Treating Software-Defined Networks Like Disk Arrays

Zhiyuan Teo, Ken Birman, Noah Apthorpe, Robbert Van Renesse, Vasily Kuksenkov
Department of Computer Science
Cornell University
Email: {zteo, ken, apthorpe, rvr, vasily}@cs.cornell.edu

***Abstract*—Data networks require a high degree of performance and reliability as mission-critical IoT deployments increasingly depend on them. Although performance and fault tolerance can be individually addressed at all levels of the networking stack, few solutions tackle these challenges in an elegant and scalable manner. We propose a redundant array of independent network links (RAIL), adapted from RAID, that combines software-defined networking, disjoint network paths and selective packet processing to improve communications bandwidth and latency while simultaneously providing fault tolerance. Our work shows that the implementation of such a system is feasible without necessitating awareness or changes in the operating systems or hardware of IoT and client devices.**

I. Introduction

The potential uses of multiple paths in a network have attracted significant attention, and many IETF standards [8] [10] have been finalized or are now being finalized to expose and exploit these capabilities. The reasons for this interest reflect multiple goals: multipath networking provides (1) improved resilience to failures and (2) improved network load-balancing, leading to (3) better data throughput, and hence improved user experience.

The basis for multipath networking is conceptually simple. Multipath networking can be seen as another form of parallelism, with the objective of improving network performance subject to some underlying constraints. These underlying constraints are highly varied, and may not necessarily be bound by engineering or physical limits. For example, one important criteria in MPTCP [8] is flow fairness. Under this criteria, subflows of MPTCP must not use an unfair share of network link bandwidth should they transit the same physical link. This constraint is not imposed by a topology or environmental limit, but is nonetheless required for acceptable deployment. The choice of which constraints to address will determine the optimality of the resultant solution over a broad range of applications.

For our work, we take the position that IoT devices are closed black boxes that are not amenable to modification of any kind, whether in software or hardware. This is a reasonable assumption because many consumer or industrial grade IoT devices are commodity-off-the-shelf components that are generally designed to be tamper-proof and maintenance-free. Critically, this constraint means that any kind of change designed to improve networking experience must be confined to the network switching equipment itself.

In our target setting, we assume that network operators use switched Ethernet and are open to upgrading their switches to OpenFlow [14]-capable models. However, we do not assume that users can or will upgrade their IPv4 networking equipment or software, although they may nonetheless desire the benefits offered by multiple paths in the network. For example, in networks created to support IoT instrumentation of the smart power grid, embedded sensors may not be subject to reprogramming, but could benefit from the resilience offered by a multipath network. Beyond these two assumptions, we do not impose any other limitation, so users are free to run their own protocols and software, oblivious to the underlying network. We believe these assumptions to be valid and powerful, as they cover many existing deployments and have the advantage of frustration-free backward compatibility.

The contributions of our work are as follows:

- RAIL (Redundant Array of Independent Links), an innovative set of network redundancy schemes adapted from RAID, that collectively provide high speed and reliable packet transportation, while being tunable in terms of latency and bandwidth efficiency.
- The design of dedicated network processing units (NPUs), analogous to RAID controllers, to support RAIL schemes.
- Engineering solutions that require no changes to existing hardware and software beyond network switch upgrades.
- A prototype and microbenchmarks to validate our claims.

Taken together, our work fills an unoccupied niche in computer networking by providing selectable (1) improvements to end-to-end network performance through packet processing and redundant routing and (2) the realization of high-assurance networking through zero-downtime failure recovery, while being (3) a drop-in upgrade that is (4) fully backward-compatible with existing end-host equipment.

II. Background

A. Ethernet and the Spanning Tree Protocol

Conventional Ethernet is poorly matched to our goal of providing a multipath-capable network because it has an important restriction that mandates a spanning tree topology for correct operation. This spanning tree requirement is not arbitrary; it was designed to connect all participating hosts while eliminating catastrophic network loops. Conventional Ethernet

networks perform loop elimination and spanning tree construction by running a distributed algorithm such as the Spanning Tree Protocol (STP) or the improved Rapid Spanning Tree Protocol (RSTP) [19]. In STP and RSTP, Ethernet switches in the same network segment collaborate and agree on which network links to use such that a single spanning tree connects the entire network without any cycles. In the process, STP or RSTP disables links that would otherwise result in loops. All traffic transits the spanning tree, which becomes a network-wide bottleneck. Accordingly, conventional Ethernet networks do not typically feature link redundancies, and where such redundancies exist, they cannot be taken advantage of without special configuration. Worse, failure recovery and redundant link activation typically take between several seconds to half a minute, resulting in network hiccups even if no apparent physical partitions have been introduced.

B. Link-layer multipath techniques

To circumvent the shortcomings of a network spanning tree, link-layer techniques for enabling multiple paths over Ethernet have been developed [2] [11]. ECMP [2] is a load-balancing strategy that takes advantage of link redundancies in a network. Under ECMP, each flow is hashed to a single path from a set of available paths between the source and the destination. Although each flow transits only a single path, the overall effect is to spread distinct flows across all the available paths, thus load-balancing the network as a whole.

Another link-layer protocol for removing the single spanning tree restriction is Multiple Spanning Tree Protocol (MSTP) [11]. MSTP allows concurrent multiple spanning trees to exist within the same network by mapping each spanning tree to a multiple spanning tree instance (MSTI). Each virtual LAN in the network can then be associated with one of the spanning tree instances, thus improving aggregate network availability and reliability when link breakages happen. Bottlenecks are also reduced because the network now distributes its load over a greater number of transit links. However, within an MSTI, participants are still subject to spanning tree outages and repair time. Thus, while some VLANs may experience little disruption, other VLANs may be severely impacted. This can seem counterintuitive especially if redundant physical paths do exist in the network and no actual physical partitioning was caused.

Unfortunately, ECMP and MSTP are not true multipath techniques because each flow does not simultaneously traverse different paths. The 'multipath' in these approaches are aggregate statements that multiple flows will take different singular routes through the same physical network. Thus, while the network experiences better load balancing and reliability as a whole, the failure of a single link would still be catastrophic for flows transiting that link.

C. Network-layer multipath techniques

Attempts at multipath networking have also been made on layer 3 protocols and have recently started to gain traction on modern operating systems, although widespread support is still sparse [9]. MPTCP [8] is a protocol designed to improve overall connection performance by splitting a TCP flow into subflows, each of which are transmitted down different network interfaces. These interfaces may be real or virtual, so a multi-homed setup is not necessary although it is desirable [10]. In an ideal setting, each participating network interface takes a completely disjoint path to the destination, maximizing throughput. Should any subflow fail due to a link breakage, the remaining subflows transparently take up the load and present the illusion of continuity to the user.

Because MPTCP operates at the network layer, it is compatible with any existing network switch that can carry IP traffic. However, MPTCP does not have knowledge of the underlying link layer and actual physical communications paths so it is possible for subflows to transit the same physical links. Thus a single failure of the shared link can still lead to multiple subflow losses.

D. Software-defined networking

To build a network free of the restrictions from conventional Ethernet, we turn to software-defined networking. Software-defined networking (SDN) is a modern abstraction that allows access to a network switch's routing fabric. In SDN models, the switch's control plane is made accessible to a special external software entity known as a controller, to which all data switching decisions are delegated.

The most widely deployed SDN standard today is known as OpenFlow [14]. OpenFlow is managed by the Open Networking Foundation and has seen significant evolution through multiple versions. To provide maximum compatibility, we describe RAIL implementation using OpenFlow 1.0.

E. RAID

We conjecture that a parallel exists between disks and network paths as data mediums, so the performance and protection schemes deployed on disks are also applicable in a network. RAID [18] is the classic work that explores various techniques of storing data on a set of independent disks for the purpose of improving redundancy and performance. Data storage using RAID is organized into standardized schemes or levels, each scheme providing a different set of benefits and tradeoffs.

III. RAID AND RAIL

We propose a set of user-tunable parameters to improve the performance and/or reliability of a flow. These parameters are conceptually similar to those available in RAID, the set of redundancy schemes used in disk arrays. The analogue of disks in our system are disjoint paths, hence the term redundant array of independent links (RAIL).

In the case of RAIL, the tunable parameters can be seen as a continuum of tradeoffs between latency/reliability and bandwidth efficiency. At one extreme end of the spectrum, each packet in a flow can be replicated onto multiple disjoint paths. The receiving end delivers the first arriving packet to the application and discards the duplicates. Such a scheme minimizes latency and improves the stability of the flow, while also tolerating up to $n-1$ link or switch failures, at a cost of n times the bandwidth.

On the other extreme end of the spectrum, each disjoint path can be seen as a separate channel through which data can be sent, so each successive packet in a flow can be transferred

down whichever path is first available (thus avoiding the problem of sending too many packets down congested paths). In a lightly loaded network, approximately $1/n$ of the packets in a flow can be sent down each path. This scheme maximizes bandwidth efficiency but clearly sacrifices on flow stability and latency, since the entire flow is now dependent on the slowest link. It also does not tolerate link failures since this scheme does not feature any redundancy.

In between these two ends, a parity protection scheme may be used to provide low-cost tolerance to link failure. Alternatively, a simple and more general k out of n scheme may be used to replicate packets such that a flow can tolerate a loss of up to $n-k$ disjoint paths, while providing a lower-bound latency performance of the $n-k+1$th slowest disjoint path.

Like RAID controllers, RAIL schemes depend on network processing units (NPUs) to handle flow packets, so we present their design first.

A. Network Processing Unit

The NPU is an abstraction that provides realtime packet processing services. NPUs have been proposed in the past [24] [23] to perform in-line network packet processing. In the context of our work, NPUs need to provide services that include (but are not limited to) automatic packet buffering, re-ordering, rewriting and de-duplication.

One of the responsibilities of the NPU in all RAIL schemes (except RAIL 1) is the tagging of ingress packets. The NPU does this by rewriting Ethernet packet headers. Each disjoint path is associated with a destination meta-address, which can be any unique Ethernet MAC address that is not an in-use or reserved (eg. broadcast and null) address. To designate a packet for transit over a certain disjoint path, its Ethernet destination field is overwritten with the meta-address of the selected path. At the OpenFlow switch, rules are installed to match these special Ethernet addresses, with corresponding actions to forward matching packets down their respective disjoint paths. Tagging packets this way permits efficient forwarding of disjoint path packets as the switch hardware can perform header matching and packet forwarding at line rate.

It is important to note that tagged packets do not need to be equally distributed across the selected disjoint paths. This allows the controller to collaborate with the NPU on dynamic traffic shaping strategies. For example, across disjoint network paths that have large bandwidth disparities, the controller may choose to instruct the NPU to tag proportionately more packets (or more aggregate bytes) for higher bandwidth disjoint paths, favoring them over those with lower available capacities.

Tagging packets with their disjoint path meta-addresses is a necessary step, but alone is not sufficient for the functioning of the system. When individual disjoint path latencies are different, it is possible for network packets arrive out of order. The direct delivery of these potentially reordered packets may have unintentional effects on the receiving system. For example, TCP may interpret out-of-order packets as an indication of packet loss and accordingly retransmit the previous packet, while reducing the data transmission rate. This runs counter to our design goal of non-interference with user protocols and systems. Therefore, the NPU also needs to provide some mechanism to preserve packet ordering at the egress switch.

To introduce packet ordering, some notion of sequencing is required. The intuitive answer to this is to use the sequence information provided by the packet itself. Unfortunately some IPv4 traffic, notably UDP, do not contain a sequence number field. Although it would be relatively straightforward to augment the packet with an extra integer field, in practice this is risky because large packets may be written with no headroom for extra data, and the inclusion of these mere extra few bytes may cause the packet to exceed the network's MTU value. This is disastrous as it would result in that packet being dropped. Thus, the key challenge in including a sequence number is the identification of a non-critical field that can be overwritten for this purpose. In our system, we have chosen to repurpose the 16-bit Ethertype field for sequencing. While this is a reserved field that is used for classifying Ethernet traffic type, we reasoned that the field was safe to hijack because Ethernet forwarding does not depend on the value stored there. Furthermore, because RAIL only handles IPv4 traffic, all network packets encountered by the NPU will effectively have a constant value of `0x0800` in the Ethertype field.

To ensure that our use of the Ethertype field does not interfere with other network devices that interpret reserved Ethertype values, we picked the IEEE unallocated range [6] `0xB000` to `0xC000`. This gives the system 4096 possible sequence values before wrapping, which is sufficient in our experience.

After flow rules on all involved switches have been set up, the last practical matter pertains to packet reassembly on the NPU at the egress switch. During RAIL service negotiation, the NPU is informed of the set of reserved destination Ethernet addresses corresponding to the original flow, and the original destination Ethernet address for that flow. Incoming packets are then binned according to the original flows they map to. Duplicates are discarded. Whenever sufficient packets have arrived, the original packet is reconstructed and rewritten to reflect its original Ethernet destination address and Ethertype (which is always `0x0800`). The packet is then put into a re-order buffer that maintains the original packet sequence. The re-order buffer releases packets from the NPU as they become available in the correct sequence. The egress switch then forwards the packet along to the true destination.

B. RAIL 1

We now describe the individual RAIL schemes. Recall that the equivalent scheme in RAID 1 is a simple mirroring process that trades storage capacity for speed and fault tolerance. Analogously, the RAIL 1 scheme replicates data packets across multiple disjoint paths, with the effect that latency and fault tolerance is improved at the cost of bandwidth efficiency. If the disjoint paths have approximately similar end-to-end latencies, RAIL 1 may also reduce latency variance.

For simplicity, we describe the flow rules and actions for a unidirectional data transfer. Bidirectional data transfer can be achieved either by relying on the network's intrinsic backward path over its spanning tree, or by installing another unidirectional RAIL scheme in the opposite direction. Bidirec-

tional data transfers are not required to employ identical RAIL schemes in each direction.

On the ingress switch, a single matching rule for the selected network flow is installed with an action that multicasts packet output to physical ports corresponding to the relevant disjoint physical links. The switch automatically replicates the network packets without further intervention from the controller. Switches along the disjoint paths act as mere waypoints and thus only need one rule each to forward network packets to the next hop. Because the egress switch potentially receives redundant copies of each network packet, de-duplication is required. At this egress switch, the subflows are redirected to an NPU that removes redundant packet copies, before re-emitting the packet back to the switch for delivery to the destination.

C. RAIL 0

On the other end of the RAID spectrum of tradeoffs is the ability to aggregate multiple storage volumes into one single logical volume. This maximizes the storage efficiency of the scheme, but completely trades away any fault tolerance. In RAIL 0, the available disjoint physical link bandwidths are aggregated together into one logical link. This translates to maximal bandwidth utilization efficiency, but has a statistically greater failure rate than RAIL 1 or even single path connections. RAIL 0 also suffers from higher packet jitter and higher latency, since the latency effects of all links will be evident at the destination.

At the ingress switch, several flow rules are required. First, a rule is installed to divert the flow into an NPU that tags each packet with the meta-address of some disjoint path. Another set of rules matches each meta-address and forwards the tagged packets onto their corresponding disjoint path. Switches along the disjoint paths merely forward packets on to their respective next hops, so only one rule is required on each of them.

At the egress switch, a set of rules are installed to forward the tagged packets to a local NPU. This NPU will buffer, reorder and rewrite packets such that emitted packets appear identical in content and sequence to the original flow at the ingress switch. Another rule on the egress switch then takes these packets to the actual destination.

D. RAIL 3 - 6

RAID levels 3-5 [1] are similar on account of using parity protection to secure data from single failures, with the only differences being the sizes and placements of parity blocks. In RAID 3, this parity block size is one byte while RAID 4 uses a larger block size. Both RAID 3 and RAID 4 use a dedicated parity disk. RAID 5 is similar to RAID 4, except that parity blocks are distributed evenly over all disks. Because these schemes are conceptually identical, we describe RAIL 4. The RAIL 4 scheme has a relatively light traffic footprint while being tolerant of single failures.

At the ingress switch, rules are installed to divert a target flow into the NPU. For each ingress packet, the NPU needs to split the Ethernet payload into $n-1$ disjoint fragments, where n is the number of disjoint paths chosen. If the resultant fragments have uneven sizes, for parity computation purposes the smaller ones are padded to the right with a zero such that all fragments have the same size. A parity fragment is then constructed by computing the XOR of all fragments, essentially assuming the form of forward error correction.

Each of the n fragments are then given an Ethernet header with its original source address, designated disjoint path destination meta-address, and an Ethertype corresponding to the sequence number of the fragment. The synthesized packets are then sent to the switch for transmission along disjoint paths. Padding bytes are not sent with the fragments.

At the egress switch, subflows are sent to the local NPU. Because of the presence of a parity subflow, only $n-1$ packets are required for the reconstruction of an original packet so the flow is able to tolerate the complete loss of one disjoint path. However this reconstruction is tricky: if the excluded fragment had been padded for parity computation, its regeneration will include the padding byte. To fix this problem, the reconstruction process consults the size field in the IPv4 header and checks this against the sum of all fragment sizes. A difference signifies the presence of the padding byte and it is truncated from the regenerated fragment. The original Ethernet payload can then be recovered by rejoining the fragments in sequence order. Finally, the Ethernet header is prepended to yield the original packet. The NPU then buffers and reorders the packet for appropriate release to the egress switch, which then conveys the packet on to the destination.

RAIL 6 could theoretically improve upon the reliability offered by RAIL 4 to tolerate double losses, although its implementation is significantly more complicated due to the need to construct a computationally-expensive second parity packet.

E. Generalized k of n RAIL protection schemes

If the RAIL 0 and RAIL 1 schemes are conceptually at diametrically opposed ends of the tradeoff spectrum, then other alternative schemes can be designed to bridge the gap and provide continuity between the two extremes. We now describe a general scheme that is identical in spirit to hybrid RAID 1 + 0 setups. This scheme is computationally cheap and simple to implement, albeit imperfect in its bandwidth usage.

Given a set of n selected disjoint paths, the paths are first ordered in a ring. Each successive ingress packet is duplicated over the next $k+1$ paths in the ring. For example, if $n = 3$ and $k = 1$, the first packet in the flow will be sent over paths 1 and 2, the second packet over paths 3 and 1, the third packet over paths 2 and 3. This scheme has the property that the failure of any k paths still allows complete reconstruction of the original flow at egress. Additionally, the ratio of the bandwidth efficiency of this scheme to the maximum possible without duplication is $\frac{1}{k+1}$. Tuning the parameter k therefore allows the user to set the tradeoff between fault tolerance and bandwidth efficiency. When $k = 0$, the algorithm converges to RAIL 0, with maximum bandwidth efficiency but no fault tolerance. On the other hand, when $k = n-1$, the algorithm converges to RAIL 1, tolerating the failure of all but 1 disjoint path, at the cost of experiencing a $1/n$ bandwidth efficiency ratio.

[1] We omit the discussion of RAIL 2 because RAID 2 uses Hamming codes and the equivalent network scheme is needlessly complicated without yielding significant benefits.

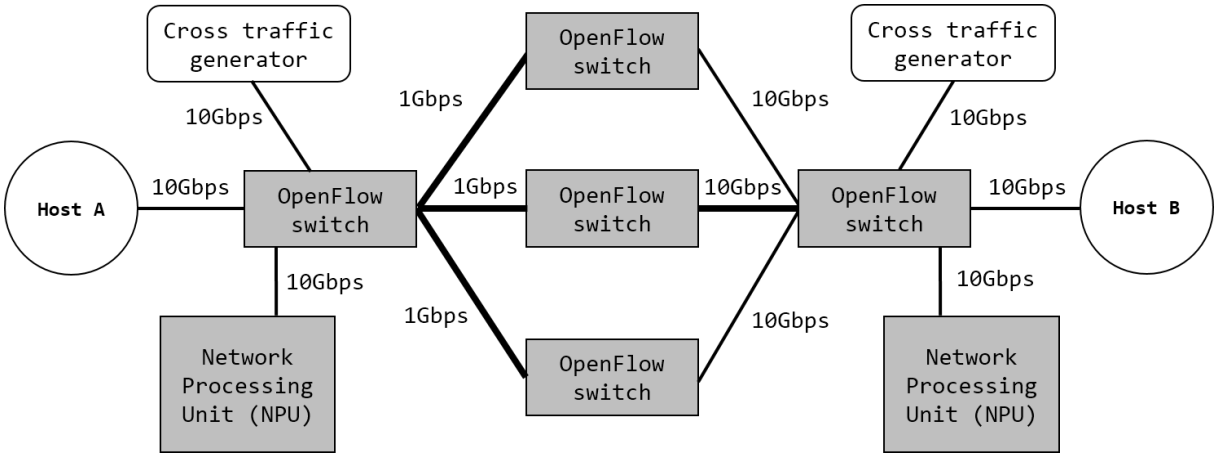

Fig. 1. The topology used in our evaluation. Bold lines represent spanning tree links.

The manner in which packets are tagged and forwarded at the ingress switch, as well as untagged, de-duplicated, reordered and reassembled at the egress switch, is exactly identical to the process described in RAIL 0.

F. Scaling the RAIL service

The main bottleneck in a RAIL deployment is the NPU, as real-time packet processing is an intensive operation. A single NPU on an ingress switch may not be sufficient to support all interested clients simultaneously.

To solve this problem, service may be linearly scaled by attaching additional NPUs where they are needed. This is typically as simple as identifying a spare port on the ingress OpenFlow switch and plugging another new NPU into that port; the OpenFlow controller can then register the NPU for immediate use. The controller may also load-balance the locally NPUs dynamically by shunting flows to less-loaded units.

If no more spare ports are available, a possible solution would be to spread wire connections on the existing switch over two new switches, effectively spacing out the cables over more switch ports to avail more attachment points for NPUs. The old network topology can be functionally retained by bridging these two switches with a high speed interconnect, for example through a 40Gbps link aggregation switch port.

IV. Evaluation

To evaluate our system, we used a Dell Force10 S4810 switch partitioned by port banks into five OpenFlow instances, in effect simulating five physical OpenFlow switches (Fig 1). The instances were connected in a way to simulate a network topology with three disjoint paths between a source to a destination. All physical links had a capacity of 10Gbps except for one link on each disjoint path, which was deliberately throttled in hardware to 1Gbps. Therefore, the total bandwidth available to any single disjoint path between two end hosts that traversed this network was 1Gbps. Because there were three disjoint paths available, the maximum available bandwidth was 3Gbps. A spanning tree-based path, on account of a singular end-to-end path, therefore had an available capacity of only 1Gbps. Two additional systems were introduced to the edge network switches connecting the two end hosts to inject cross traffic into the spanning tree path.

Two NPUs were connected to the experimental setup. One NPU was located on each virtual switch corresponding to the edge network switches that connected the two end hosts. The NPUs were NetFPGA 10G cards, each providing four 10-Gigabit Ethernet ports connected to a Xilinx Vertex-5 FPGA. Purpose-designed logic in the FPGA performed the various duties of rewriting, reordering and deduplicating packets. The cards were well-suited for line rate traffic; during experiments, we noticed no dropped packets and the latency cost of forwarding packets through the card was too small to measure.

To benchmark end-to-end bandwidth in our system, we ran iperf, a TCP/UDP bandwidth measurement tool to measure aggregate bandwidth between two hosts. Because of a persistent hardware configuration issue in the 10G network interface cards we used, the MTU used in the experiments was 536 bytes. Cross traffic was generated by running bidirectional iperf. End-to-end latencies were measured using the system ping utility and listed respectively in the table as min/avg/max over 100 samples. Our microbenchmark results are as follows:

A. Microbenchmark results

	Ethernet STP	RAIL 0	RAIL 1	RAIL 4
latency[1] min/avg/max	0.122ms 0.152ms 0.185ms	0.126ms 0.166ms 0.196ms	0.125ms 0.160ms 0.210ms	0.125ms 0.158ms 0.184ms
bandwidth[1]	0.85Gbps	2.55Gbps	0.85Gbps	1.52Gbps
latency[2] min/avg/max	4.017ms 11.911ms 17.506ms	0.126ms 3.244ms 13.157ms	0.125ms 0.161ms 0.200ms	0.126ms 0.175ms 0.215ms
bandwidth[2]	0.51Gbps	2.02Gbps	0.85Gbps	1.52Gbps
link failures tolerated	0	0	2	1

[1] Without cross traffic. [2] With cross traffic.

V. Related Work

Path splicing [16] is a mechanism that provides multiple paths through a network through multiple routing trees. By allowing traffic to switch routing trees at each forwarding node, the system ensures path reliability even where disjoint path routing may fail. Path splicing has fast recovery time but flows are not redundantly routed and it experiences latency stretch.

Multiple Topologies for IP-only Protection Against Network Failures [12] describes the use of multiple topologies for transparent routing recovery and fault tolerance. Routers precompute some backup topologies and reroutes packets along the backup paths in the event of failures. It performs very well in realistic scenarios even though IP traffic only take single routes at any instance.

STAR [22] is a spanning tree-compatible protocol that improves QoS routing in an extended LAN. Packets are forwarded over the spanning tree by default but may also take shorter, non-spanning tree alternate paths where they are available.

SPAIN [21] is an Ethernet-based solution that implements redundancy by mapping strategically computed paths to separate VLANs. SPAIN provides increased bisection bandwidth and resistance to network failures. However, its implementation relies on static, pre-installed paths, and does not adapt to substantial network topology changes. SPAIN also does not offer a continuum of latency/bandwidth tradeoffs.

802.1 Ethernet link aggregation [7] combines several physically distinct network links on a switch into a single large logical link, similar to RAIL 0. With appropriate failover recovery, Ethernet link aggregation can also improve the resilience of the network against individual link failures. However, only two switching elements may participate in each aggregated link. When a participant switch fails, the entire aggregated link also fails. This is fundamentally different from RAIL 0, where link aggregation is the result of multiple paths across multiple switches. If a switch fails in RAIL 0, the aggregated link can continue to exist with reduced bisectional bandwidth. Ethernet link aggregation can be seen as a special case of RAIL 0, the latter of which is a more general scheme that allows for finer control.

Various work have been proposed to perform the high speed packet processing as required by an NPU. Split SDN Data Plane (SSDP) architectures [24] and loadable packet processing modules [23] offer industrial-strength alternatives to the FPGA designs in this paper, by integrating the packet processing requirements into an alternate data path that is directly connected to a switch co-processor. NetSlice [13] and DPDK [1] offer software alternatives that allow user-space programs a way to rapidly access and process network packets.

VI. Concurrent and future work

We would like perform more exhaustive benchmarks comparing our system to MPTCP and other related solutions. An obvious concern that was not discussed here is security. Two general classes of issues exist with data transport: confidentiality and anonymity. The topic of security involves a non-trivial discussion and merits its own paper.

VII. Conclusion

We presented the design of a novel solution that provides tunable high performance and reliability for OpenFlow data networks via RAIL schemes that are analogous to RAID. RAIL schemes are supported by network processing units, similar to RAID controllers. Our proposed system is backward-compatible with existing hardware and software. RAIL service capacity can be scaled linearly by adding more NPUs as required. Finally, the evaluation shows that our proposed system is practical and offers real, tangible improvements over existing network setups.

References

[1] Data Plane Development Kit. http://dpdk.org/

[2] IEEE 802.1Qbp. Equal Cost Multiple Paths, IEEE 2014.

[3] Reitblatt, Mark, et al. "FatTire: declarative fault tolerance for software-defined networks." Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013.

[4] Floodlight OpenFlow controller. http://www.projectfloodlight.org/floodlight/

[5] Al-Fares, Mohammad, et al. "Hedera: Dynamic Flow Scheduling for Data Center Networks." NSDI. Vol. 10. 2010.

[6] http://standards.ieee.org/develop/regauth/ethertype/eth.txt

[7] IEEE 802.1-AX 2008. Link Aggregation, IEEE 2008.

[8] A. Ford, C. Raichu, M. Handley, O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", IETF, RFC 6824, Jan. 2013. [Online]. Available: https://tools.ietf.org/html/rfc6824

[9] Kostopoulos, Alexandros, et al. "Towards multipath TCP adoption: challenges and opportunities." Next Generation Internet (NGI), 2010 6th EURO-NF Conference on. IEEE, 2010.

[10] R. Winter, M. Faath, A. Ripke, "Multipath TCP Support for Single-homed End-Systems", IETF, Internet-Draft draft-wr-mptcp-single-homed-05, Jul. 2013. [Online]. Available: https://tools.ietf.org/html/draft-wr-mptcp-single-homed-05

[11] IEEE 802.1Q-2011. VLAN Bridges, IEEE 2011.

[12] Apostolopoulos, George. "Using multiple topologies for ip-only protection against network failures: A routing performance perspective." ICSFORTH, Greece, Tech. Rep (2006).

[13] Marian, Tudor, Ki Suh Lee, and Hakim Weatherspoon. "NetSlices: scalable multi-core packet processing in user-space." Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems. ACM, 2012.

[14] OpenFlow Switch Consortium. "OpenFlow Switch Specification Version 1.0.0." (2009).

[15] Open vSwitch. http://openvswitch.org/

[16] Motiwala, Murtaza, et al., Path splicing. ACM SIGCOMM Computer Communication Review. Vol. 38. No. 4. ACM, 2008.

[17] POX. http://www.noxrepo.org/pox/about-pox/

[18] Patterson, David A., Garth Gibson, and Randy H. Katz., A case for redundant arrays of inexpensive disks (RAID). Vol. 17. No. 3. ACM, 1988.

[19] IEEE 802.1D-2004. Media Access Control (MAC) Bridges, IEEE 2004.

[20] Weatherspoon, Hakim, et al., Smoke and Mirrors: Reflecting Files at a Geographically Remote Location Without Loss of Performance. FAST. 2009.

[21] Mudigonda, Jayaram, et al., SPAIN: COTS Data-Center Ethernet for Multipathing over Arbitrary Topologies. NSDI. 2010.

[22] Lui, King-Shan, Whay Chiou Lee, and Klara Nahrstedt. "STAR: a transparent spanning tree bridge protocol with alternate routing." ACM SIGCOMM Computer Communication Review 32.3 (2002): 33-46.

[23] Narayanan, Rajesh, et al., A framework to rapidly test SDN use-cases and accelerate middlebox applications. Local Computer Networks (LCN), 2013 IEEE 38th Conference on. IEEE, 2013.

[24] Narayanan, Rajesh, et al., Macroflows and microflows: Enabling rapid network innovation through a split SDN data plane. Software Defined Networking (EWSDN), 2012 European Workshop on. IEEE, 2012.