STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

NETWORK INTERFACE AGGREGATION FOR IP TUNNELING

ADRIAN SEREDINSCHI(1) AND ADRIAN STERCA(1)

Abstract. We present in this paper a network interface aggregation framework for IP tunneling. Our framework periodically measures the level of congestion on routes going through each network interface and based on these measurements, it decides how many flows to send on an outgoing network interface at a time. In this way, we can achieve better inter-flow fairness regarding bandwidth utilization.

1. Introduction. IP Multihoming and IP Tunnels

IP multihoming means any form of providing reliable network connectivity to a service point in the network. It implies two or more redundant network connections from this node to the core network. IP multihoming can be realized in a static setup (i.e. when specific flows are statically mapped/routed to a specific uplink interface) or in a more dynamic setup using BGP.

IP tunnels are used to transparently connect two distant networks that normally do not have a native routing path between them. There are several types of IP tunnels, from simple ones like IP-in-IP encapsulation to GRE and to IPSec tunnels which create a virtual private network composed of two remote sites.

In this work we would like to combine multihoming routing algorithms with IP tunnels in order to provide reliable, load-balanced virtual private network links between two remote sites. More specific, the network architecture we consider is depicted in Figure 1.

Received by the editors: April 30, 2013.

2010 *Mathematics Subject Classification.* 68M20, 68U99.

1998 *CR Categories and Descriptors.* C.2.2 [**Computer-Communication Networks**]: Subtopic – *Network Protocols*; C.2.3 [**Computer-Communication Networks**]: Subtopic – *Network Operations*.

Key words and phrases. network interface aggregation, IP tunneling, IP multihoming.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

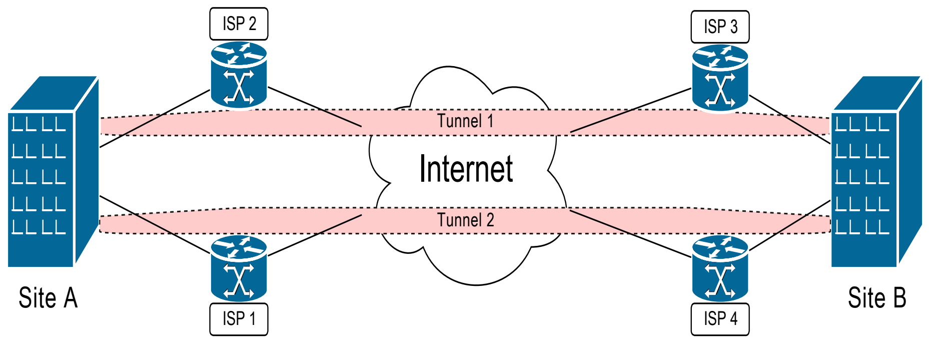

FIGURE 1. The network architecture

We consider a network setup where two remote local area networks denoted in the figure by *Site A* and *Site B* have each several uplink connections to the Internet (generally, we consider a small number of network links, two or three, per network site) and we want to connect them in a virtual private network. For this, we need an IP tunnel (i.e. IPSec or something else) between these two sites, but we also want to spread the outgoing tunnel traffic on all uplink connections of a site, so that the network bandwidth of a site is optimally used. In Figure 1, *Site A* has two uplink connections, one through ISP1 and the other through ISP2 and *Site B* has also two uplink connections, one through ISP3 and the other through ISP4. There are four possible network paths between *Site A* and *Site B*: a path going through ISP2 and ISP3, another path going through ISP1 and ISP4, another path going through ISP2 and ISP4 and, finally, the last path going through ISP1 and ISP3. We depicted in Figure 1 and consider in our framework only the first two paths because we want the network paths to be as independent as possible (i.e. to have a number of common links close to zero), at least theoretically. Of course, in reality there is no guarantee that the network route going through ISP2 and ISP3 and the network route going through ISP1 and ISP4 do not share any intermediate network link. When the network paths between *Site A* and *Site B* share a large number of network links our algorithm would be less effective, but this situation will be discussed further in section 3. Our algorithm routes packets sent by the local area network through the tunnel logical interface on one of the existing paths from the source site to the destination site based on the level of congestion experienced by each network path. The goal is to achieve a better utilization of the network bandwidth (between the source site and the destination site) and to achieve a higher degree of inter-flow fairness between the flows originating from the same network site and sharing the tunnel.

The rest of the paper is organized as follows. In the next section related work is reviewed. Then, section 3 describes the main algorithms of our framework of network interface aggregation for IP tunnels followed by section 4 which shows some experiments performed using our framework which will validate its usefulness. The paper ends with conclusions in section 5.

2. Related work

Dynamic IP multihoming is normally realized using BGP [1]. But since BGP is a general distance-vector routing protocol and BGP routing tables are normally quite large, BGP makes its route selection decision based on local preferences or weights and hop-count costs and it does not use real-time link state information like the congestion level of a whole route in this selection process; this is because it is very costly to probe every possible destination network it knows and it does not have control over the whole network path.

Policy-based routing is another technique used for IP multihoming. In policy-based routing, the next hop is determined based on the source IP address of the packet and routing policies are set by a human operator. A more general form of policy-based routing is source based routing in which the network route on which a packet will travel is specified partially or totally in the packet header. Source based routing is especially used in the context of wireless ad hoc networks, but it requires modifications in the TCP/IP stack currently deployed in Internet routers [2], [3].

Another routing technique related to our framework is multipath routing used also in the context of wireless ad hoc networks [4]. In multipath routing a flow's packets are sent to the destination over multiple routes, possibly overlapping, in order to improve the efficiency of the transport. An important problem with multipath routing is that sending a flow over different network routes can cause many out of order packets at the receiver which can degrade the throughput of the upper level protocol, i.e. TCP.

All the above techniques function at level 3 in the TCP/IP stack and they are all meant to increase the throughput efficiency of the data at the router. Similar approaches exist also at level 2, namely Ethernet link aggregation/bounding (LACP protocol in IEEE 802.1ax or open-source Linux bounding driver).

The same idea of using different network paths between the source network and the destination network depending on the level of congestion on each path is explored at a higher level, level 4 (i.e. the transport level), in Multipath-TCP [5] where a TCP flow is split over several routes depending on the level of congestion on each route. In Multipath-TCP selecting the current route for a packet is done in an end-to-end fashion at the source end node, while in our case is done inside the network by the source router which does not split a single TCP flow on different routes, but considers all the flows originating from the source network/site. Also, in Multipath-TCP an end host relies on some support from the network (i.e. from network routers) in order to route the packet on a specific route established by the source end node.

Probably, the work closest to ours is presented in [6]. In [6] authors present a multi-homing solution using IP tunnels in order to achieve higher throughput for TCP flows originating on a specific site and having a specific destination site. TCP flows are split over several network interfaces at IP level so that the resulted total bandwidth is the bandwidth aggregated over these network interfaces. Our work is different than the one presented in [6], because we estimate differently the congestion (i.e. available bandwidth) on each network path and our paper considers all flows originating in the source sites while the author of [6] only deal with splitting a TCP flow across the outgoing network links.

3. Network Interface Aggregation for IP Tunneling

The goal of our network interface aggregation framework for IP tunneling is to split outgoing traffic on the IP tunnel so that a link that is more congested than other should receive less traffic in order to maintaining a high degree of inter-flow fairness among all flows originating at the same source. In order to achieve this, our framework periodically measures the current level of congestion of each outgoing link and sends a corresponding fraction of the total traffic on this link.

The architecture of our framework is depicted in Figure 2. The framework runs at both sides of the tunnel and at each side it is composed symmetrically from two modules:

- IP mangler module
- statistics gatherer module

The IP mangler module is responsible with choosing an outgoing link for the current packet. It chooses an outgoing link for a packet by setting (mangling) the source IP address and the destination IP address in the outer IP header of the packet. Using the statistics gatherer module, our framework computes a congestion metric for each (considered) route between the source site and the destination site. If we consider the routes from Figure 1 and we consider that the route going through ISP2 and ISP3 has the congestion metric $cong\_metric\_0$ and the route going through ISP1 and ISP4 has the congestion metric $cong\_metric\_1$ then $traffic\_weight\_0$ of the total traffic will be sent on the route ISP2-ISP3 and $traffic\_weight\_1$ of the total traffic will be sent on the route ISP1-ISP4 where

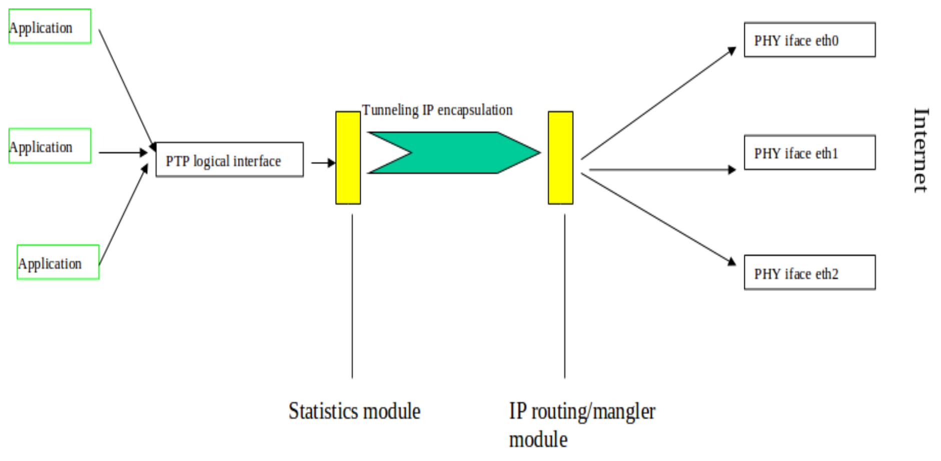

FIGURE 2. Our IP Tunneling Aggregation Framework

$$traffic\_weight\_0 = 1 - \frac{cong\_metric\_0}{cong\_metric\_0 + cong\_metric\_1}$$

$$traffic\_weight\_1 = 1 - \frac{cong\_metric\_1}{cong\_metric\_0 + cong\_metric\_1}$$

and $traffic\_weight\_0, traffic\_weight\_1 \in [0, 1]$ and $traffic\_weight\_0 + traffic\_weight\_1 = 1$.

Considering that there are N flows going out through the tunnel and all flows are non-limited TCP flows (i.e. they use as much bandwidth as it is available), because we want to send, as much as possible, all the packets of a flow through the same outgoing link (to avoid packet reordering), we approximate $traffic\_weight\_0$ of the total traffic with $traffic\_weight\_0 \cdot N$ flows and $traffic\_weight\_1$ of the total traffic with $traffic\_weight\_1 \cdot N$ flows. In order to ensure a correct functioning of the statistics gatherer module, we enforce that there exists at least one flow on each outgoing link (if the total number of flows is greater than 1).

The statistics gatherer module monitors the congestion level on the outgoing links of the site. This module uses passive measurements on the incoming traffic in order to determine the level of congestion. More specifically it probes incoming acknowledgment packets on each network interface for the timestamp value in the TCP header (i.e. the TSecr field) and it measures the RTT on that interface. Computing the current congestion level on a link is done in a way similar to how TCP Vegas does [7]. For each link a smoothed weighted average value of the round-trip time is kept in the state variable *srtt*:

$$srtt = srtt \cdot 0.975 + rtt\_curr \cdot 0.025$$

where *rtt_curr* is an approximation of the current RTT obtained by subtracting the TSecr value of a packet from the current time. The congestion level on a link is computed like this:

$$cong\_metric = cong\_metric \cdot 0.9 + (srtt - min\_srtt)^{+} \cdot 0.1$$

where *min_srtt* is the minimum *srtt* value measured so far on this interface.

The statistics gatherer module relies on the fact that all the packets of a flow (i.e. data packets and acknowledgment packets) follow (approximately) the same outgoing route, so that the RTT measurement per link is consistent. For this reason, our framework maintains a list of connections/flows for each outgoing network interface. An entry in this connections list (i.e. a flow) contains the following data:

- source IP address
- destination IP address
- source port
- destination port
- switch flag

where the first four items are taken from the inner packet, after the outer IP and Ethernet headers are removed and the switch flag is set to 'true' if this flow is sent through a different network route by the remote site (e.g. the flag is set to 'true' if site A sends a flow/connection through the route ISP2 - ISP3 and site B sends the same flow/connection (i.e. the ACK packets) back through the ISP4 - ISP1 route in Figure 1). The switch flag is used by our framework when a new decision to reallocate flows on network interfaces is taken so that if some flows need to be moved from one interface to the other, the flows having the switch flag set to true are chosen first.

From time to time, depending on the traffic weights set by the statistics gatherer module and if the congestion level changed on at least one of the routes, the IP mangler module reallocates flows on the network interfaces (i.e. moves flows from the connections list of an interface to the connections list of another interface and mangles the IP addresses of every packet belonging to that flow accordingly). The IP mangler module reallocates flows on interfaces if the following conditions are met for at least one outgoing network interface:

$$|srtt - min\_srtt| > 0.5 \cdot min\_srtt$$

$$|srtt\_last\_reallocation - srtt| > min\_srtt$$

where *min_srtt* is the minimum SRTT value measured on that network interface, ever and *srtt_last_reallocation* is the *srtt* measured on this interface when the last reallocation took place. When both conditions from above are met for an interface, approximately $traffic\_weight\_0 - traffic\_weight\_1$ flows are moved from interface 1 to interface 0 if $traffic\_weight\_0 > traffic\_weight\_1$ and $traffic\_weight\_1 - traffic\_weight\_0$ flows are moved from interface 0 to interface 1 if $traffic\_weight\_0 < traffic\_weight\_1$.

A final observation about our framework's functioning is needed and that is the initialization of the framework's state. When started, our framework does not have initial values for the $srtt$, $cong\_metric$ and $traffic\_weight$ variables for any interface, so the IP mangler module distributes flows equally among existing network interfaces. Once there is at least 1 flow on each interface, the IP aggregation framework can compute these values and it can distribute flows on interfaces according to the level of congestion on each route/interface. The $srtt$ value is computed only from packets belonging to flows that follow the same network path in both directions (i.e. flows that have the switch flag set to 'false') - this is required for the measurement to be relevant for that network path.

4. Experiments

In order to evaluate our network interface aggregation framework, we have set up a test scenario as depicted in Figure 3. We have two machines, Site A and Site B, connected to the network by 2 network interfaces each and a router with 4 network interfaces which connects Site A and Site B. An IP-in-IP tunnel is established between Site A and Site B. We started 32 TCP senders on Site A and 32 TCP receivers on Site B and on each flow we sent random data as fast as the network allows from site A to site B. We used a linux traffic shaper on the network interfaces eth2 and eth3 of the Router to limit the bandwidth on each route (i.e. simulate congestion). The traffic shaper alternates the bandwidth of network interfaces eth2 and eth3 of the Router between 1500 Kbits/sec and 3000 Kbits/sec at 30 seconds time intervals. The bandwidth pattern is the following: in the first 30 seconds of a cycle both interfaces are limited to 3000 Kbits/sec each, in the next 30 seconds eth0 is limited to 1500 Kbits/sec while eth1 is still at 3000 Kbits/sec, in the following 30 seconds both interface have again 3000 Kbits/sec, and in the last 30-second period of a cycle interface eth1 is limited to 1500 Kbits/sec and interface eth0 keeps sending at 3000 Kbits/sec. This cycle of 4 states repeates indefinitely.

We have run 2 experiments on this network setup, each lasting for 300 seconds. In the first experiment we have used a static algorithm for splitting the 32 TCP flows at site A on 2 network interfaces; this static algorithm sends 16 flows on one interface/route and the remaining 16 flows on another interface/route and it never realocates the flows on a different interface. In the second experiment we have used our framework at site A to perform dynamic reallocation of flows on outgoing interfaces depending on the level of congestion on each route. We were only interested in the traffic flowing from site A to site B, in this direction only. We measured at site A the flows per interface allocation, the average RTT (i.e. *srtt*) measured on each interface in both experiments and the average bandwidth per second received by a flow on each interface in both experiments. Our findings are shown in Figures 4, 5, 6 and 7.

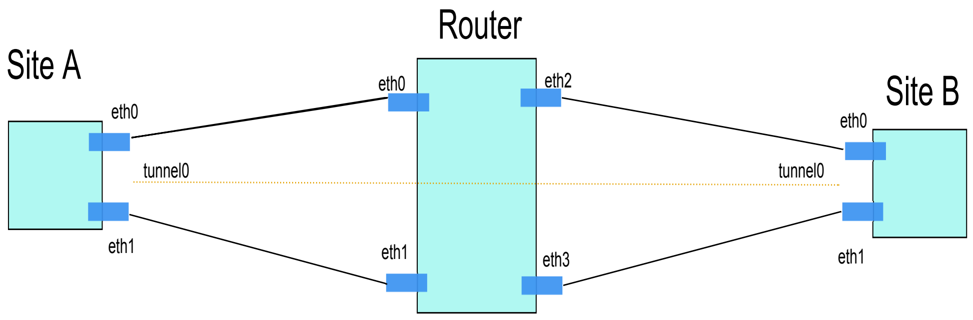

FIGURE 3. Network setup used for evaluation

The *rtt* and *srtt* values measured on interface eth0 of site A in the dynamic reallocation experiment is pictured in Figure 4. The values of *rtt* and *srtt* measured on interface eth0 of A in the static allocation experiment is shown in Figure 5. In both experiments, similar values of *rtt* and *srtt* were obtained on interface eth1 of A, so these plots are not included here. The *srtt* values are measured in milliseconds. It can be seen in figures 4 and 5 that out framework does not change significantly the RTT and SRTT measurement on a network interface.

Figure 6 depicts the allocation of flows per interface performed by our network interface aggregation framework at site A.

For each of the two experiments, we have computed the average bandwidth obtained by a flow on each of the two network interfaces (i.e. interfaces eth0 and eth1 of site A). Then we computed a bandwidth-allocation-per-flow fairness index which is equal to $\frac{min\_bw_i}{max\_bw_i}$ where min_bw_i is the minimum of the two bandwidth-per-flow values on each network interface in second i and max_bw_i is the maximum of the two bandwidth-per-flow values (on each network interface) in the same second i.

The bandwidth allocation per flow fairness index is depicted as a function of time in Figure 7 for each of the two experiments: the experiment when our

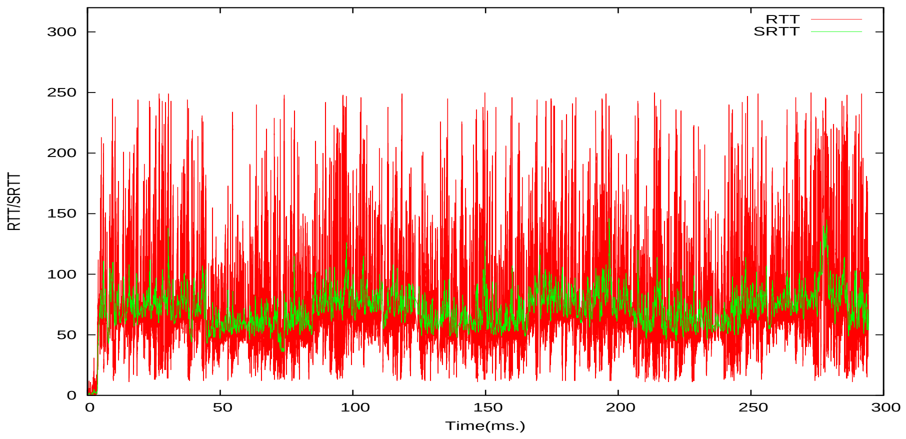

FIGURE 4. The *rtt* and *srtt* measured on interface eth0 of A when our framework was used

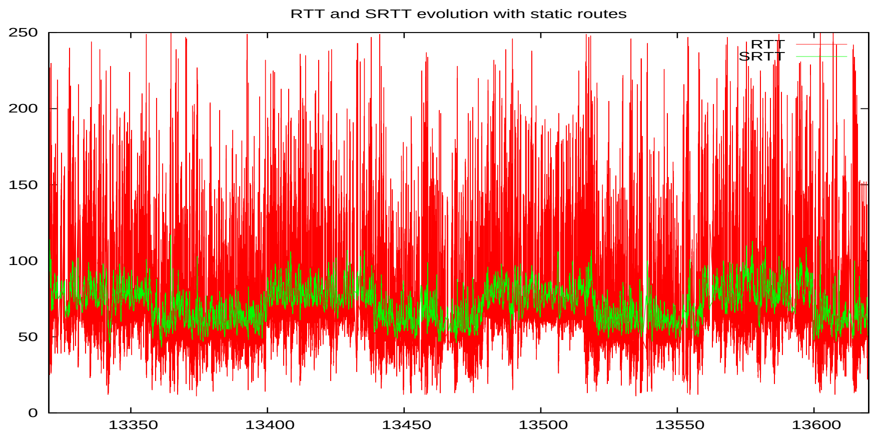

FIGURE 5. The *rtt* and *srtt* measured on interface eth0 of A when static allocation of flows was used

framework was used and the experiment when static allocation of flows was employed (i.e. 16 flows on interface eth0 and 16 flows on interface eth1).

We can see that, in general, our framework attains a higher Bandwidth-per-Flow fairness index and on average, across all 300 seconds of the experiments, our framework obtains a Bandwidth-per-Flow fairness index of 0.843688 while the static allocation scenario obtains a values of 0.741942, so an improvement of slightly over 10%.

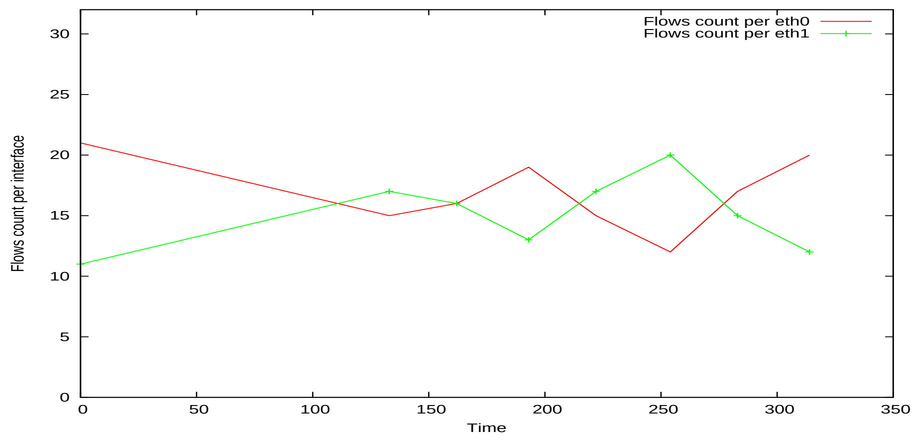

FIGURE 6. Distribution of flows on interfaces eth0 and eth1 of A when our framework was used

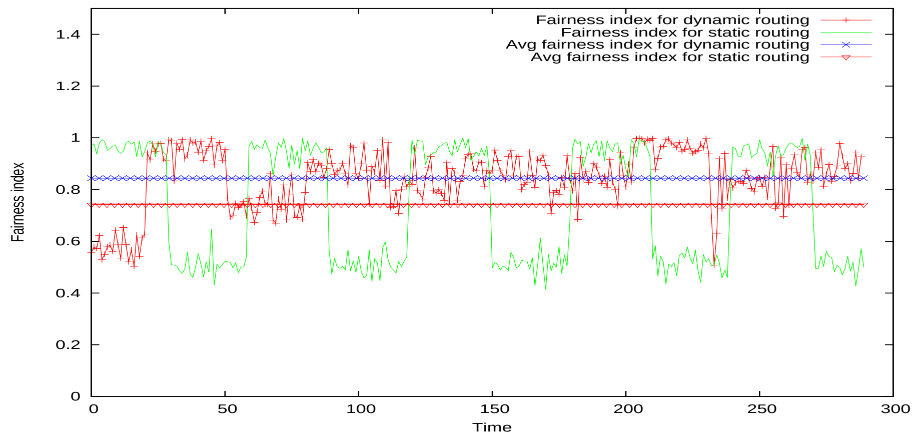

FIGURE 7. Bandwidth-per-Flow fairness index obtained by our framework vs. static allocation scenario

5. Conclusion

We have presented in this paper a framework for network interface aggregation for IP tunneling. Our framework periodically measures the level of congestion on routes going through each network interface and based on these measurements, it decides how many flows to send on an outgoing network interface at a time. The utility of our framework is validated in section 4.

References

[1] Y. Rekhter, T. Li, S. Hares, *A Border Gateway Protocol 4 (BGP-4)*, RFC 4271, January 2006.

[2] Andrews, M., Fernandez, A., Goel, A., Zhang, L., *Source routing and scheduling in packet networks* Journal of the ACM (JACM), 52, pp. 582-601, 2005.

[3] Johnson, D. B., Maltz, D. A., Broch, J., *DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks*, in Ad hoc networking, 5, pp. 139-172, 2001.

[4] Nasipuri, A., Castaneda, R., Das, S. R., *Performance of multipath routing for on-demand protocols in mobile ad hoc networks*, in Mobile Networks and applications, 6(4), pp. 339-349, 2001.

[5] Han, H., Shakkottai, S., Hollot, C. V., Srikant, R. and Towsley D., *Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the internet* in IEEE/ACM Transactions on Networking 14, 6, pp. 1260-1271, 2006.

[6] Phatak, D. S., Goff, T., Plusquellic, J., *IP-in-IP tunneling to enable the simultaneous use of multiple IP interfaces for network level connection striping*, in Computer Networks, 43(6), pp. 787-804, 2003.

[7] Lawrence S. Brakmo , Sean W. Omalley , Larry L. Peterson, *TCP Vegas: New techniques for congestion detection and avoidance*, in Proc. of ACM SIGCOMM Conference, 1994.

(1) Babeş-Bolyai University, Department of Computer Science, Cluj-Napoca, Romania

E-mail address: sdsd0494@scs.ubbcluj.ro

E-mail address: forest@cs.ubbcluj.ro