Fast Path Ethernet Switching: On-demand, Efficient Transparent Bridges for Data Center and Campus Networks

Guillermo Ibáñez, *Member, IEEE*, Juan A. Carral, Alberto García-Martínez, José M. Arco, Diego Rivera, Arturo Azcorra, *Member* IEEE

Abstract— In this paper we propose Fast Path Ethernet, an evolution of the transparent bridges learning mechanisms to increase infrastructure utilization for campus and datacenter networks in a simple way. Fast Path Ethernet Switches reuse standard ARP Request and Reply packets to set up fast on-demand paths between hosts. This architecture uses the standard Ethernet frame format, so it is fully transparent to hosts and compatible with 802.1D bridging in core-island mode. A proof of concept has been implemented in Linux. Preliminary simulations in metropolitan and campus network topologies show superior performance to spanning tree and even to shortest path forwarding, at a fraction of the their complexity.

***Index Terms*—Ethernet, Routing bridges, Spanning Tree**

I. INTRODUCTION

Ethernet switched networks offer important advantages in terms of price/performance ratio, compatibility and auto configuration. The use of the Spanning Tree Protocol (STP) [1] enables loop-free operation of bridged networks without the need to configure complex routing information, and without costly and error prone administration of IP addresses and segments. However STP severely limits the performance of Ethernet networks because it blocks all links exceeding the number of network bridges minus one.

Current proposals under standardization like Shortest Path Bridges (SPB) [2] and Routing Bridges [3] rely on a link-state routing protocol which operates at layer two to obtain shortest path routes between bridges. However, link state protocols have significant complexity both in terms of computation and control message exchange.

In this paper we propose Fast Path Ethernet Switching (or Fast Path, for short), a zero-configuration protocol for data center and campus networks to enable the use of the whole available toplogy. Fast path directly evolves from the transparent learning bridge paradigm

Fast paths are set up as a result of the controlled flooding of an ARP Request, which is encapsulated into a broadcast frame. The mechanism assures that the ARP Reply frame follows the same path back to the source, so this frame is used to confirm the set up of the path. In order to assure that the fastest path is chosen, the first arriving copy of the ARP Request locks at every bridge the corresponding port for the path. Further (late) copies arriving to other ports of the bridge are discarded.

Although flooding of packets to find the shortest path is not a new concept, it has never been applied to transparent learning bridges, mainly due to the problem of broadcast frame loops . The use of flooding to obtain shortest routes has been proposed at layer three and wireless applications, but not for standard bridged Ethernet networks, where either the spanning tree protocol is used to prevent loops, loop free topologies are used, or routing in layer two is performed and specific loop prevention mechanisms are in use [2].

The Fast Path protocol uses the standard Ethernet frame format, it is fully transparent to hosts and routers and may coexist with standard bridges in core-island mode. As most of high performance IEEE 802.1 protocols, Fast path bridging requires point to point links between bridges. The performance of Fast Path both in terms of infrastructure utilization and path length is similar to shortest path routing protocols, but with lower complexity.

II. FAST PATH PROTOCOL

There are three basic differences between Fast Path bridges and standard spanning tree transparent bridges: first, Fast Path increases the number of usable links, limited to a tree for transparent bridges; second, it modifies the address learning mechanism at ports, introducing the concept of *locking* of the address learned to the first port receiving the frame; and third, it prevents the replication of frames with an unknown unicast destination address by means of a mechanism which rebuilds a damaged path.

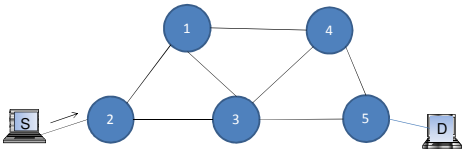
A. Fastpath set up

1) Path discovery (ARP Request)

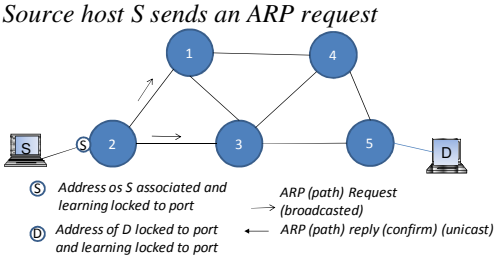
The mechanism to set up *fast paths* is partially inspired by the Reverse Path Forwarding [4]. A *fast path* is the fastest (and so, unique) path created by the first copy of an ARP Request frame reaching its destination host. The process, described in Fig.1, works as follows:

Host S sends an ARP Request encapsulated into a broadcast frame B to resolve the IP address of a given destination host D. The ingress bridge 2 receives the frame from S and associates the global MAC address of S to the port through which it has received the message, temporarily locking the learning of S address to this port and blocking all other ports of bridge 2 from learning and forwarding further received broadcast frames from source address S. Thus, frames with source address S, arriving to other ports of bridge 2, are discarded as *late* frames. Then, bridge 2 forwards B to all ports except the one through which it was received. Bridges 3 and 1 behave as bridge 2, locking

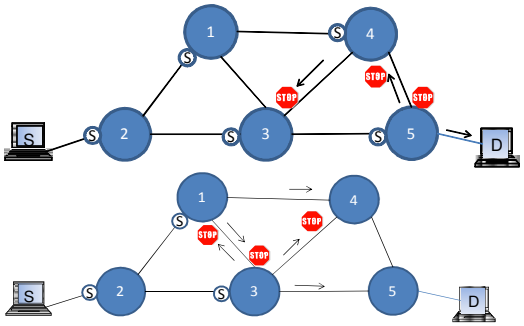
address S to the port that first receives the frame. Afterwards, bridges 3 and 1 broadcast the frame through all other ports except the port where it was first received, so that duplicate copies of B arrive to 3 and 1, sent by each other. However these frames arrive at a port different from the port temporarily locked to S, so they are discarded.



a) Source host S sends an ARP request



b) Bridge 2 associates S address to the input port, blocks learning of S address in other ports and forwards the frame through all other ports



c) Bridges 1 and 3 also associate S address to its input port, forward the frame through all other ports and block learning of S address in other ports

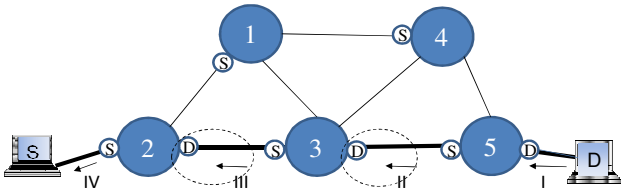
d) The same happens in Bridges 4 and 5. Finally, the ARP Request arrives at destination host D

Figure 1. Path discovery from source host S to destination host D

The same happens at bridges 4 and 5. Hence, the temporary association (*locking*) of address S to a port at every bridge is propagated across the network as a tree rooted at host S, until the network edge bridges and their hosts are reached, including the host D, destination of the ARP Request. A chain of bridges with an input port locked to S is now active between S and D.

2) Path confirmation (ARP Reply)

The mechanism for path confirmation in the opposite direction is described in Fig. 2.



I. Host D sends a unicast ARP Reply towards host S

II. Bridge 5 associates address D to the input port and blocks learning of D address in other ports. Then it forwards the frame via the port associated to S. Bridge 5 has now confirmed routes to S and D. Cache timers are activated for both S and D

III. Bridge 3 associates address D to the input port and blocks learning of D in other ports. Then it forwards the frame through the port associated to S, confirming the path and activating cache timers for S and D

IV. Bridge 2 associates D address to the input port and blocks learning of D address in other ports. Then it forwards the frame through the port associated to S, confirming the path and activating cache timers for S and D

Figure 2. The ARP Reply packet confirms at every traversed bridge the existing association (temporary lock) of S address to a port. It also sets up a confirmed association of D address to its input port.

The confirmation mechanism ensures that the path is symmetric (i.e. it coincides in both directions S-D and D-S). The path in S->D direction (address S learnt at ports of bridges) is confirmed backwards by the unicast reply from the destination over the same path D->S. This is required to keep the backward learning mechanism safe and to prevent oscillations in port to address associations. Specific priority mechanisms are used to prevent the setting of parallel paths by two simultaneous ARPs sent in opposite directions.

3) Path restoration

Established Fast Paths (i.e. chains of learnt addresses at bridge ports) may get broken at some point either by the expiration of an address timer or by a link failure.

The failure of a link connecting two Fast Path bridges provokes the flushing of all MAC addresses associated to the two ports of that link. The same happens at all ports of a node, when it is the node which it reinitializes.

Whenever a bridge receives a frame with an unknown destination address (i.e. the address is not associated to any port), the path may be rebuilt from the source bridge or from the bridge detecting the missing the path.

If the path is rebuilt from the source bridge (i.e. the bridge closest to the source host), the bridge that received the unknown destination unicast frame encapsulates it inside a *Path_Fail* message and returns it in the backward direction towards the source host. This message is processed at each bridge in the backward path, which forwards it via the port associated to the source host till it reaches the source edge bridge. The *Path_Fail* message is addressed to the *All_Fastpath_Bridges* MAC multicast group and delivers the the unicast frame rejected as payload. Frames sent to the multicast group are (only) processed by Fast Path Bridges. Every bridge in the path checks if it is the source

edge bridge of the source host of the rejected unicast frame (i.e. if the host is directly connected to it). In this case the bridge broadcasts a new ARP Request on behalf of the source host of the unicast frame and the path is recreated in the normal way.

Alternatively, the path may be rebuilt directly from the affected bridge onwards by issuing, either a standard ARP Request on behalf of the source host, or a *Path_Request* message addressed to the *All_Fastpath_Bridges* multicast address. In the former case the ARP Request is replied by the destination host with an ARP Reply that selects the path towards the failed bridge, which intercepts the ARP Reply. In the latter case, a *Path_Request* message containing the source and destination MACs and IP addresses is broadcasted in the forward direction and processed and forwarded by all the bridges traversed till the bridge attached to the destination host. The pseudo code in Fig. 3 summarizes the frame processing in a Fast Path bridge.

If the path is rebuilt from an affected bridge onwards and the next nodes in the forward path have a port already locked to the source address, the request will be rejected at all ports not locked but will be accepted, even if received later, at the port that was already locked. If a link corresponding to a previously locked port is broken the link failure is detected at port and locking to all MAC addresses cancelled.

```
Frame processing at bridge
-Destination address is broadcast or multicast
   - Destination address is multicast Fast Path: process as control
frame:
    -  Is Path Fail message: resend ARP Request with frame data)
   - Else if:
        - source address is unknown
           -Lock  temporarily source address to input port
        - source address is known (a Fast Path exists)
           -Discard frame if input port is not the associated port
 -Forward frame through all ports except prohibited turns and refresh
persistency timer of source address
-Destination address is unicast
    - Destination address is known
       -Frame is ARP Reply to a pending ARP Request:
          - Confirm locked address (frame destination address  to
            output port). Activate persistence timer.
          -  Associate source address of unicast frame to input
             port. Start refresh timer.
       -  Else if Source address is known
              -Forward to associated output port. Refresh timers at
              ports for source and destination  addresses
       - Else:  associate source address of unicast frame to input
          port. Start refresh timer
   - Destination address  is unknown
          -Send Path Fail  in backward direction, encapsulating packet
header in multicast frame
```

Figure 3. Fast Path protocol frame processing pseudo code

B. Address- to- port association state machine

Fast Path switches replace the standard learning mechanism by a new mechanism for associating learned addresses to bridge ports. Fig. 4 shows the Finite State Machine model. The ellipses show address states and the transitions show in the upper line the event (type of packet received in italics) and in the lower line (in bold) the action performed. A MAC is in *released* state when it is not associated to any port of the bridge (i.e. it is unknown to the bridge). When an ARP Request (or a *Path Request*) packet, sourced at MAC address A, is received at a Fast Path Bridge via its port *x*, the MAC address A changes its state to *locked* for the duration of the bridge's *lock_To* timer or until the corresponding ARP Reply (or *Path_Reply*) is received, whatever happens first. If the Reply packet is received before *lock_To* expiration, the state changes to *learned/confirmed*, so the association of A address to port *x* is now firm, else the association is discarded and the state falls back to *released*. The *learned/confirmed* state is maintained at least for a *learned/confirmed_To* period. This timer is similar to bridge cache expiration timers (300 s.) and gets refreshed upon reception of new unicast frames originated at A address received via port *x*. A B address in the *released* state (unknown to the bridge) may directly transit to *learned/confirmed* state when a reply packet, sent from B to A, is received (via port *y*) and triggers the transit of A address state from *locked* to *learned/confirmed*. B address gets associated to port *y*.

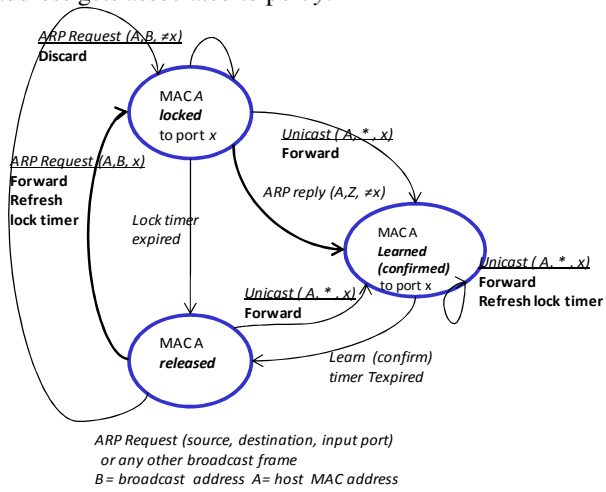


Figure 4. Basic state machine of address-to-port association (locking and confirmation)

C. Reconfiguration

We now describe how Fast Path handles network reconfiguration after a link or a bridge failure. The failure of a bridge could be detected by the physical layer at the input ports of neighboring nodes as a link failure. Optionally, Continuity Check Messages could be used to detect node or link failures as defined by specifications Y.1731 [5] and 802.1ag [6].

When the failure of a link connecting two bridges is detected at the respective ports, it provokes the flushing of all MAC addresses associated to these ports. The same happens, at all ports of a node, when a node reinitializes. The paths are no longer valid and will be rebuilt, but only when needed, by the path restoration procedure, as described above.

The dynamics of learned MAC addresses in Fast Path bridges after reconfiguration is similar to standard bridges, although in standard bridges the topology is restricted by the spanning tree protocol. When a port is no longer part of the active topology due to failure or after being disable by the operator, hosts are no longer reachable through that port, so their MAC addresses are removed from the cache.

When a link deactivates, the addresses associated to the corresponding port are flushed. Contrary to standard bridges using the spanning tree protocol, Fast Path bridges do not need to propagate Topology Change Notification Messages to the whole network. Paths are rebuilt (addresses are relearned) only when needed (after a path failure reported from a reconfigured bridge), as described above.

D. Load distribution

Fast Path bridges set up paths on an on-demand basis; this means that automatic adaptation of paths to load conditions comes for free. By design, when a new path is requested, the fastest path reaching to the destination host will be selected. This means that new paths selected from hosts (distinct from existing active communicating hosts) will follow the fastest path at the moment of request, thus the paths with higher load (delay) will not be selected. Switches with shorter latency will tend to be selected. Due to the high number of hosts, traffic will be balanced over the whole infrastructure, thus offering shortest latencies.

E. Compatibility with standard bridges and routers

Fast Path switches may cooperate connected to standard bridges in core-island mode, as shown in Fig. 5. A core of Fast Path bridges may interconnect islands of standard bridges running the spanning tree protocol. Self configuration of islands of standard bridges operates as follows: Fast Path bridges connected to standard bridges receive standard BPDUs on the ports connecting to the standard bridge islands. As a consequence they run the standard STP protocol on those ports, emitting BPDUs to announce the Fast Path bridge as having a direct connection to a virtual root bridge with maximum priority. Hence, Fast Path bridges are automatically selected as root bridges by the standard bridges and a number of separate trees are built rooted at the fast path core. Note that no frame encapsulation is needed to traverse the core.

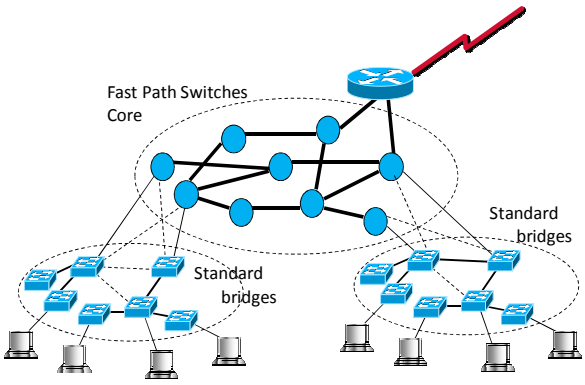


Figure 5. Hybrid network of Fast Path bridges and standard bridges in core-island mode with virtual root activated at the edge Fast Path bridges.

F. *Using Etherproxy for Scalability:*

There are two main problems to solve in Ethernet scalability: efficient layer two routing over Ethernet (replacing spanning tree protocol to overcome its limitations in active links and size), and broadcast reduction, to prevent host load processing excessive broadcasts. Fast Path focuses on the routing and Etherproxy [7] in the reduction of broadcast. Two aspects of broadcast are of interest: broadcast between bridges and broadcast processing at hosts

Etherproxy is a recent proposal which can be combined with Fast Path because it is focused on minimizing broadcasts to improve Ethernet scalability to higher limits (e.g. 50K hosts) that would impose significant load in hosts for processing all flooded ARP Requests.

Etherproxy minimizes broadcast traffic in the network by caching IP-MAC address pairs learned from ARP Reply packets, responding directly to ARP Requests that hit its ARP cache. The recommended location of Etherproxy devices is at the edge of the network, with one Etherproxy per up to 500 hosts (which is the broadcast domain size recommended by Cisco). Etherproxy can be implemented either as a separate device or inside the switches. Adding the Etherproxy functionality to Fast Path switches has significant advantages: first, reusing of the address learning resources of the Fast Path switch by the Etherproxy functionality, a close to 100% hit rate of ARP cache by using automatic refresh of the ARP cache addresses by unicast frames of user traffic traversing the switch and the perfect location at network edge. Among the costs are: increased processing at switches of the ARP Requests by sending ARP Reply to requesters. Etherproxy is self configuring, like Fast Path, so the combination of both stays self configuring.

III. RELATED WORK

The need of scaling Ethernet campus networks and datacenters to a single IP subnet is today accepted as the best way to simplify IP address management [3]. Two main obstacles have been identified as roots of the Ethernet scalability problem: the excessive broadcast traffic processing at hosts and forwarded by bridges, and the spanning tree protocol limitations. Solving the excessive broadcast problem requires alternative mechanisms to ARP for host address and location resolution and the spanning tree needs to be replaced by mechanisms for efficient routing and forwarding without restrictions.

The three main proposals in this area use link state routing (IS-IS or OSPF) protocol on layer two to build routes or shortest path trees between bridges. These are Shortest Path Bridges [2], RBridges (TRILL) [3] and SEATTLE [11].

The first two focus on the routing problem and do not pay special attention to the broadcast problem. Their routing complexity, performing shortest path computation between all bridges, exceeds the simplicity and network stability of Fast Path address learning.

SEATTLE pays special attention to the broadcast problem. It uses a one-hop distributed hash table to cache ARP table entries in a distributed form, reducing cache sizes at switches. Hosts are registered by its parent switch at its *resolver* switch obtained by hashing its MAC address. SEATTLE uses additional encapsulation to carry packets between switches. Failure or recovery of a resolver seems

to be the worst case condition, taking several seconds due to the dead interval detection of OSPF and host re-registration. There is no procedure defined for SEATTLE switches to interoperate with regular Ethernet switches.

Rbridges provide optimal pair-wise forwarding and support for multipathing of both unicast and multicast traffic. They achieve these goals using IS-IS routing and encapsulation of traffic with a header that includes a hop count to prevent loops and specific RBridge identifiers. Rbridges do not fully address the scalability problem due to broadcast but can limit host processing of broadcasts by performing ARP proxying for their attached hosts. RBridges are fully miscible with standard IEEE 802.1 bridges and end nodes at the cost of complexity: at each RBridge hop the destination address of the next RBridge must be inserted in the outer header.

Shortest Path Bridging (SPB) was initially proposed as an alternative of the complexities for configuration and optimization of Multiple Spanning Tree Protocol (MSTP). SPB provides logical Ethernet networks on native Ethernet infrastructure using a link state protocol to advertise topology and logical network (VLAN) membership. Packets are encapsulated at the edge either in mac-in-mac 802.1ah or q-in-q 802.1ad frames and transported only to other members of the logical network. Unicast and multicast are supported and all routing is performed on symmetric shortest paths. SPB bridges are compatible with standard bridges in core-island mode and do not limit broadcast.

IV. Evaluation

We describe here the evaluation we have carried out regarding complexity, amount of stored state information needed and infrastructure utilization. We also provide some performance measures obtained from a proof-of-concept Linux implementation and via software simulations.

A. Complexity

We now compare the message, state and computational complexities of Fast Path bridges with those of transparent bridges using the spanning tree protocol. Both the spanning tree protocol and routing protocols like IS-IS are *proactive*: in the spanning tree protocol, every node periodically emits its best BPDU (lowest cost route to root bridge) to its neighbours, processes the d (d being the average node degree) BPDUs received from them, to select the neighbour offering the shortest distance to the root bridge as its parent bridge; this means that message complexity is $\Theta(d)$. Shortest path bridges using link state protocols (Djikstra shortest path algorithm) have, for a network with N bridges, N^2 (minimum $N \cdot logN$) complexity. Besides this, they need an additional synchronizing mechanism to prevent loops caused by temporary route inconsistencies. Although N may not be too big, each bridge must keep informed the others of the hosts associated to him. This means that forwarding tables may grow big and the amount of control traffic significant to keep updated the list of active hosts.

Fast Path is a *reactive* protocol; fast path bridges do not exchange routing information periodically. The standard ARP Request and Reply message exchange between hosts that bridges use to set up on demand paths has no additional cost in messages, with the exception of frames arriving to a bridge with an expired route, typically after a failure of a link or node. When this happens, extra messages (path request/path reply or standard ARP Request/Reply) are generated to rebuild the path.

Regarding to *stored state*, fast path bridges store an amount of state information similar to standard bridges. Standard bridges learn MAC addresses of active hosts by associating each address to a certain bridge port and maintaining a cache expiration timer per learned address. Fast Path bridges store the same association of MAC addresses to each port, but use two different timers for locked (short duration) and learning states (long duration) respectively, of the association of a MAC address to a port. So, only an additional and shorter timer is used during the path establishment phase. Handling the expiration of addresses for the shorter locking period requires some additional computational in the bridge. For the second timer, the computational effort for handling expiration of addresses at ports in learned locked state is equal to standard bridges (with the same timeout period, default 300 seconds), but the number of learned addresses with timers to handle is much lower. Note that there is no flooding of unknown unicast destination address frames, as this effort of learning the path is performed in the locking state.

B. Infrastructure utilization

In this section we compare the number of active links when using Fast Path and the Spanning Tree Protocol.

Fast Path does not block any link, so all the L links in the topology are active (although a link may not be used at a given time). STP only activates $N-1$ links. The ratio of active links with Fast Path versus STP is then:

$$U = L/(N-1) \qquad (1)$$

Substituting $L = N*d/2$ in (1):

$$U = N*d/((N-1)*2) \approx d/2 \qquad (2)$$

Table I shows the infrastructure utilization ratio range for network degrees of 4, 6 and 8. Highest (leftmost) range values correspond to 16 node networks while lowest (rightmost) range values correspond to increasing values of N (up to 256). The improvement in the ratio of active links for Fast Path ranges from 2 to 4,3 times for average network node degrees of 4 to 8 respectively.

TABLE I. FASTPATH TO STP RATIO OF ACTIVE LINKS

Average node degree	D	4	6	8
Active links ratio range	U	2,1-2,0	3,2-3,0	4,3-4,0

A more detailed evaluation of the instantaneous utilization of active links is performed in the throughput section, where the most loaded link at every situation is found and used to determine maximum network throughput.

C. Linux Implementation

A proof-of-concept of the Fast Path bridge protocol has been implemented on Linux kernel 2.6, working on user

space. This implementation is oriented to functional verification and not to maximize or demonstrate performance, which would require an implementation at kernel space.

To verify compatibility with standard services like DHCP two hosts were connected to the campus network via a triangle composed of three Fast Path Switches implemented on standard PCs running Linux with the modified bridge functionality. The test network is shown in Fig. 6.

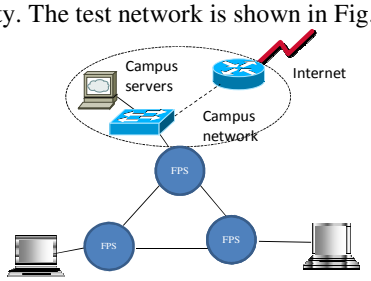


Figure 6. Fast Path Switches Proof-of Concept validation network

Our results show that hosts get their addresses via DHCP normally. And its access to Web services, file transfer and video playing operates normally. Broadcast loops do not occur even when connecting two ports of the same bridge. However, loops appear when the standard campus edge switch is connected via two or more links to the Fast Path network. It is worth noting that the network scenario for interoperation of Fast Path and standard bridges is the core-island mode described in section II.C and fig. 5, where the enhanced switches create a mesh at the network core and the standard bridges create trees attached to it. With this configuration, if each Fast Path bridge emits standard BPDUs announcing a virtual root bridge with maximum priority, fully separated trees are created at the bridge islands and loops are prevented.

We measured and compared ping delays between two hosts separated by a Linux-based Fast Path switch with those of a standard D-link 10/100 Mbps Ethernet switch with the same connectivity. Auto negotiation mode was set in both cases. When the host does not have the MAC of the destination host in its cache, it issues an ARP. The first ping may take up to 48 ms in the Linux Fast Path and only 2.43 ms in the standard hardware switch because a transfer between kernel and user space is involved to set up the path in Linux. Once the path is set up with the first ARP Request/Reply, the response to the ping only takes 238 microseconds on average for Linux Fast Path bridge and 200 microseconds for the standard hardware bridge. The reason for the high switching speed of the Linux when an address has been learnt, is because in this case forwarding is performed directly by the kernel.

D. Simulations

A Fast Path simulator has been implemented in Omnet (INET framework) [8] by modifying the Ethernet switch implementation. We compared the performance of the Fast Path protocol, shortest path routing, and spanning tree protocol, focussing on the data flow performance resulting from each forwarding mechanism, not in the dynamics of the establishment of the paths. We used two network topologies: a generic two-level enterprise network (fig.7) and a pan european reference network [10].

1) Enterprise network

To obtain the resulting latency, we simulated TCP sessions of different data sizes (2 K, 100 K and 100 Mbytes) repeated 100 times with exponential probability between 3 sender hosts and 3 receiver hosts (labeled 1, 2, 3 and 4, 5, 6 respectively at Fig. 7). All network links have a propagation delay of 1 microsecond and a transmission rate of 100 Mbps. When the network is lightly loaded, network frame latency from host to destination (measured from Ethernet layer of source to Ethernet layer of destination) equals to 275 microseconds on average for all three protocols with small variations.

With low to medium loads (100K sessions), the network latency reaches 370 microseconds, on average, for all three protocols. With high loads (100 MB sessions), latency stays moderate for routers and Fast Path bridges but grows to 1645 microseconds for the spanning tree, due to congestion at the links shared by many flows (links around the root). We conclude that Fast Path latency is equivalent to that of shortest path routing and both are superior to spanning tree with high network loads due to congestion at spanning tree links.

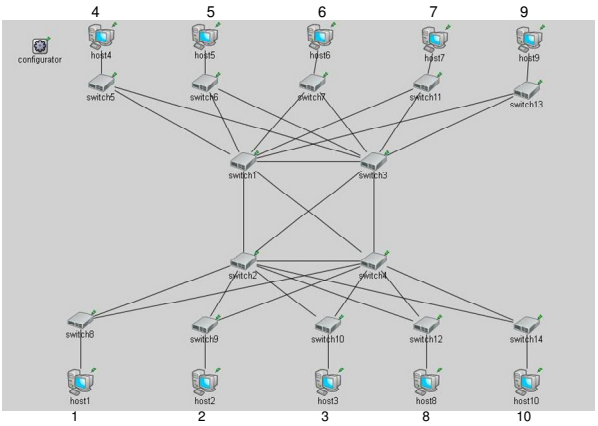


Figure 7. Enterprise network. Active topology for shortest path routers and Fast Path switches.

To compare the throughput of the Data Center network with the three protocols, we simulated traffic from the hosts located in the lower part of figure 7 to the hosts depicted in the upper part, generating the same traffic for all communications. Traffic was generated from host 1 to 9, 2 to 7, 3 to 5 and 8 to 4. UDP traffic of increasing intensity was generated (from 1 to 60 KB sent per exponential average time of 150 milliseconds, 3 second simulations), in order to force saturation at the most loaded link of the network, and the load at that link was registered. Fig. 8 shows the saturation process till reaching near 100% of load at the most loaded link for the three protocols.

Shortest path and spanning tree networks saturate at 16% of the maximum host link capacity while Fast Path saturates at 32 % approximately. In this particular case, and contrary to the usual behaviour, shortest path routing saturates slightly before spanning tree. This is specific of the enterprise topology (where spanning tree paths are often also shortest

paths). Fast Path saturates at significant higher loads than spanning tree and slightly higher than shortest path routers because it achieves path diversity because each path is set up asynchronously for each host upon ARP. Some paths go through the left vertical link of the core and some through diagonal link.

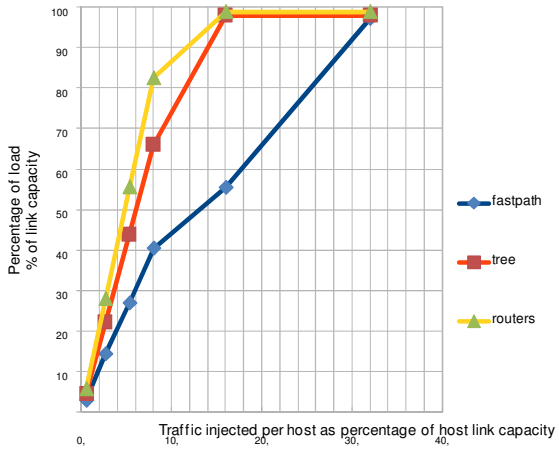


Figure 8 . Throughput comparison of Fast Path, spanning tree and shortest path routers for enterprise network. Percent of maximum load at bottleneck link versus injected traffic as a percentage of maximum capacity of source host link.

Spanning tree concentrate routes over the links close to root bridge, congesting those links.The core vertical link on the left becomes the bottleneck link.

2) Pan European network

We also simulated a pan-European core network, a flat mesh of 16 nodes [10]. UDP traffic with message lengths of 1 to 60 KB is sent with exponential distribution of average 75 ms. Link delays are according to map distance, between 1 and more than 3 msec. All links have the same capacity. Traffic is originated at hosts at west nodes and directed toward east nodes to facilitate saturation of links. Activation of traffic at nodes is sequenced randomly with an average delay of 0.5 seconds between activations. Links saturate slightly later with Fast Path than with shortest path routers and much later than with spanning tree. When no there is no sequencing of traffic, the results for Fast Path are equal than for shortest path (not shown in figure).

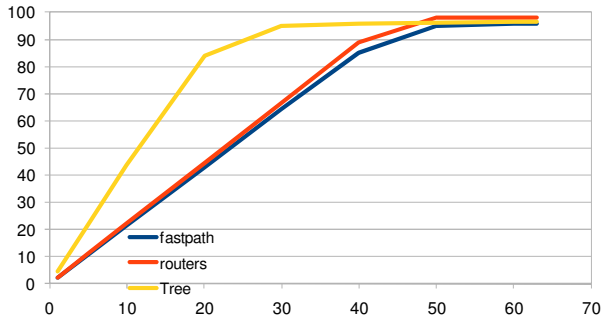


Figure 9. Throughput comparison of pan european network in % of most loaded link versus % of average traffic load applied at the sending host link

V. Conclusions

We have presented Fast Path, a variant of the transparent bridge paradigm. Fast Path Switches use standard ARP frames to set up fastest paths between hosts. Preliminary performance evaluations show superior performance to spanning tree and similar to shortest path routing in delay and better than both in terms of throughput and protocol complexity. A working proof-of-concept has been implemented on Linux. Throughput simulation results show an interesting potential for traffic load distribution that should be further investigated. Next planned steps are hardware implementations on NetFPGA with Openflow. The combination of Fast Path protocol with Etherproxy in edge switches will likely enhance protocol scalability through radical reduction of broadcasts, although it could impact the load distribution capability.

VI. Acknowledgment

This work was supported in part by grants from Comunidad de Madrid through Project MEDIANET-CM (S-2009/TIC-1468) and from Comunidad de Castilla la Mancha through Project EMARECE (PII1I09-0204-4319). Thanks to Bart de Schuymer, who implemented Fast Path Ethernet in Linux, and to Aaron Montalvo for the router simulations.

References

[1] IEEE 802.1D-2004 IEEE standard for local and metropolitan area networks-Media access control (MAC) Bridges. http://standards. ieee.org/getieee802 /802.1.html.

[2] M. Seaman. Shortest Path Bridging. http://www.ieee802.org/ 1/files/public/docs2005/new - seaman-shortestpath-par -0405-02.htm.

[3] Transparent interconnection of lots of links (TRILL) WG. Available on line at: http://www.ietf.org/html.charters/trill-charter.html

[4] Dalal, S., Metcalfe, R. Reverse path forwarding of broadcast packets. Communications of the ACM Vol. 21, No. 12 pp. 1040-1048, December 1978.

[5] ITU-T Recommendation Y.1731 - OAM functions and mechanisms for Ethernet based networks, Feb. 2008.

[6] IEEE 802.1ag - Connectivity Fault Management, http://www.ieee802.org/1/pages/802.1ag.html, 2007.

[7] Elmeleegy, Khaled and Cox, Alan L. EtherProxy: Scaling The Ethernet By Suppressing Broadcast Traffic. Proceedings of IEEE INFOCOM 2009, Rio de Janeiro, Brazil.

[8] Omnet simulator. Available online: http://www.omnetpp.org

[9] G. Ibáñez et al. Fast Path Bridges. http://ww.ieee802. org/1/ files/public/ docs2009/fyi-ibanez-fast path-0909-02.pdf

[10] NRS Reference Networks http://www.ibcn.intec. ugent. be/INTERNAL/NRS/index.html

[11] C. Kim, M. Caesar, and J. Rexford. Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises. In ACM SIGCOMM 2008, Aug. 2008