending Probes testing the rest of the available address pairs, using all possible source/destination address pairs. Currently, a sequential algorithm is defined to drive the exploration of locator pairs, behavior that will be assumed for the rest of the paper. So far the REAP specification process has focused on functionality, without paying too much attention to performance metrics in common operation conditions, such as the time required to detect a failure and recover it. An experimental analysis would provide relevant guidelines to tune the main parameters that define the REAP behavior when combined with different types of applications. In particular, interaction with applications using TCP should be considered, in order to characterize the interactions between REAP and the flow and congestion control mechanisms provided by this protocol. When UDP transport is considered, the resulting behavior is driven mainly by the application protocol; in this case relevant applications should be analyzed. In the next sections we perform some simulations that try to provide valuable information related with REAP timer configuration for applications using UDP and TCP.

3 Simulation Setup

In this section we present the scenario used to test the path failure detection functionality of the REAP protocol. Figure 1 shows two nodes, Node A and B, each one with two interfaces and an IPv6 address configured on each interface. All simulations have been performed by establishing a communication through the pair (IP_{A1}, IP_{B1}). All traffic exchanged between these IP addresses goes through Cloud 1 and 2. At a certain time, the link connecting Cloud 1 and 2 fails, this is detected by REAP and after a path exploration, the communication is continued using the IP pair (IP_{A2}, IP_{B2}). Tests performed involve the TCP

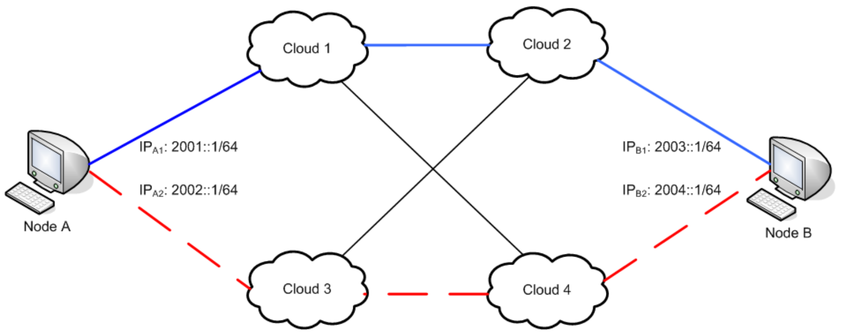


Fig. 1. Simulated Scenario

and UDP protocols. The TCP tests, designed to evaluate the TCP behavior in cases with high and low data rates, are performed using an FTP file download application and a Telnet application. The traffic used to evaluate UDP behavior corresponds to either a Voice over IP (VoIP) application showing bidirectional packet exchange or to an unidirectional voice flow. Note that unidirectional flows result in increased exchange of REAP specific packets.For TCP, the Windows XP model defined in OPNET has been used. For UDP, a VoIP conversation, using the codec G.729 with a compression delay of 0.02 seconds.

The RTT in both paths is the same, it has been implemented as a normal distribution with mean 80ms and 20ms variance. The failure event occurs at a time defined by an uniform distribution between 75 and 125 seconds. All simulations have been run for 250 seconds, the presented results are the average of 45 samples. The real values are within ±10% (on the worst case) of the estimated values with a confidence interval of 95%.

4 Analysis of the Results

In order to find the values for the REAP timers that optimize the behavior of TCP and UDP when a path failure occurs, several measures have been performed. The main metric used through the analysis is the *Application Recovery Time*. This metric is defined as the difference in time between the last packet arriving through the old IP locators (addresses) and the first packet arriving through the new ones. This metric accurately measures the time to recover from a path failure when there is continuous traffic. The analysis is structured in the following items:

– UDP behavior: To fully understand the behavior of applications using UDP, two types of traffic have been considered, bidirectional traffic (VoIP conversation) and unidirectional traffic (streaming of audio).
– TCP behavior: TCP incorporates several characteristics such as congestion control and reliability that determines the resulting performance when a valid path is provided as a result of the REAP operation. To understand the behavior of applications using TCP two traffic types have been considered, a FTP download from a server and a telnet session showing sparse traffic exchange. With these two traffic types the behavior of applications with high traffic demands and applications with low traffic profiles are considered.

4.1 UDP Behavior

Consider that a failure occurs, when the T_{Send} timer expires, the node tries to reach the peer by sending a probe to the IP address that is currently in use. This probe expires after T_{RTx} seconds. At this time, a second probe is sent to the secondary IP address. The path exploration mechanism finalizes after the exchange of 3 probes per peer, directed to the secondary IP addresses. The time required to finalize the path exploration mechanism is almost constant (there is some variation due to the RTT variance) with a value of 0.7 seconds[2]. Figure 2 shows the Recovery time for different T_{Send} ($T_{KA} = T_{Send}/3$) values and for two types of UDP applications, Voice Over IP (VoIP) and an unidirectional VoIP flow. The results follow the expected behavior, being the relation between the Recovery Time and the T_{Send} linear. This relation was expected to be linear since UDP is not reactive to path conditions and once the path is restored traffic is immediately sent through the new IP locators. Note that in figure 2 the Recovery Time of the unidirectional traffic is lower than the bidirectional one. The difference between them can be quantified and, in mean, it is approximately

[2] This value is obtained by experimental results, although it can be computed as $0.5sec + 3R_{TT}$.

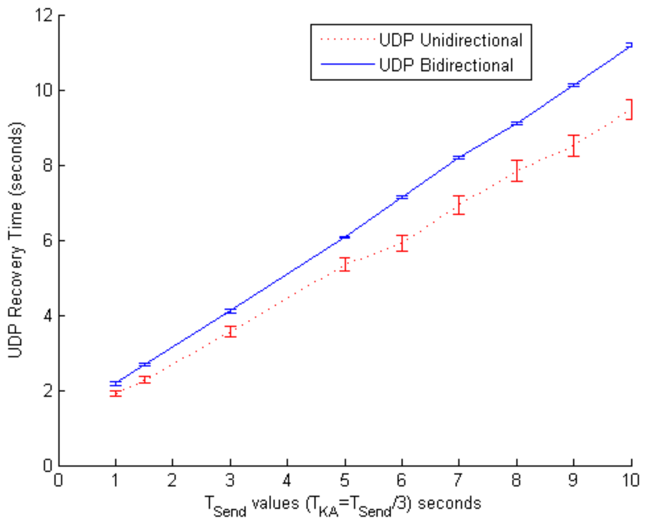


Fig. 2. UDP Recovery Time

equal to $\frac{T_{KA}}{2}$. This behavior is due to the fact that when there is only unidirectional traffic, Keep Alive messages are always exchanged in a regular basis. When a failure occurs, the last Keep Alive was probably received at the peer side some time before the failure. So the T_{Send} was started when the first packet after the reception of the Keep Alive is sent, and thus is, probably, some time before the failure. On the other hand, if we have continuous traffic in both ways, the T_{Send} timer is probably started closer to the time of the failure (the last time a packet was sent after the reception of a packet from the other side).

Keep Alive Signaling on the unidirectional case. The worst case scenario related with the overhead on signaling introduced by REAP is the unidirectional communication traffic case. If the traffic is unidirectional, Keep Alive messages are exchanged in a regular basis to maintain the relation updated. Once a packet is received, the T_{KA} timer is initiated, after this time period without sending any packet, a Keep Alive message is sent. The timer is not set again until a new packet arrives, hence the number of Keep Alive messages sent is dependant on the transmission rate of the source. If we call δ the time between two consecutive packets sent by the source, δ is an upper boundary to the time between sending a Keep Alive message and starting the Keep Alive timer again. Finally, the formula providing the number of Keep Alive messages sent per second is $\frac{1}{T_{KA}+\delta}$.

4.2 TCP Behavior

FTP Application. Figure 3 shows the Recovery Time achieved while varying the T_{Send} timer. Note that the results for TCP traffic are not linear with

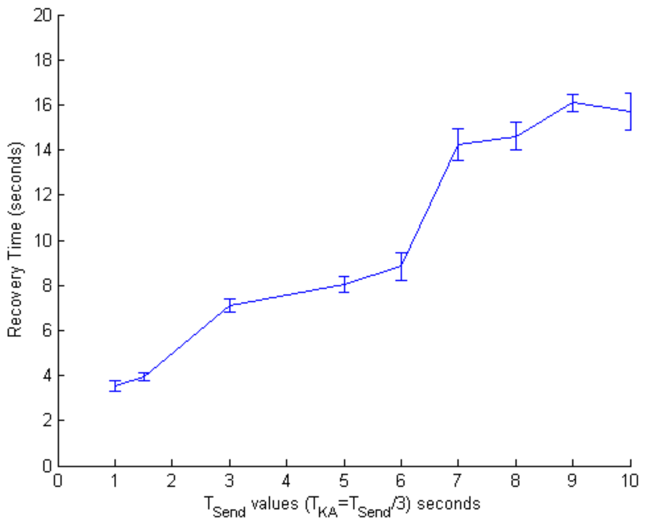
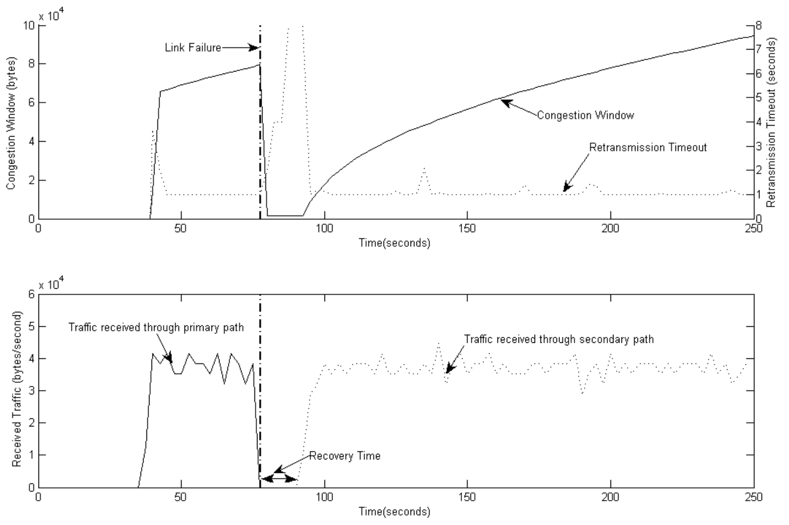
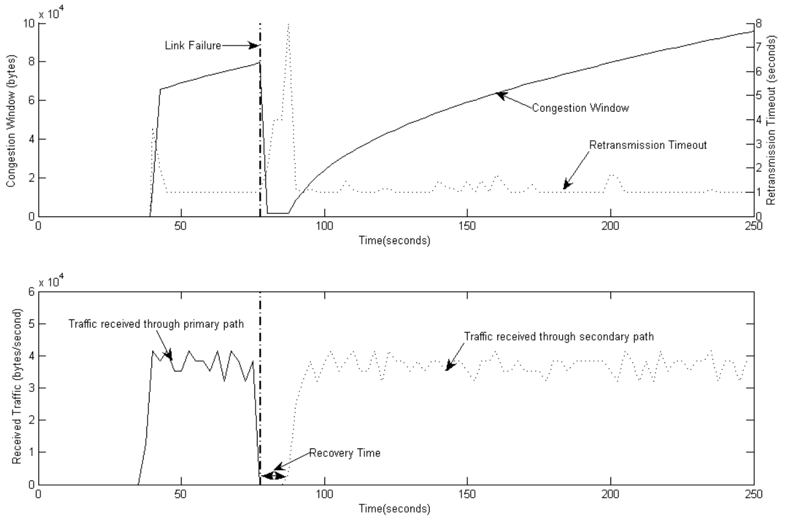


Fig. 3. TCP Recovery Time

the T_{Send} parameter as occurred with the UDP values (figure 2). This behavior is due to the mechanisms implemented in TCP for congestion detection and avoidance, in particular dependent on the retransmission timeout of TCP. TCP interprets a retransmission timeout as an indication of congestion in the network, so it uses an exponential back-off to retransmit the packets to avoid increasing the congestion. This mechanism affects the Recovery Time, since although the path has been reestablished, the packets will not be retransmitted until the retransmission timeout expires. To show a detailed explanation of this behavior we present figure 4(a). Figure 4(a) presents, for a given experiment ($T_{Send} = 10sec$), the Retransmission Timeout, Congestion Window and traffic sent through both paths. Traffic starts being sent through the primary path, until the link fails. At this moment the congestion window decreases and the retransmission timer increases. When the path exploration mechanism ends, the retransmission timer set up to 8 seconds has not expired. When it expires, packets are sent according to the slow start policy set for TCP. Figure 5 shows the difference in time between the arrival of a packet in a connection with a failure and the arrival of the same packet if no failure in the path occurs. As can be observed, packets suffer a big delay when the link fails (this delay is equivalent to the time needed to discover the failure and complete a path exploration mechanism), and then it remains roughly constant. This effect is due to the increase in the congestion window after the communication is recovered, packets will start to be sent faster until the congestion window reaches its top, after this packets are sent in a constant way, this behavior can be observed in figures 4(a) and 5. Due to the explanation presented above, we argue that the stair shaped graph in figure 3 is caused by the impact of the backoff mechanism of the retransmission timer of TCP. Figure 6, presents the backoff mechanism used by TCP to set up the



(a) Normal TCP operation



(b) TCP operation resetting the retransmission timeout

Fig. 4. TCP behavior explanation

retransmission timer. As the number of retransmissions increases, the retransmission timer duplicates its value. We argue that as the T_{Send} varies, the instant in time when the path is recovered falls in one of the steps presented in figure 6, this is the cause of the differences in time presented in figure 3.To improve

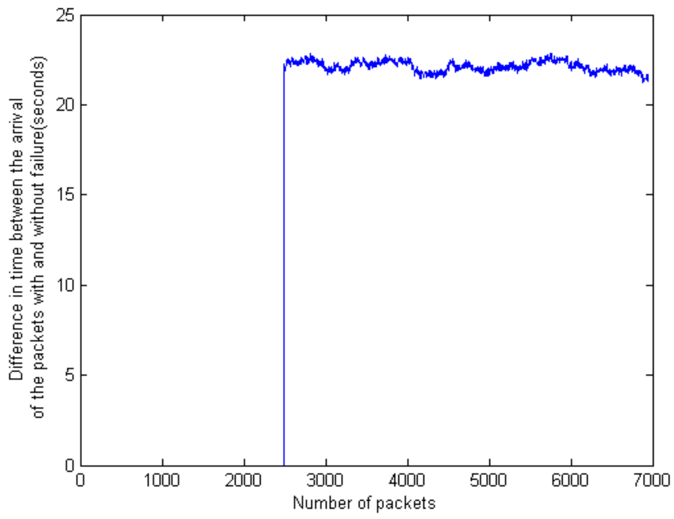


Fig. 5. Difference in time between packets in a communication with path failure and without path failure

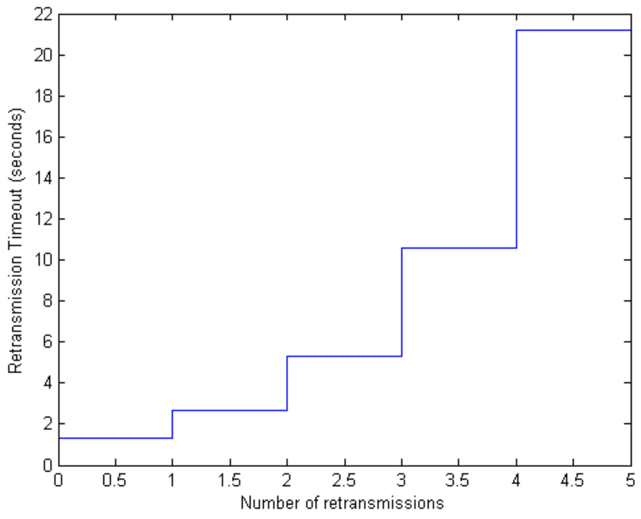


Fig. 6. TCP Retransmission Timeout

the performance, we propose to reset the retransmission timer of TCP after a new path is chosen for the communication (figure 4(b)). Notice that this is not only more efficient, but also appropriate as the retransmission timer value is dependant on the properties of the path. In the simulator, we implemented a

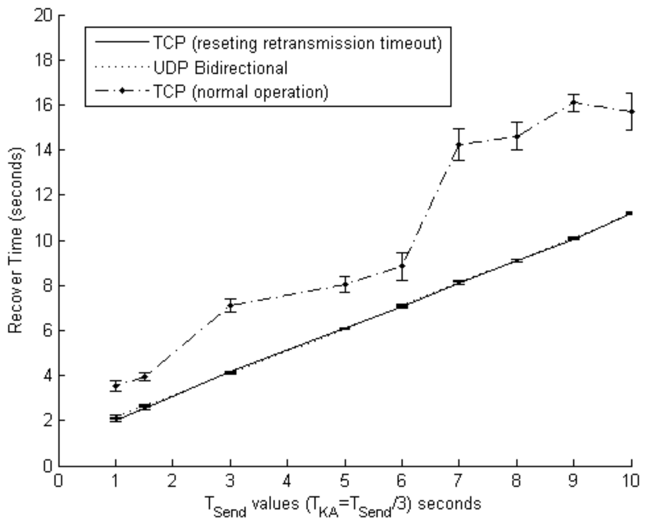


Fig. 7. UDP vs TCP (resetting the retransmission timer) Recovery Time

hook in the TCP stack to allow REAP to reset the retransmission timer forcing TCP to start retransmitting packets immediately. The experimental results of this proposal are presented in figure 7 along with the previous results for UDP bidirectional VoIP traffic and TCP traffic to easy the comparison. As expected, the relation between the TCP Recovery Time and T_{Send} is linear, and even more, the TCP modified behavior is very similar to UDP.

Telnet Application. To conclude the TCP study, an application with low data traffic has been analyzed. The chosen application is Telnet, in which usually long periods of time without packet transmission occurs. The design of REAP tries to minimize signalling overhead, so no Keep Alive probing is performed when no upper layer traffic is exchanged, due to this behavior, REAP will notice the failure after a time, defined by T_{Send}, of the first packet sent after the failure. To show this behavior figure 8 is presented. The Application Recovery Time metric, refers to the time elapsed between the first packet (using the old locators) sent after the failure and the first packet sent using the new locators. On this time period, the failure discovery and path exploration mechanisms are performed. The recovery procedure start time offset depends on the time between the path failure and the time when the application first tries to send a packet, but this offset is not important since the application is not affected by the failure because it was not trying to exchange any traffic. Figure 8 presents a similar trend to figure 3, presented on this figure for comparison purposes. The impact on the Application Recovery time of resetting the retransmission timer of TCP is shown in figure 8. As can be seen the effect is similar to the one presented on figure 7, being it a noticeable decrease on the Recovery Time of the application.

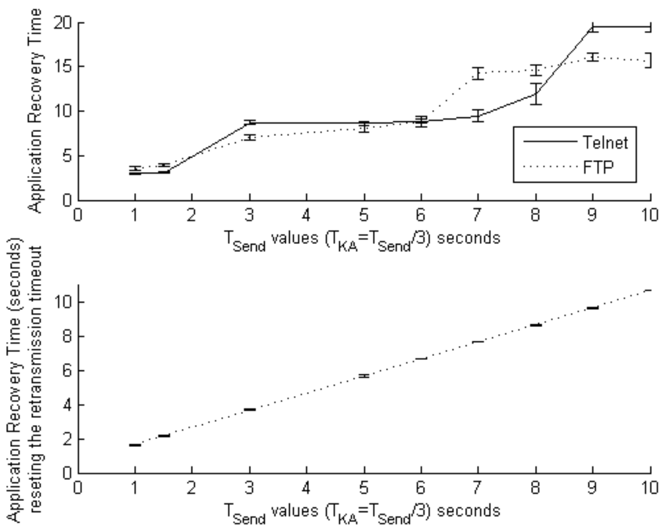


Fig. 8. Recovery time in a telnet application

TCP Reset Time. One of the most important characteristics of the REAP protocol to work in a TCP environment is to handle the recovery before the TCP session expires. In order to measure how much address pairs may be checked in the path exploration phase before TCP resets the connection, several tests have been done, using the default TCP configuration of a Microsoft Windows Server 2003 machine. The TCP stack implemented on it, resets the TCP connection if a limit of 5[3] retransmissions is reached. The time between the failure detection and the reset of the connection is of 75 seconds in mean.The REAP specification sets a backoff mechanism to handle the retransmission in the path exploration protocol. This mechanism follows $T_{out} = 2^n$ where n is the number of the retransmission count. This exponential backoff starts when the number of addresses proved is higher than 4, for the first 4 retransmissions a T_{out} of 0.5 seconds is used. The exponential backoff has a limit of 60 seconds, where it reaches the maximum, being the T_{out} for the rest of the retransmissions of 60 seconds.Taking into account the backoff mechanism, the number of possible IP address pairs explored is of 10. It is worth to notice that the first try of the REAP protocol is to check the current IP pair being used.As the previous results prove, the REAP protocol can check a big number of IP addresses before the TCP session expires, providing a mechanism to restore the communication.

5 Conclusion and Future Work

This paper presents details of the appropriate timer configuration of the REAP protocol as well as the effect of the protocol configuration in the two main

[3] http://technet2.microsoft.com/WindowsServer/

transport protocols, TCP and UDP. We also present a possible modification to the TCP stack which enables TCP to take advantage of the REAP protocol providing a faster Recovery Time. The results show a clear increase in the performance of the failure detection over TCP, being comparable to UDP.

As future work, there are several issues to be analyzed such as the aggressiveness of the path exploration mechanism (for example, explore alternative paths on parallel instead of serially) or a timer configuration based on the RTT which will decrease the Recovery Time while minimizing false positives on failure detection. We are also interested on increasing the information provided by the REAP protocol, going for path availability to gather some characteristics of the path (RTT could be the simplest example), and finally to study the combination of REAP with other protocols and scenarios, for example Mobile IP with multiple registrations and mobility.

References

[REAP] Arkko, J., van Beijnum, I.: Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming; IETF draft; draft-ietf-shim6-failure-detection-06 (September 2006)

[LOCSEL] Bagnulo, M.: Default Locator-pair selection algorithm for the SHIM6 protocol; IETF draft; draft-ietf-shim6-locator-pair-selection-01 (October 2006)

[HIP] Moskowitz, R., Nikander, P.: Host Identity Protocol (HIP) Architecture; Request for Comments: 4423

[MIP] Johnson, D., Perkins, C., Arkko, J.: Mobility Support in IPv6; Request for Comments: 3775

[SHIM] Nordmark, E., Bagnulo, M.: Level 3 multihoming shim protocol; IETF draft; draft-ietf-shim6-proto-06 (November 2006)

[SHIMAPP] Abley, J., Bagnulo, M.: Applicability Statement for the Level 3 Multihoming Shim Protocol(shim6); IETF draft; draft-ietf-shim6-applicability-02 (October 2006)

[BGPMULT] Van Beijnum, I.: BGP: Builiding Reliable Networks with the Border Gateway Protocol; Ed O'Reilly (2002)

[MONAMI] Wakikawa, R., Ernst, T., Nagami, K.: Multiple Care-of Addresses Registration; IETF draft; draft-ietf-monami6-multiplecoa-01 (October 2006)