In-network computing to the rescue of faulty links

Hans Giesen
University of Pennsylvania

Lei Shi
University of Pennsylvania

John Sonchack
University of Pennsylvania

Anirudh Chelluri
University of Pennsylvania

Nishanth Prabhu
University of Pennsylvania

Nik Sultana
University of Pennsylvania

Latha Kant
Vencore Labs

Anthony J McAuley
Vencore Labs

Alexander Poylisher
Vencore Labs

André DeHon
University of Pennsylvania

Boon Thau Loo
University of Pennsylvania

ABSTRACT

Failing network links are usually disabled, and packets are routed around them until the links are repaired. While it is often possible to utilize some of a failing link's capacity, losing what remains of a link's capacity is typically deemed preferable to the erratic effect that unreliable links can have on application-level behavior.

We describe a new network function that relies on in-network computing to limit the erratic effect of failing network links, to enable the continued use of those links until they can be repaired. We explore the design space using ns-3, and evaluate our implementation on a physical test-bed that includes programmable switches and reconfigurable hardware. Our current hardware prototype can almost saturate a 10GbE link while using around 10% of our FPGA's resources.

CCS CONCEPTS

• **Networks** → **Programmable networks**; *Error detection and error correction*; Data center networks;

KEYWORDS

P4, FPGA, In-Network Computing, Fault Mitigation

ACM Reference Format:
Hans Giesen, Lei Shi, John Sonchack, Anirudh Chelluri, Nishanth Prabhu, Nik Sultana, Latha Kant, Anthony J McAuley, Alexander Poylisher, André DeHon, and Boon Thau Loo. 2018. In-network computing to the rescue of faulty links. In *Proceedings of ACM SIGCOMM 2018 Workshop on In-Network Computing conference, Budapest, Hungary, August 2018 (NetCompute'18),* 6 pages.
https://doi.org/10.475/123_4

1 INTRODUCTION

Advances in networking, hardware architecture, and programming languages in recent years have converged to make the choice between network performance and programmability easier: one can increasingly afford both. This has enabled research into hardware implementations of various applications that were previously confined to software for programmability or to expensive and inflexible ASICs for performance.

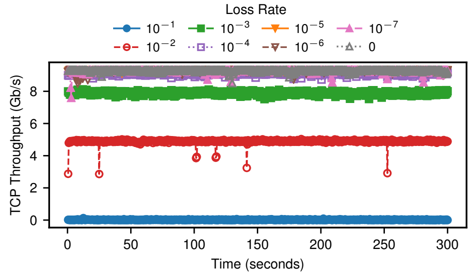

Figure 1: Intermittent packet drops at low rates, like those caused by faulty links, drastically reduce TCP throughput.

Most prior research in this area has adapted *existing* applications to run in hardware at line rate. Example applications include key-value stores [10], and regex matching on payloads [17].

In this paper we describe Wharf, which to our knowledge is the first application of its kind. Wharf is an in-network distributed mitigation of faulty links. Our design requires minimal configuration, is transparent to end-points and does not conflict with existing network architecture choices (e.g., the network's topology, and how packets are routed over it). We evaluate implementations for CPUs and FPGAs, and model a modified design using ns-3.

Wharf can work with various types of networks, but we think it can be especially helpful in mitigating failing links in datacenters, which has previously been studied by Zhuo et al. [18]. Datacenter networks are ever more expansive and their architecture involves a large number of links to attain a larger bisection bandwidth [15]. Their performance is critical to many widely-used applications, including those running on private and public clouds. Faulty links have a significant impact on these applications, as Figure 1 illustrates. The current practice of polling and disabling links from the edge of the network reacts slowly and adds more contention over non-failing links. Wharf can help make such networks more resilient to failing links, with minimal configuration overhead.

Running Wharf as an in-network function makes more sense than running it in end-hosts since the problem that Wharf mitigates

NetCompute'18, August 2018, Budapest, Hungary

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

is typically restricted to a fraction of the network. Applying Wharf to end-hosts adds overhead to all links, also the links that never fail. Just as some application features are best managed end-to-end [14], handling link failure seems best done at the link's level. Not only are we able to identify the faulty link directly, but, unlike higher-layer protocols, we can also distinguish link failure from congestion.

Programmable network hardware alone is not sufficient to solve this problem: one also needs a careful design since link management might interfere with other features of the network, such as topology and routing [7], transport [12] and load balancing [2]. In designing Wharf, we sought to make it transparent to other features of the network, to facilitate its interoperation with existing networks.

Our contributions include (i) a link-layer forward error-correction (FEC) scheme to mitigate failing links, and (ii) an FPGA implementation of this scheme that fully utilizes 10-Gbps links. Part of our implementation is in P4 [3] over SDNet. P4's limitation to header processing [5] was transcended by writing external functions in C for high-level synthesis. The key challenge we encountered was in streaming packets through our external functions.

The next section describes the background and related work of the problem we are solving, before we describe our design (§3) and implementation (§4), which we evaluate (§5) before concluding.

2 BACKGROUND

A communication network is architected to handle variation in its state arising from environmental interference, component failure, congestion and corruption of data. In this paper we describe a technique to enhance the reliability of networks with failing links.

Our approach could be compared to existing link-layer mitigations for unreliable transmission, such as network-wide hop-by-hop protection, as in X.25 [8], and link-layer retransmission as in the 802.11 [6] family of standards. The mitigation chosen for each system mostly depends on the transmission medium: X.25 was designed to work with unreliable links, whereas 802.11 uses a shared medium. Our thinking is similar, and our design is based on properties of the medium: we concentrate on wireline 10Gbps Ethernet; typically such links are reliable (i.e., have low error rates) and they are not shared (i.e., point-to-point). Instead of using an ARQ scheme as in X.25 and 802.11, we use forward error-correction (FEC). This simplifies our design, obviating the need for retransmission windows and this diminishes the memory needed for in-flight data since the sender will not attempt to resend a frame.

Our link-layer FEC complements the physical-layer FEC that is used in high-capacity Ethernet links: the physical-layer FEC helps the link sustain a given capacity over longer distances, whereas our FEC is intended to mitigate errors that do not arise because of challenging environmental factors; rather the errors arise because of physical damage to the link or failing transceivers [18]. Note that changing the PHY-layer FEC would deviate from the Ethernet standard, leading to loss of interoparability with COTS hardware.

Our approach also complements higher-layer reliability measures, as provided by TCP for example. TCP provides end-to-end reliability, whereas we concentrate on link-level reliability. Unlike TCP, we are able to distinguish congestion from corruption as the cause of packet loss, and we are able to locate the lossy links in the network. Thus we can react to them much more quickly than TCP at the end-points; we evaluate this in §5. We show that, as with using TCP alone, our mechanism results in a reduced transmission rate, but Wharf helps TCP over lossy links to improve its throughput under high loss rates. Experiments using the QUIC transport protocol suggested that the gains of end-to-end FEC did not outweigh the bandwidth overhead (even if $h = 1$), and adversely affected some kinds of traffic [9, §7.3]. In this paper we use FEC to cross a single link, rather than multiple links only one of which might be faulty.

In this paper we focus on Clos topologies [4], which are used in datacenters [15]. We believe that our design could be useful in mitigating faulty links in a Clos topology due to the large number links it uses when compared to a hierarchical topology [1], which exposes it more to link-related faults.

Centralized approaches have been described in the literature to mitigate faulty links in a WAN [11] and datacenters [18]. In comparison our approach is not centralized, and takes place in the network: a switch can activate FEC with adjacent switches over faulty links until the links are replaced or repaired.

3 DESIGN

Wharf infers failing links and follows a policy on how to process frames that are about to cross failing links. It uses a *forward error-correction* (FEC) scheme to enable the next hop to recover frames that were lost in transit by using extra parity frames that are inserted into the medium.

In this section we describe Wharf's policy choices for managing failing links, and how the chosen policy is followed in the network.

3.1 Traffic classification

Wharf is configured to have a number of traffic classes, which partition the frames arriving at a switch. Outbound frames are encapsulated and complemented with parity frames, forming *blocks* that are sent across the faulty link. Each class c is defined by a map $T : c \mapsto (k, h, t)$, where k is the number of frames in a block, h is the number of parity frames sent for each block, and t is the timeout. The values (k, h, t) influence the latency with which frames belonging to c cross the switch, as well the likely recovery of frames belonging to c.

3.2 Link-failure management policy

For each port the policy stipulates how to react if the link becomes faulty: frames are either dropped (i.e., the link is disabled), or they are processed to use FEC. For the latter case, the policy specifies what are the traffic classifications, and how each classification maps into parameters (k, h, t).

3.3 Execution

Wharf consists of three concurrent activities carried out for each port of a switch:

- **Link monitoring agent** attempts to infer link malfunction.
- **Sending proxy** processes frames and adds tags before they are sent over a faulty link.
- **Receiving proxy** processes inbound Wharf-tagged frames.

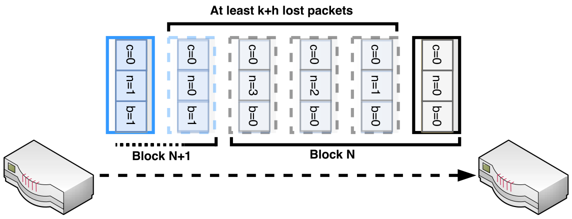

Figure 2: Packet numbering on lossy link between two switches. For low failure rates, assigning monotonically increasing frame numbers *n* that are unique within a block is sufficient to distinguish successive blocks. On links with bursty loss behavior, adding a block sequence number helps to distinguish packets. Note that frame reordering is not possible because the frames are being sent over the same link, and their processing is sequential across that link.

Frames to be sent over non-faulty links, and inbound frames that are not Wharf-tagged, are processed as normal by the switch. Otherwise outgoing frames are processed by the Sending proxy prior to egress, and inbound frames by the Receiving proxy after ingress.

Link monitoring agent. We continuously poll network port counters to infer malfunctioning transceiver modules or links. This is done as in CorrOpt [18], but failure-inference is done locally on the switch, rather than remotely. Sometimes a switch cannot itself realize if one of its links is malfunctioning, since the malfunction would be inferable from the adjacent element's counters (e.g., through an increase in frame errors). Thus switches might need to inform each other about errors on the transmitting side. To do this we employ LLDP and use a custom TLV to signal to the receiving switch that the link (for traffic travelling in the opposite direction) is failing.

Wharf frame encapsulation. As Fig. 2 demonstrates, frames sent over faulty links are tagged to allow the receiving switch to distinguish frames processed by Wharf and to provide the traffic classification and index of a packet within a block. The tag also includes a block identifier to protect against bursty losses.

Sending proxy. The encoding of a block – to produce parity frames – is triggered when the block is full (all k frames have been accounted for) or t for that c expires (relative to when the first frame was inserted into the block). Non-parity frames are tagged and forwarded immediately.

Due to its operation, the sending proxy can cause congestion at egress. For example, if $k = h$, and we are receiving traffic bound for a faulty link at rate R, then we would need to send at rate $2R$ at each interval of k incoming frames, at which time the sending proxy produces h additional frames for output. We simply drop new frames that cannot be put onto the link fast enough (i.e., before they are placed into a block). This loss will communicate to higher-layer protocols that congestion is occurring so the end-hosts' network stacks can react to this congestion as they normally would.

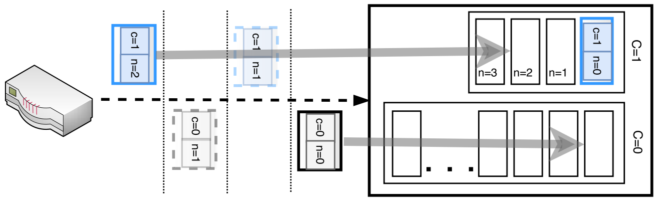

Figure 3: Prior to decoding a block on arrival, each block is mapped to the buffer for its traffic class. A dotted border indicates that a frame has been lost while transiting.

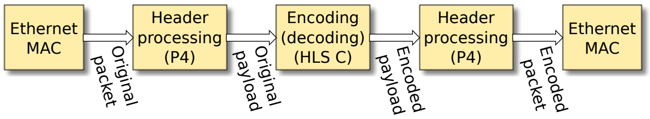

Figure 4: Block diagram of FEC encoder / decoder pipeline.

Receiving proxy. Wharf-tagged frames are buffered as shown in Fig. 3, and non-parity frames are untagged and forwarded immediately. For each c, if its t (relative to when the first frame was buffered) expires, or a frame from a successive block arrives, then decoding is triggered. Decoding consists of using the parity frames to reconstruct the lost data frames; if no data frames have been lost then there is no more work to be done for this block.

4 IMPLEMENTATION

4.1 Overview

We implemented Wharf on the Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit. This board has 4 SFP+ cages, and we process network traffic on its ZU9EG System-on-Chip FPGA, which consists of a quad-core ARM Cortex-A53, a dual-core ARM Cortex-R5, and a programmable fabric with 274K lookup tables (LUTs) and 1800 18Kb embedded memories (BRAMs).

A top-level diagram of the implementation is shown in Fig. 4. We implement header processing in P4 [3], which is compiled to FPGA logic using Xilinx's SDNet tool suite. Header processing (described further in §4.2) includes the frame tagging described in §3.

The frame is then forwarded to the FEC core, which can be an encoder (for the Sending Proxy) or decoder (for the Receiving Proxy). P4 is not suitable for exploiting the parallelism of the FEC core, so we implemented the core in hardware-synthesizable C. The code can be compiled and executed on a general-purpose CPU, but we can also synthesize the code for an FPGA in Vivado HLS. Guided by pragmas added by the developer, Vivado HLS takes advantage of parallelism in the code to achieve high performance.

The encoded or decoded output of the FEC core feeds into the header post-processing, which encapsulates the payloads in Ethernet frames, before packets are returned to the network.

4.2 Header processing

We use P4 to implement high-level packet processing logic and use Xilinx's P4-SDNet toolchain to translate our P4 program into RTL and integrate modules together. Our P4 program carries out packet

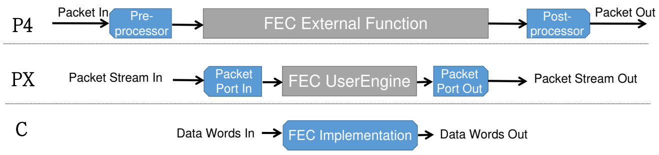

Figure 5: Abstraction provided by each layer; grey blocks are translated to the pipeline illustrated in next level.

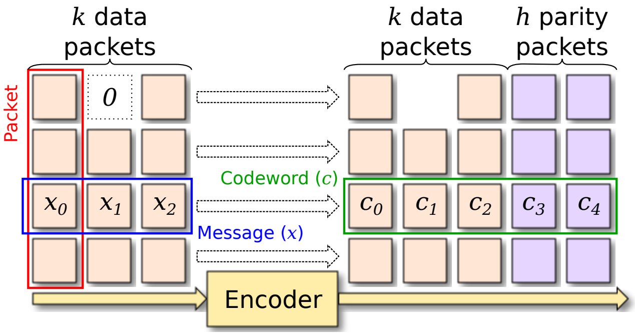

Figure 6: Correspondence between packet data, messages and codewords.

parsing, header tagging, payload extraction for the encoder, and book-keeping of packet classes.

Since we compute over whole frames – and not only headers – we could not write our entire system in P4. We wrote the header processing part of our system in P4, and called out from the P4 code to carry out the more general computing we need for FEC (§4.3), which we implemented in hardware-synthesizable C.

This crossing from P4 to more general logic emerged as a very important part of our implementation, since whole packets must flow through it between two different levels of abstractions. As is shown by Fig. 5, we used SDNet's extensible pipeline to implement our function as a stream processor with cut-through behavior and glue it to the header processor.

4.3 FEC Encoder

In this section we describe our generation of parity frames. Currently we can only decode on a CPU; our FPGA implementation of the decode is work-in-progress.

In Wharf, error correction is performed with a Reed-Solomon erasure (RSE) code [13]. Assuming we have a vector x, which contains a message with k symbols of m bits, encoding consists merely of multiplying x with a $k \times n$ generator matrix G. The result is an n-symbol codeword $c = xG$, where n satisfies $n = k + h$. The coefficients of the generator matrix are constants that are determined by the values of k and h. What makes RSE complex is that arithmetic operations are performed in a finite (Galois) field to ensure that mathematical operations on integers in a finite range yield again integers in the same range. In §3, we used k and h also to count packets. This choice was intentional as Figure 6 demonstrates. Due to space limitations, we will not discuss RSE in more detail.

The encoder could be implemented to work in two ways: non-incrementally on a block of packets, or incrementally as each packet arrives. The non-incremental approach is more straightforward, but incurs high latency and storage requirements, since encoding cannot start until all packets in a block have been received.

We opted for the incremental approach, which exploits the associativity of addition to rearrange the terms of the sums that form the matrix-vector multiplication. When a new input symbol is received, the partial sums for that symbol are calculated and the output symbols are updated.

We now derive the resource requirements in terms of multiplications. The matrix-vector multiplication requires $k \times n$ multiplications. The first k columns are an identity matrix because RSE does not alter data packets. The output values can be computed without multiplications. The remaining kh multiplications are performed once for each k-symbol message, resulting in h multiplications per input symbol. As a consequence, we expect that providing a higher level of protection against erasures for a given throughput constraint requires more hardware. The encoder receives packets from a 10-Gbps Ethernet MAC via a 64-bit AXI bus at a 156.25 Mhz line rate. For 8-bit symbols, 8 matrix-vector multiplier instances running at line rate would suffice to sustain 10 Gbps, so altogether $8h$ Galois-field multiplications are needed. To save resources, we use an implementation that computes the multiplication m of a and b with the formula $m = e^{\log a + \log b}$. The logarithms and exponents can be looked up in 8-bit tables with 256 entries. Such tables can be implemented in the local memories (BRAMs) of the FPGA. The coefficients of the generator matrix are constants. The associated logarithms can be precomputed, leaving only 2 lookup tables per multiplier, resulting in a grand total of $16h$ BRAMs.

Listing 1 shows the source code of the matrix-vector multiplier. It can be compiled with a regular software compiler and executed on a microprocessor for a software implementation or to debug the code. In addition, Vivado HLS can generate an FPGA implementation from the code, guided by the pragmas, which are ignored by software compilers. A similar implementation in Verilog or VHDL requires on the order of hundreds of lines. The `pipeline` pragma directs the tool to construct a hardware pipeline that can start executing one invocation of the function every clock cycle. Vivado automatically inserts a suitable number of registers to achieve the desired clock period. The `Par` and `Gen` arrays are mapped on BRAMs by default. The pipeline would need `MAX_H` values from each array per clock cycle although a dual-port BRAM can supply only 2 values. A solution is to divide the data over multiple BRAMs that can be accessed simultaneously. We accomplish that for `Gen` with the `ARRAY_PARTITION` pragma. To `Par`, we applied the `ARRAY_RESHAPE` pragma to combine all array elements into a single wide word.

```
void Enc(char Dat, char Par[MAX_H], int Pkt, int h) {
#pragma HLS ARRAY_RESHAPE variable=Par complete
  static char Gen[MAX_H][MAX_K] = { ... };
#pragma HLS ARRAY_PARTITION variable=Gen complete dim=0
#pragma HLS pipeline
  for (int i = 0; i < MAX_H; i++)
    Par[i] = i >= h ? Par[i] :
        GF_add(Par[i], GF_mul(Dat, Gen[i][Pkt]));
}
```

Listing 1: Matrix-Vector Multiplier

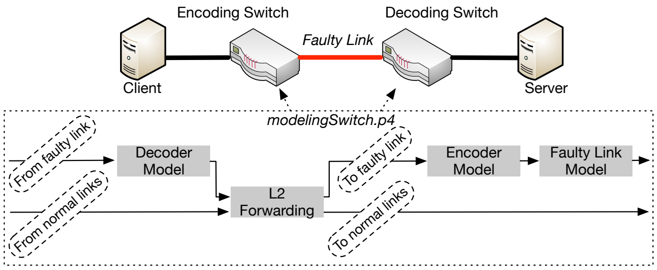

Figure 7: Topology of the 10 Gb/s testbed for real-time TCP benchmarks, using P4 to model FEC and faulty links.

5 EVALUATION

We evaluated Wharf with microbenchmarks and simulation.

5.1 Wharf Models

5.1.1 Line Rate P4 Model. To measure the effect of faulty links and Wharf at the protocol and application level, we developed a P4 pipeline for the Barefoot Tofino that models the behavior of faulty links and FEC in real-time. Figure 7 illustrates the pipeline.

The **encoder model** adds the Wharf header to packets egressing on the faulty link and generates blank parity packets.

The **faulty link model** adds a *corruption header* to each packet egressing the faulty link, indicating whether the next switch should consider the packet lost. It uses the Tofino's random number generator to select packets for loss according to a binomial distribution.

Finally, the **decoder model** processes packets ingressing from the faulty link. It removes added headers and passes non-corrupt packets to forwarding. It withholds corrupt packets, using recirculation, until the block ends. If it has counted at least K data plus parity packets in the block, the model "recovers" the corrupt packets by allowing them to pass to forwarding; else, the model drops them.

We deployed the model in the testbed network shown in Figure 7 and ran 10-second TCP file transfers from the client to the server using Iperf with different settings for k, h, and loss rate.

Figure 8 shows TCP throughput with different FEC configurations as loss rate varied. With Wharf, Iperf sustained over 5 Gb/s with loss up to 10^{-1} (1 out of every 10 packets dropped). Without Wharf, Iperf's throughput at that loss rate was under 25 Mb/s.

Figure 9 shows average congestion window size (*cwnd*) in the trials. Random packet loss at rates higher than 10^{-5} caused TCP to reduce *cwnd* significantly, resulting in the throughput drop in Figure 8. With FEC, *cwnd* remains high as loss rate increases, especially when using high levels of redundancy, e.g., $(k, h) = (5, 5)$.

Figure 10 shows end-to-end network latency. Latency increased with h/k because of the dynamics between the FEC model and TCP. The TCP source under-estimated its contribution to congestion because it was unaware of congestion related drops of parity packets. This caused high average queue depths in the egress to the faulty link, e.g., up to 1 MB for $(5, 5)$, and therefore high latency. A solution that requires no modification to TCP is rate limiting entry to the encoder based on the effective capacity of the faulty link, given h, k and the loss rate. This would drop data packets before parity packet generation to ensure that TCP senders are aware of all drops relating to their flows. We tested our hypothesis by repeating the $(5, 5)$ trials with the sending host limited to 4.5 Gb/s, just under effective capacity. The average latency was 41 μs, within the same range as latency in the no FEC trials. We plan to integrate rate limiting into the next versions of Wharf and the P4 models, using the metering features of P4.

(k, h)	(25, 1)	(25, 5)	(25, 10)	(50, 1)
BRAM (18Kb)	130 (7%)	183 (10%)	248 (14%)	130 (7%)
Flip-flop	49019 (9%)	49291 (9%)	49781 (9%)	49019 (9%)
LUT	29909 (11%)	30758 (11%)	31851 (12%)	29919 (11%)

Table 1: Resource requirements of different configurations of Wharf on the ZU9EG FPGA. BRAMs are local memories, and LUTs (lookup tables) are programmable gates.

5.1.2 Event-based simulation. We customized a fat-tree datacenter topology in ns-3 [16] to model (i) a link with loss characteristics as described by Zhuo et al. [18]; and (ii) FEC to support transport protocols. In this model we experimented with end-to-end error correction rather than at the link-layer, to simulate a more complex implementation without incurring the burden of implementing it fully.

We simulated a 128-node fat-tree network with 10Gbps links where two nodes communicate over TCP to transfer a 10MB file at 2Gbps. We found that using (5, 1) FEC completely eliminated retransmissions (which consisted of 152, 23, and 2 packets for loss rates of 10^{-3}, 10^{-4}, and 10^{-5} respectively). But achieving end-to-end reliability over a lossy link with little sacrifice to latency came at a steep end-to-end overhead of 20%. The approach described in this paper only adds overhead on lossy links, rather than across paths that contain a lossy link.

5.2 Encoder Microbenchmarks

We measured the performance of our current encoder implementation. Packets in a single flow with uniformly-distributed payload sizes of 64–1450 bytes are supplied to the FPGA with the packet generator of DPDK 17.08.1. The encoder processes the packet with parameters $k = 50$ and $h = 1$. At the output, we measured a throughput of 9.3 Gbps over a 10-minute period, nearly saturating the 10-Gbps link.

For comparison, we also evaluated a reference CPU implementation, not optimized for performance. In the same deployment, its throughput measured 227Mbps on a single core, and 1399Mbps using all 8 physical cores of our Xeon E5-2450L running at 1.8 GHz.

5.3 FPGA resource consumption

Table 1 shows the resource requirements for the FPGA implementations of Wharf with different k and h parameters. The resource requirements are post-implementation utilization values reported by Xilinx Vivado. We observe that varying k has a negligible effect on resource consumption, whereas BRAM consumption has a strong dependence on h. We believe that the BRAM consumption can be further reduced because several arrays were overpartitioned.

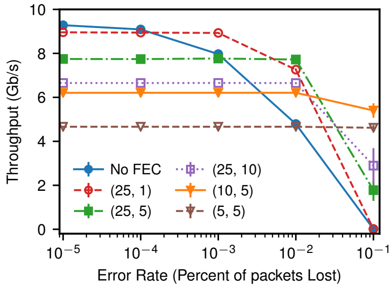

Figure 8: Iperf throughput.

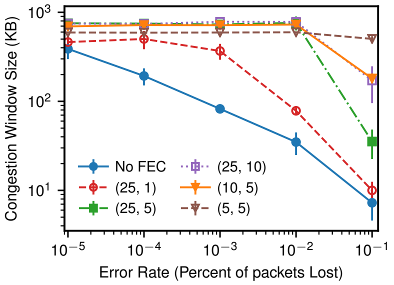

Figure 9: Iperf TCP window sizes.

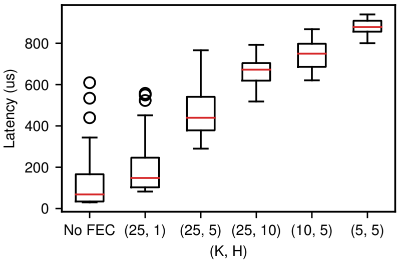

Figure 10: Latency at error rate=0. Boxes and whiskers show quartile and 1.5*quartile ranges.

6 CONCLUSION

In this paper we described Wharf, a new in-network mitigation against failing network links. It monitors network ports to infer abnormal loss, and uses forward error-correction across lossy links. Adjacent switches coordinate to set up complementary encoding and decoding stages for FEC across failing links. We prototyped the FEC encoding and decoding stages on x86 systems for ease of development and also prototyped the encoding stage to run on reconfigurable hardware.

Our implementation is a work in progress, and we are extending it to carry out the decoding stage on reconfigurable hardware, too. Another improvement consists of using all the network ports on our development board and scaling up our testing to use all available bandwidth. Finally, we will add the configuration and coordination logic to specify traffic classes and the parameters to the FEC.

We also plan to extend this work further to explore autonomous network resource management, going beyond in-network mitigation of faulty links. We will explore in-network services that require a level of programmability that exceeds P4's current capabilities, such as data deduplication and compression and build on the system described in this paper.

Acknowledgements

We thank Isaac Pedisich for programming support. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contracts No. HR001117C0047 and No. HR0011-16-C-0056. Xilinx provided tools and IP.

REFERENCES

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. *SIGCOMM Comput. Commun. Rev.* 38, 4 (Aug. 2008), 63–74. https://doi.org/10.1145/1402946.1402967

[2] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 503–514. https://doi.org/10.1145/2740070.2626316

[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[4] C Clos. 1953. A study of non-blocking switching networks. *Bell Systems Techical Journal* 32, 2 (March 1953), 406–424.

[5] Huynh Tu Dang, Han Wang, Theo Jepsen, Gordon Brebner, Changhoon Kim, Jennifer Rexford, Robert Soulé, and Hakim Weatherspoon. 2017. Whippersnapper: A P4 Language Benchmark Suite. In *Proceedings of the Symposium on SDN Research (SOSR '17)*. ACM, New York, NY, USA, 95–101. https://doi.org/10.1145/3050220.3050231

[6] The Working Group for WLAN Standards. 1997. IEEE 802.11TM WIRELESS LOCAL AREA NETWORKS. (1997). http://www.ieee802.org/11/ Accessed 8th March 2018.

[7] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. 2011. VL2: A Scalable and Flexible Data Center Network. *Commun. ACM* 54, 3 (March 2011), 95–104. https://doi.org/10.1145/1897852.1897877

[8] ITU-T. 1996. X.25 : Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit. (Oct. 1996). https://www.itu.int/rec/T-REC-X.25-199610-I/

[9] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. ACM, New York, NY, USA, 183–196. https://doi.org/10.1145/3098822.3098842

[10] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. 2017. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 137–152. https://doi.org/10.1145/3132747.3132756

[11] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. 2014. Traffic Engineering with Forward Fault Correction. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 527–538. https://doi.org/10.1145/2740070.2626314

[12] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. 2011. Improving Datacenter Performance and Robustness with Multipath TCP. *SIGCOMM Comput. Commun. Rev.* 41, 4 (Aug. 2011), 266–277. https://doi.org/10.1145/2043164.2018467

[13] Irving Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the Society of Industrial and Applied Mathematics* 8, 2 (06/1960 1960), 300–304. http://www.jstor.org/pss/2098968

[14] J. H. Saltzer, D. P. Reed, and D. D. Clark. 1984. End-to-end arguments in system design. *ACM Transactions on Computer Systems* 2 (1984), 277–288.

[15] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Hong Liu, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2016. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. *Commun. ACM* 59, 9 (Aug. 2016), 88–97. https://doi.org/10.1145/2975159

[16] D. Wong, K.T. Seow, C.H. Foh, and R. Kanagavelu. 2013. Towards Reproducible Performance Studies of Datacenter Network Architectures Using An Open-Source Simulation Approach. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM'13)*.

[17] Louis Woods, Jens Teubner, and Gustavo Alonso. 2010. Complex Event Detection at Wire Speed with FPGAs. *Proc. VLDB Endow.* 3, 1-2 (Sept. 2010), 660–669. https://doi.org/10.14778/1920841.1920926

[18] Danyang Zhuo, Monia Ghobadi, Ratul Mahajan, Klaus-Tycho Förster, Arvind Krishnamurthy, and Thomas Anderson. 2017. Understanding and Mitigating Packet Corruption in Data Center Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. ACM, New York, NY, USA, 362–375. https://doi.org/10.1145/3098822.3098849