TCP-Splitter: A TCP/IP Flow Monitor in Reconfigurable Hardware

David V. Schuehler John Lockwood
Applied Research Laboratory, Washington University
{dvs1, lockwood}@arl.wustl.edu

Abstract

TCP/IP is the most commonly-used protocol on the internet. It provides a reliable transport for nearly all applications that utilize a network. These include Web browsers, FTP, Telnet, Secure Shell and other applications. New types of routers require the examination of TCP/IP flows transiting this networking equipment.

This paper describes TCP-Splitter, a reconfigurable hardware based solution for analyzing and processing TCP/IP flows at multi-gigabit line rates. A consistent byte stream is delivered to a client application for every TCP/IP connection processed by TCP-Splitter. In order to maintain a design that is lightweight, efficient, and able to process a nearly unlimited number of flows at gigabit line rates, the system uses a non-passive flow processing algorithm.

1 Motivation

High speed network switches currently operate at OC-48 (2.5Gb/s) line rates, while faster OC-192 (10Gb/s) and OC-768 (40Gb/s) networks are on the horizon. At the same time, the number of traffic flows continues to increase [16].

Studies on the breakdown of internet traffic state that over 85% of the packets on the internet are Transmission Control Protocol (TCP) / Internet Protocol (IP) based [8][18]. New types of networking processing systems require scanning and processing of data both in headers and in payloads of TCP/IP packets. In order to scan payloads at high rates, new methods are needed to process TCP/IP data in hardware.

1.1 Related Work on TCP/IP Monitors

Protocol analyzers and packet capturing programs have been around as long as there have been networks and protocols to monitor. These tools provide the ability to capture and save network data with a wide range of capabilities.

Programs such as `tcpdump` allow for the capture and storage of TCP packets [13]. These tools work well for monitoring data at low bandwidth rates, but the performance of these programs is limited because they execute in software. Post processing is required with these tools in order to reconstruct TCP data streams.

`HTTPDUMP` was designed to capture and store web based HyperText Transfer Protocol (HTTP) traffic [17], yet due to the extra filtering logic for processing HTTP traffic, this tool requires more processing than `tcpdump` and runs slower.

`PacketScope` developed at AT&T is able to monitor much larger volumes of network traffic, but still relies on the capabilities of `tcpdump` to perform the packet capturing [1]. `BLT` leverages the `PacketScope` monitor to perform HTTP monitoring of links with line speeds greater than 100Mbps [4]. This tool does not ensure the processing of all packets, but instead attempts to obtain statistically relevant results by capturing a high percentage of the HTTP traffic.

The Internet Protocol Scanning Engine is another software based TCP/IP monitor [5]. Typically, only header information is captured and written to a log file for TCP stream content. This program also has performance limitations which preclude it from monitoring high bandwidth traffic.

The Cluster-based Online Monitoring System does a much better job of capturing data associated with web requests [12]. Performance is improved over other systems by having multiple analysis engines working in parallel, yet even running a system with eight analysis engines, traffic on a 100Mbps network was not consistently monitored.

None of these solutions are able to operate in a high speed active networking environment where data rates exceed 1 Gbps, nor can they guarantee the processing of every byte of data on the network.

A TCP state tracking engine with buffer reassembly has been developed by Necker *et al* [14]. The focus of this research is on intrusion detection and tracking the TCP/IP processing state of a single connection. Limited buffer reassembly is also performed. This solution utilizes a similar hardware environment and is able to process data at line rates equal to that of TCP-Splitter. By instantiating multiple processing circuits, a maximum of 30 TCP/IP con-

nections can be monitored simultaneously on a single Field Programmable Gate Array (FPGA).

1.2 Hardware-Based TCP/IP Processing

A hardware implementation of a full TCP/IP protocol stack acting as a communication endpoint would be useful. Unfortunately, there are issues that make a full implementation of a TCP/IP stack in hardware impractical. These issues include: (1) the need for a multitude of TCP timers, (2) the need for large memories for reassembly buffers, and (3) the need to support a large number of connections.

A circuit has been developed which supports TCP flow monitoring. Instead of acting as a connection endpoint for a few TCP connections, TCP-Splitter monitors all TCP flows passing through the network hardware. There are many advantages to this technique over implementing a TCP protocol endpoint.

In order to provide the reliable delivery of a data set into a client application, a TCP connection only needs to be established that transits through the device monitoring the data. The bulk of the work for guaranteed delivery is managed by the TCP endpoints, not by the logic on the network hardware. This eliminates the need for a complex protocol stack within the reconfigurable hardware because the retransmission logic remains at the connection endpoints, not in the active network switch.

2 Background

The current generation of Field Programmable Gate Arrays (FPGAs) have approximately the equivalent capacity of a million-gate ASIC, a few hundred kilobytes of on-chip memory, and operate at speeds ranging from 50MHz to 200MHz. FPGAs can be used to implement network processing functions by placing them in the data path of a high-speed network switch.

The Applied Research Laboratory (ARL) at Washington University has developed the Washington University Gigabit Switch (WUGS) as a research platform for high speed networks [19]. This hardware, along with the Field-programmable Port Extender (FPX), is used as the test bed for this project [11].

Components of the Layered Protocol Wrappers [3] that had been developed for the FPX have been used as a foundation for this research. These wrappers were developed to process high level packets in reprogrammable logic. The wrappers include an ATM Cell Wrapper, an AAL5 Frame Wrapper, an IP Wrapper and a UDP Wrapper. This set of wrappers allows a client application to send and receive packets with FPGA hardware. Utilizing the cell, frame, and IP wrappers from this research, a TCP flow monitor called TCP-Splitter has been implemented. The TCP-Splitter name comes from the idea that the TCP byte stream is split into two separate directions as shown in figure 1. One flow is delivered to the client application on a local host while the other is forwarded to the remote endpoint.

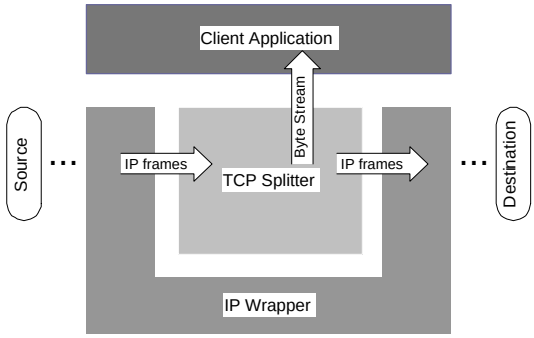

Figure 1. TCP-Splitter data flow

3 Design Requirements

The TCP-Splitter was designed to be a lightweight, high performance circuit that contains a simple client interface which can monitor a nearly unlimited number of flows. Design tradeoffs were made in order to achieve these results within the practical bounds of today's hardware. Some of the issues faced in the development of the TCP-Splitter were: (1) dealing with dropped frames, (2) handling reordered packets, (3) maintaining state for a large number of flows, (4) processing data at line rates, and (5) minimizing hardware gate count. In order to overcome these challenges, this implementation places some restrictions on how data flows through the network switch.

TCP-Splitter requires placement in the network where all packets of monitored flows will pass. All packets associated with a TCP/IP connection being monitored must pass through the networking node where monitoring is taking place. It would otherwise be impossible to provide a client application with a consistent TCP byte stream from a connection if the switch performing the monitoring only processed a fraction of the TCP packets. In general, this requirement is true at the edge routers but not true for interior nodes of the internet. This requirement can also be enforced in private networks where the network has been designed to pass traffic in a certain manner.

4 A Non-Passive Solution

A problem with attempting to monitor a large number of TCP/IP flows is that such a system would require a large amount of memory. The maximum window scale factor

supported by TCP is 2^{14} which leads to a maximum window size of 1 GByte [9]. In a worst case scenario, this amount of memory would be required to reassemble packets in each direction of a TCP connection. A high speed switch monitoring both directions of 128k connections would require 256 TeraBytes of high speed random access memory (RAM). Even if one assumes that TCP window size is limited to 1MByte, the system still requires 256Gbytes of memory to monitor both directions of the same 128k connections. This large quantity of memory is prohibitive and led to the consideration of other lightweight designs.

The need for reassembly buffers can be eliminated if all frames for a particular flow transit the switch in order. Because there is no guarantee that TCP frames will traverse the network in order, some action will have to take place when packets are out of order. By actively dropping out of order packets, a TCP byte stream can be generated for the client application without requiring reassembly buffers. If a missing packet is detected, subsequent packets are actively dropped until the missing packet is retransmitted. This ensures in-order packet flow through the switch.

This design feature will force the TCP connections into a Go-Back-N sliding window mode when a packet is dropped upstream of the monitoring node. As it turns out, the Go-Back-N retransmission policy is widely used on machines throughout the internet. Many implementations of TCP, including that of Windows 98, FreeBSD 4.1, and Linux 2.4 [7], use the Go-Back-N retransmission logic. The benefit or detriment on the throughput is dependent on the specific TCP implementations being utilized at the endpoints. In instances where the receiving TCP stack is performing Go-Back-N sliding window behavior, the active dropping of frames may actually improve overall network throughput by eliminating packets that will be discarded by the receiver. In situations where the endpoints are utilizing selective retransmission and there is a high percentage of data loss on the network, the operation of TCP-Splitter has the potential to exacerbate the dropped packet problem and render the connection unusable.

5 Architecture

A high level view of the data flow through TCP-Splitter can be seen in figure 1. TCP-Splitter has been implemented in FPGA hardware and fits within the existing FPX wrapper framework. IP frames enter into the TCP-Splitter from the IP protocol layer contained in the IPWrapper [3].

The inbound frames enter the TCP-Splitter and are classified, checksummed, and cached. Outbound IP frames are delivered back to the IPWrapper to be sent on to the next hop router. A TCP byte stream is also delivered to the client application for each TCP flow in the network.

The TCP-Splitter consists of two logical sections. The first is called TCP-Input which handles the ingress of IP frames. The majority of the processing in the TCP-Splitter is performed by this section. The second section is called TCP-Output and is responsible for packet routing and the delivery of frames to the outbound IP stack as well as to the client application.

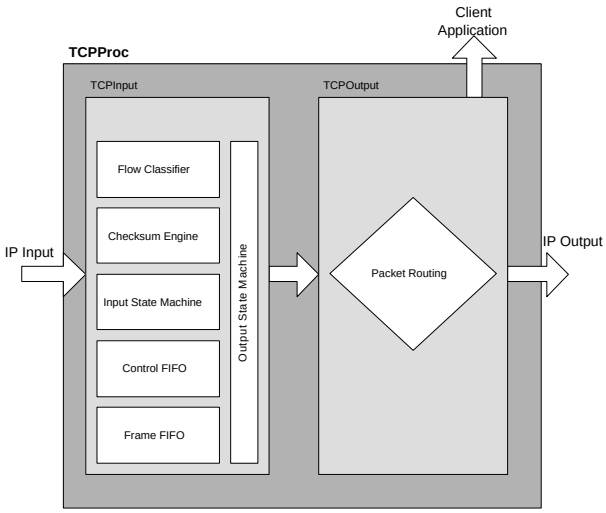

Figure 2. TCP-Splitter layout

5.1 Input Processing

Input frame processing consists of 6 different components as seen in figure 2. The Flow Classifier, the Checksum Engine, the Input State Machine, the Control FIFO and the Frame FIFO all process IP packet data received from the IP protocol wrapper. The Output State Machine is responsible for clocking data out of the control and frame FIFOs and into the output processing section.

IP frames are clocked into the input section 32 bits at a time. The TCP checksum is computed utilizing the appropriate bits in each data word. The input data is also stored in the Frame FIFO so that the the TCP checksum result can be delivered to the output section along with the start of the IP packet.

Once the TCP checksum has been computed, information about the current frame is written to the Control FIFO. This data includes the checksum result (pass or fail), the flow identifier, start and end of flow signals, a TCP frame indicator, and a signal to indicate whether or not the frame should be forwarded. The Control FIFO is required in order to hold state information of smaller frames while preceding larger frames are still being clocked out of the Frame FIFO for outbound processing.

The Output State Machine is responsible for clocking data out of the Control and Frame FIFOs and into the out-

put processing section of the TCP-Splitter. Upon detecting a non-empty Control FIFO, the output state machine starts clocking the next frame out of the Frame FIFO. This frame data along with the control signals from the Control FIFO exit the TCP-Input section.

5.2 Flow Classification

A simple flow classifier was designed for the TCP-Splitter which can operate at high speed and has minimal hardware complexity. The flow table is a 256k element array contained in a low latency static RAM chip. Each entry in the table contains 33 bits of state information. An 18 bit hash of the source IP address, destination IP address, source TCP port, and destination TCP port is used as the index into the flow table. The detection of a TCP FIN flag signals the end of a TCP flow and the hash table entry for that particular flow is cleared. Hash table collisions in this flow classifier are not handled at present and cause packets from different flows to be processed as if they were a single connection.

Other classifiers could be used to identify traffic flows for TCP-Splitter. `SWITCHGEN` is a tool which transforms packet classification rules into reconfigurable hardware based circuit design [10]. The Recursive Flow Classification (RFC) algorithm is another high performance classification technique which optimizes rules by removing redundancy [6]. Both of these research projects are developing flow classifiers to perform 30M to 100M classifications per second. Prakash *et al.* have proposed a packet classification solution which performs lookups utilizing a series of pipelined SRAMs[15]. One billion packet classification lookups per second could be supported with this technology. The design of the TCP-Splitter does not impose any restrictions on the flow classification technique utilized and could be used with any flow classifier.

5.3 Output Processing

The output processing section of the TCP-Splitter is responsible for determining how a packet should be processed. There are three possible choices. Packets can be (1) passed on to the outbound IP layer only, (2) passed both to the outbound IP layer and to the client application, or (3) discarded.

The rules for processing packets are as follows:

- All non TCP packets are sent to the outbound IP stack.
- All TCP packets with invalid checksums are dropped.
- All TCP packets with sequence numbers less than or equal to the current expected sequence number, are sent to the outbound IP stack.
- All TCP packets with sequence numbers greater than the current expected sequence number are dropped.
- All SYN packets are sent to the outbound IP stack.
- All other packets are forwarded both to the outbound IP stack and the client application.

5.4 Client Interface

The client interface provides a simple hardware interface for application circuits. Only data that is valid, checksummed, and in-order for each specific flow is copied to the client application. This allows the client to solely process the consistent stream of bytes from the TCP connection. All packet headers are clocked into the client application along with a start-of-header signal so that the client can extract information from these headers. This method eliminates the need to store header information, but still allows the client access to this data.

The client application does not sit in the network data path and therefore does not induce any delay into the packets traversing the network switch. This allows the client application to have arbitrary complexity without affecting the throughput rate of the splitter.

6 Results

TCP-Splitter has been implemented as a module on a Xilinx FPGA. The circuit has been synthesized to operate at 101 MHz to in order to provide processing at full OC-48 line-speeds on the FPX platform. The critical path in the design involves the 16-bit arithmetic operations used to compute the TCP checksum. The TCP-Splitter implementation is small – it utilizes only 2% of a Xilinx XCV1000E FPGA. A complete solution that includes the TCP-Splitter, Internet Protocol Wrappers, and a sample client application that simply counts bytes TCP data bytes requires a total of 21% of the FGPA resources. TCP-Splitter has a pipeline delay of only 7 clock cycles, which introduces a total delay to the datapath of 70 ns. In order to avoid forwarding errored frames, TCP-Splitter adds one store-and-forward delay to allow for the time of computing and verifying the TCP checksum.

7 Future Work

Performance improvements to increase the throughput rate of the TCP-Splitter are planned with the intent of supporting OC-768 line rates. In order to accomplish this goal, additional pipeline stages and parallelism available within the FPGA will be exploited.

In the current implementation, classifier performs a maximum of 2 memory accesses for each packet. Given that the TCP-Splitter input data length is 32 bits (4 bytes), and assuming minimum length packets of 64 bytes, the smallest period of operation is 16 clock cycles. Within this time, 8

TCP-Splitter engines could be implemented to run in parallel and perform one memory access on every clock cycle. By utilizing both of the static RAM modules on the FPX platform, a solution could be designed with 16 TCP-Splitter engines, each operating at 101MHz, which would process data at 51Gbps. This is sufficient bandwidth to monitor all TCP/IP flows at OC-768 line rates.

Another planned enhancement is the addition of a small number of packet reassembly buffers. These buffers would support the reassembly of IP fragments and TCP packets in order to provide a passive monitoring solution.

Improvements to the flow classifier are also planned in order to eliminate the hash table collision problem.

8 Conclusions

A TCP/IP network monitoring component, called TCP-Splitter, has been developed to provide client application systems with a consistent TCP data stream. This research differentiates itself from other related works in the following ways:

- It is implemented in reconfigurable hardware.
- It processes packets at line rates exceeding 3 Gbps.
- It is capable of monitoring 256k TCP flows simultaneously.
- It delivers a consistent byte stream for each TCP flow to a client application.
- It processes data in real time.
- It eliminates the need for large reassembly buffers.

TCP-Splitter has been synthesized for a Xilinx Virtex 1000E-7 FPGA and has a post place-and-route frequency of 101MHz and a corresponding throughput of 3.2 Gigabits per Second. A sample application has been successfully tested in hardware utilizing simulated TCP data packets. The circuit was developed as a module on the FPX platform, but could be easily ported to other FPGA-based packet processing systems.

References

[1] N. Anerousis, R. Caceres, N. Duffield, A. Feldmann, A. Greenberg, C. Kalmanek, P. Mishra, K. Ramakrishnan, and J. Rexford. Using the AT&T Labs PacketScope for Internet Measurement, Design, and Performance Analysis. http://citeseer.nj.nec.com/477885.html.

[2] F. Baboescu and G. Varghese. Scalable packet classification. In *ACM Sigcomm*, August 2001.

[3] F. Braun, J. W. Lockwood, and M. Waldvogel. Layered protocol wrappers for internet packet processing in reconfigurable hardware. In *Proceedings of Symposium on High Performance Interconnects (HotI'01)*, pages 93–98, Stanford, CA, USA, Aug. 2001.

[4] A. Feldmann. BLT: Bi-Layer Tracing of HTTP and TCP/IP. *WWW9 / Computer Networks*, 33(1-6):321–335, 2000.

[5] I. Goldberg. Internet Protocol Scanning Engine. http://www.cs.berkeley.edu/ iang/isaac/ipse.html.

[6] P. Gupta and N. McKeown. Packet Classification on Multiple Fields. In *ACM Sigcomm*, 1999.

[7] A. Gurtov. Effect of delays on tcp performance. In *Proceedings of IFIP Personal Wireless Communications '2001*, Aug 2001.

[8] IETF. RFC793: Transmission Control Protocol. http://www.faqs.org/rfcs/rfc793.html, Sep 1981.

[9] V. Jacobson and R. Braden. RFC1072: TCP Extensions for Long-Delay Paths. http://www.faqs.org/rfcs/rfc1072.html, Oct 1988.

[10] A. Johnson and K. Mackenzie. Pattern Matching in Reconfigurable Logic for Packet Classification. In *ACM CASES*, 2001.

[11] J. W. Lockwood. An open platform for development of network processing modules in reprogrammable hardware. In *IEC DesignCon'01*, pages WB–19, Santa Clara, CA, Jan. 2001.

[12] Y. Mao, K. Chen, D. Wang, and W. Zheng. Cluster-based online monitoring system of web traffic. In *Proceeding of the Third International Workshop on Web Information and Data Management*, pages 47–53. ACM Press, 2001.

[13] S. McCanne, C. Leres, and V. Jacobson. tcpdump. ftp://ftp.ee.lbl.gov, 1998.

[14] M. Necker, D. Contis, and D. Schimmel. TCP-Stream Reassembly and State Tracking in Hardware. FCCM 2002 Poster, Apr 2002.

[15] A. Prakash and A. Aziz. OC-3072 Packet Classification Using BDDs and Pipelined SRAMs. In *Proceedings of Symposium on High Performance Interconnects (HotI'01)*, pages 15–20, Stanford, CA, USA, Aug. 2001.

[16] L. Roberts. Internet still growing dramatically says internet founder. http://caspiannetworks.com/pressroom/press/08.15.01.shtml, Aug 2001.

[17] P. B. Roland Wooster, Stephen Williams. HTTPDUMP Network HTTP Packet Snooper. http://citeseer.nj.nec.com/332269.html, Apr 1996.

[18] S. Shalunov and B. Teitelbaum. Bulk tcp use and performance on internet2. http://www.internet2.edu/abilene/tcp/i2-tcp.pdf, Aug 2001.

[19] J. Turner, T. Chaney, A. Fingerhut, and M. Flucke. Design of a Gigabit ATM Switch. In *In Proceedings of Infocom 97*, Mar. 1997.

[20] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable high-speed prefix matching. *ACM Transactions on Computer Systems*, 19(4), Nov. 2001.