SoC-based Implementation of a Lightweight Label Switching Router

Timothy Mark VanEtten†, Amy Charissa Williams†, Deng Jiahuan†, Feng Wang†, Lixin Gao*
† School of Engineering and Computational Science, Liberty University, Lynchburg, VA 24515, USA
* Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, USA

***Abstract*—The rapid proliferation of devices of the Internet of Things requires a vast network of heterogeneous devices to maintain the ability to communicate rapidly and seamlessly. The transition to delay sensitive Tactile Internet applications would demand on a new information and communications infrastructure. In our previous work, we have proposed a fast Location basEd Source Switching (LESS) mechanism with rerouting capabilities to ensure reliable throughput. In this paper, we present an SoC implementation of LESS and evaluate the performance and reliability of a LESS network. The evaluation results show that LESS can bring significant improvements in performance and reliability to sustain Tactile Internet.**

I. Introduction

One of the fastest growing areas of technology is the Internet of Things (IoT), a catch all term used to indicate the development of wirelessly connected smart devices and technologies, encompassing every current and future thing in a network of sensors, control systems, and information gathering. To allow large numbers of devices to communicate with each other simultaneously, extremely low end-to-end latency and high reliability are the essential characteristics for IoT. For example, various applications in the "Tactile Internet", such as automated vehicles, require the reaction time of the vehicle to be on the order of milliseconds. Furthermore, in the event of a node or link failure, packets must nonetheless be able to reach the intended destination within the specified amount of time.

However, IoT is the extension of the current Internet, which performs its best effort to deliver every packet. These Best Effort networks do not guarantee any bounds on delay and cannot achieve those stringent requirements. Therefore, the transition to delay sensitive Tactile Internet applications has demands on a new information and communications infrastructure. Many research work has been focused on reducing the Internet latency. For example, SDN technology, such as OpenFlow based switches [1], [2], relies on a central controller to improve the bandwidth efficiency. Centralized solutions has a scaling constraint due to the size of forwarding tables as the table size grows linearly with network size [3]–[5]. More importantly, it usually takes substantial time for a centralized controller to update switches to reroute around failures [6]–[8].

In our previous work, we have proposed a Location basEd Source Switching (LESS) mechanism with rerouting capabilities to ensure reliable throughput of the system. [9]. We have demonstrated that the proposed switching mechanism can be achieved at a minimal hardware cost. LESS eliminates the need for routing tables and integrates with minimum required functionality for packet forwarding. Our previous work has shown that LESS can enhance the network performance in terms of packet processing latency, throughput and reliability. In this paper, we present an SoC implementation of LESS and evaluate the performance and reliability of an SoC-based LESS router. The recovery time is less than 1 millisecond so that the dynamic protection method is suitable for time sensitive applications.

The remainder of this paper is organized as follows. Section II introduces the LESS system architecture and provides background. Section III describes the SoC-based implementation of the LESS router. Section IV presents the measurement results and data analysis. Finally, Section V concludes with some final remarks and further areas of study.

II. LESS Architecture

Overall, a LESS-enabled router forwards packets based on Label Switched Paths that specify output ports that packets should traverse. Since the output port at each hop is embedded in packet headers, LESS routers forward packets without regard to IP addresses and expensive IP lookup.

A. LESS Switching Labels

A LESS switch forwards packets based on the switching labels that are carried in a shim header between the data link layer and IPv4 layer. A LESS switching label contains a sequence of output ports in successive switches through which the packet is to be transmitted. The length of each LSP label is 32 bits, and more than one label can be encoded into a packet header. Fig.1 shows an example of the switching labels. We leverage a Multi-Protocol Label Switching (MPLS) encapsulation format to represent LSP labels. The first 24 bits encode 3 port numbers. The next bit (S bit) is a label stack bit, which indicates the last LSP label. If the stack bit is 1, it indicates that the LSP label is the last label, and an IP packet header is stored after the label. Otherwise, there is another LSP label before the IP packet. The next 7 bits are the Time To Live (TTL) field, which is decremented by 1 at each hop. If the TTL reaches 0, the LSP label is dropped. Note that our LSP label is different from Tag switching and MPLS. Both methods require a table lookup at each intermediate switch. In addition, tags and labels have meaning only per link, and have to be swapped at each switch. On the contrary, LESS LSP labels do not require a table lookup because the port numbers

978-0-9883-0453-6/17 $31.00 © 2017 ITC
DOI 10.1109/ITC.2017.26

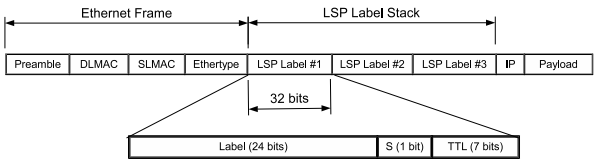


Fig. 1. LSP label format. The LSP label stack may contain more than one LSP label.

are local determined by those switches. The detail of LESS switching labels is presented in [9].

LESS employs an automatic addressing protocol to allocate addresses. A LESS address is a sequence of input and output ports in each of the succession of switches from a root switch to a switch or host. For example, in Fig. 2, the root switch would pass its address "`1:`" to switch 1 by pre-pending the downward port number "`1`". As a result, the address of switch 1 is "`1:1:`". After that, switch 1 could allocate a new address to switch 3 by prepending "`1:1:`" with the upward port number "`1`" and the downward port number "`2`". Thus, the address of switch 3 is "`1:1:1:2:`".

B. LESS Switching Operation

A LESS router performs a simple LSP label reading operation. The labels are pushed and popped on and off packets as they flow in a LESS router. In general, upon receiving a packet, a router examines the first byte of the top LSP label, and uses the label value directly as output port number. Then, the TTL value is decremented by 1. If the TTL value is zero, the top LSP label is discarded. Otherwise, the first byte is deleted from the top LSP by performing a left shift on the first 24 bits of the LSP label by one byte. Finally, the router sends the packet out that port. The operations are repeated for every switch through which the packet traverses. When a packet reaches its destination, the entire LSP labels have been removed from the packet so that the destination host only obtains an ordinary IP packet.

For instance, in Fig. 2, a packet with a switching label "`2.8.2`" is transmitted to switch 1, the first number of the label is extracted to determine the outgoing port (port 2) for the packet. Then, the number is deleted from the label, and the label becomes "`8.2`". Meanwhile, the switch decrements TTL value in the label. Finally, the switch sends the packet to the next switch 3, which has a physical link to the outgoing port 2. The process will continue until reaching the destination host. When the TTL value becomes 0, the whole switching label is deleted. Finally, the destination host receives a regular IP packet, which could be forwarded by IP routing. The detail of LESS switching operations is presented in [9].

III. SoC-based LESS Implementation

This section describes the FPGA implementation of a LESS router. We develop a LESS router prototype based on an Altera DE4 FPGA board (EP4SGX230KF40C2), which contains 4 Gigabit Ethernet ports, a PCI card, SRAM and DDR2 DRAM [10].

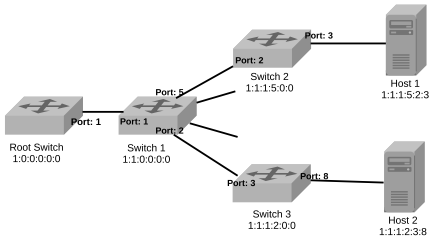


Fig. 2. An example of LESS label switching.

A. LESS Forwarding IP Core

Fig 3 shows the functional block diagram of the FPGA design. It consists of two main components: LESS forwarding IP core and SoC-based rerouting calculation. The LESS forwarding IP core is a custom IP core, consisting of an LSP parser and a packet rewriter. The LSP parser is used to derive the output port from an LSP label as a packet arrives at the input interface. The port number is then checked to see if the port is available. If it is, the LSP label header manipulation actions are performed at the packet rewriter. If the output port is not available, the packet is transmitted to a Nios II soft processor to derive a rerouting LSP.

A Triple Speed Ethernet (TSE) IP core provided by Intel FPGA is used to implement the Gigabit Ethernet functionality. The queue component of the switch receives the packets from the input ports and utilizes a first in, first out manner of managing the packet flow from the input ports to the LESS forwarding engine. The forwarding IP core contains the functionality for switch; it receives the packet from the queue, processes the label, then passes the packet to the switching fabric to send it to the next hop from the correct output port. The switching fabric receives the modified packet and forwards it out of the appropriate output port.

B. SoC-based Rerouting

Fault tolerance is an important factor for IoT. LESS provides a dynamic rerouting approach. A backup address is configured in advance and carried in packet headers. When a failure occurs, an alternative switch path is established based on the backup address. More, specificlly, when the switching fabric finds that the output port is unavailable, it forwards the packet to the Nios II processor embedded in the DE4 board to be processed for rerouting. The Nios II reroutes the packet based on the additional information provided in the header, then forwards it back to the queue to be reprocessed by the forwarding engine. The detail of LESS rerouting aglgrithm is presented in [9].

The SoC-based Rerouting module is shown in Fig.4. The Nios II fast core is selected to achieve the rerouting function. Due to small memory footprint of the rerouting calculation, we use on-chip RAM memory of the DE4 board. Two SGDMA IP cores are used for sending and receiving data. Every time the failed packets are written to the buffer, an interrupt for the Nios II is generated. Once the rerouting calculation module receives the interrupt, the contents of the buffer are read by the SGDMA TX and are passed to the rerouting module

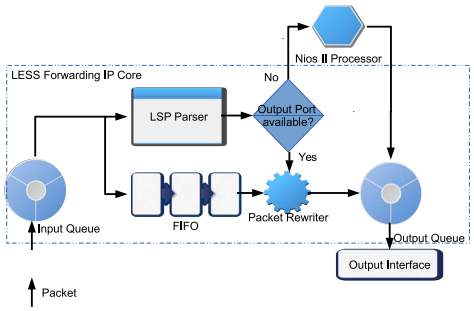


Fig. 3. LESS-enabled switch hardware implementation.

via the Avalon switch fabric. After the rerouting packets are calculated, they are transmitted from on-chip memory to the LESS forwarding module by the the SGDMA RX.

IV. Performance Evaluation

The testbed consists of three components: a traffic generator, a traffic sink and the SoC-based LESS router. The traffic generator ejects LESS packets to the SoC-based LESS router, and the traffic sink is responsible for collecting the LESS packets forwarded by the router. In this testbed, we measure the overhead of LESS packet processing and rerouting. To thoroughly test the system, the packet latency was tested under normal conditions and failure conditions, which include additional delay from the Nios II processing time.

A. Packet Processing Latency and Throughput

The LESS packet processing latency refers to the time it takes for a LESS packet to be processed by the SoC-based router. To measure the latency, we measure 1) the end-to-end packet latency with LESS forwarding (T_l), and 2) the end-to-end packet without LESS forwarding (T_d). The former represents the end-to-end latency involving LESS packets forwarding. We design a simple router by directly connecting the incoming port to the output port to measure the latter, which is the overhead associated on the computer side. The packet processing latency is calculated by the difference $T_l - T_d$.

We use a server as both a traffic generator and a traffic sink. The traffic generator generates LESS traffic towards the ports of the SoC-based LESS router. Tcpdump is utilized to monitor incoming and outgoing packets to record the injection timestamp and packet arrival timestamp. Enabling monitoring the outgoing and returning packets on the same machine reduces the difficulty of the time delta calculation. The captured packet logs include timestamps, with which the time delta can be easily calculated. When a LESS packet arrives at the sink, it calculates the time delta between the two timestamps.

We measure the packet processing time due to different rewrite actions, including label shifting and deletion. The packet latency is shown in Fig.5(a). In the figure, we present the packet latency for different packet sizes vs. different switching label sizes. We use 5 different frame sizes for the measurement, and for each frame size we repeat the measurement over 100 runs. It can be observed that the type of

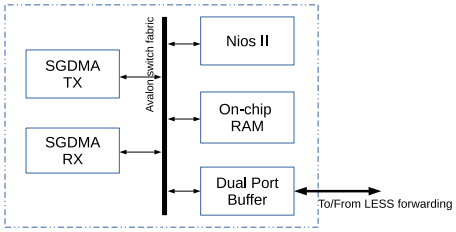
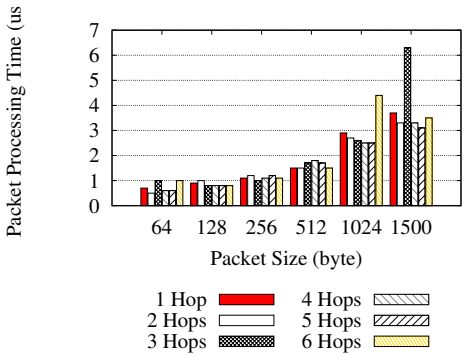
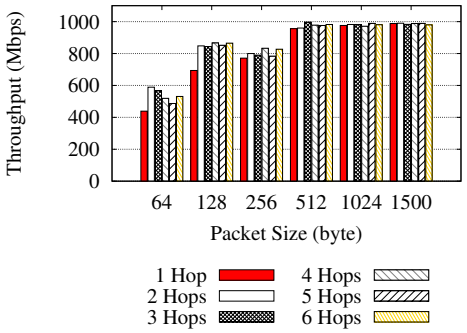


Fig. 4. SoC-based rerouting calculation module.



(a) Packet Processing Time



(b) Average Throughput

Fig. 5. LESS packet processing latency and average throughput.

LSP label headers has an impact on packet latency because of the cost of header manipulation. We observe that the latency increases proportionally to the size of labels. From Fig.5(a), we also find that the third type of LSP label header with 3 hops has the longest average latency because two bytes have to be shifted.

The throughput is measured by using a FPGA-based traffic generator, which can generate the maximum rate. Fig. 5(b) shows the throughput obtained for the LESS switch. The y-axis shows the amount of received frames in bytes per second. The figure shows that the LESS switch is capable of forwarding packets at a line-rate of 1Gbit/s.

B. Rerouting Latency and Throughput

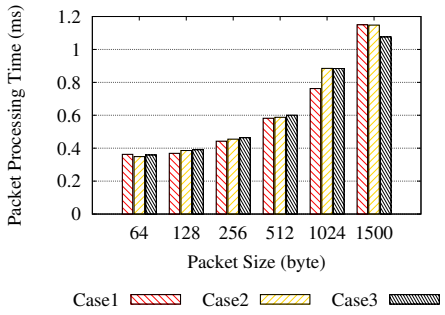
To evaluate the fault tolerance of LESS, we measure the overhead of constructing a new LSP when a link failure is

encountered in the SoC-based router. The packet generator injects packets with a failed port of the router. Based on its available addresses, the router performs different steps to construct the rerouting LSP.

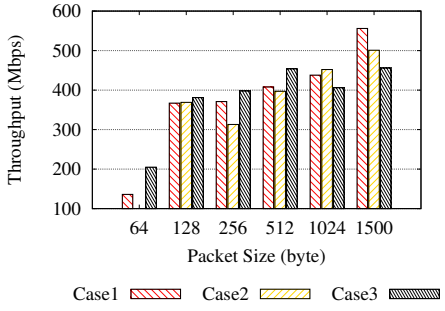
We consider three cases where the Nios II processor calculates the rerouting labels by different methods. The first cases accounts for a link failure at a node and the router has a local backup path to reach the destination. In the second case, the router does not have a backup path, but one of its neighbors has the path. The router reroutes the packets to the neighbor. In the third case, the router and its neighbors do not have any backup path. Therefore, the router uses backtracking to reroute the packets to previous visited routers that may have a backup path. The detail of LESS rerouting processing is presented in [9].

Fig. 6(a) shows the latency for the three cases vs. different packet size. In first case, the average rerouting latency is about 670 microseconds, 685 microseconds for the second case, and 650 microseconds for the last case. We find that the worst performance is at the first and second cases with large packets. We also find that the backtracked LSP label calculation (case 3) does not increase the overhead of the rerouting. Comparing the worst latency (about 685 us) with the convergence time of routing approach during a link failure (about 65 ms [11]), we find LESS can provide fast failover.

Next, we measure the throughput during a rerouting process. We use the SoC-based generator to generate different size flows, and measure throughput over 100 runs. The average throughput during the rerouting process is shown in Fig. 6(b). We can see that the throughput during rerouting degrades due to the bottleneck between the LESS forwarding engine and the Nios II processor. In our future work, we plan to use an ARM-based SoC implementation to improve the performance of LESS rerouting.



(a) Rerouting LSP Construction Time



(b) Throughput during rerouting

Fig. 6. Latency of rerouting LSP construction and throughput during rerouting.

V. Conclusion and Future Work

In this paper we have demonstrated the viability of a location based source switch as a scalable, reliable, high performance network switch with applications to the Future Internet and Internet of Things. Continued research will look to minimize the delay caused by looping of the Nios II processor in order to reroute packages that fail forwarding.

Acknowledgment

This work was supported by National Science Foundation grant CNS-1402857 and CNS-1402594 and under NSF-NICT Collaborative Research JUNO (Japan-U.S. Network Opportunity) Program.

References

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," NSDI'10, pp. 19–19, 2010.

[2] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," CoNEXT '11, pp. 8:1–8:12, 2011.

[3] W. Braun and M. Menth, "Wildcard Compression of Inter-Domain Routing Tables for OpenFlow-Based Software-Defined Networking," in *Proceedings of the 2014 Third European Workshop on Software Defined Networks*, EWSDN '14, pp. 25–30, 2014.

[4] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A Software Defined Internet Exchange," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 551–562, August 2014.

[5] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An Industrial-scale Software Defined Internet Exchange Point," in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pp. 1–14, 2016.

[6] M. Walraed-Sullivan, A. Vahdat, and K. Marzullo, "Aspen Trees: Balancing Data Center Fault Tolerance, Scalability and Cost," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pp. 85–96, 2013.

[7] K. Levchenko, G. M. Voelker, R. Paturi, and S. Savage, "Xl: An Efficient Network Routing Algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 15–26, August 2008.

[8] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring Connectivity via Data Plane Mechanisms," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pp. 113–126, 2013.

[9] F. Wang, L. Gao, S. Xiaozhe, H. Harai, and K. Fujikawa, "Towards Reliable and Lightweight Source Switching for Datacenter Networks," in *IEEE INFOCOM*, 2017.

[10] Altera, "Altera DE4 Development and Education Board." http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=501.

[11] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A Scalable Fault-tolerant Layer 2 Data Center Network Fabric," SIGCOMM '09, pp. 39–50, 2009.