Optimizing Packet Lookup in Time and Space on FPGA

Thilan Ganegedara, Viktor Prasanna
Ming Hsieh Dept. of Electrical Engineering
University of Southern California
Los Angeles, CA 90089
Email: {ganegeda, prasanna}@usc.edu

Gordon Brebner
Xilinx Labs
2100 Logic Drive
San Jose, CA 95124
Email: gordon.brebner@xilinx.com

***Abstract*—The evolution of the Internet has transformed the simplistic Ethernet/IP based packet forwarding into a complex collection of lookup schemes. Depending on the location of the networking equipment (core, provider/customer edge, etc.,) a router/switch may potentially have to support several such complex lookup schemes. However, the hardware resources allocated to perform such operations are limited, especially in single chip implementations. In this paper, we propose techniques to map such complex lookup schemes on to hardware platforms under a limited resource budget and produce a design for a pipelined packet lookup architecture. An Integer Linear Programming (ILP) based technique is introduced to optimally allocate the limited hardware resources for a single lookup scheme. We extend our solution to multiple lookup schemes by proposing techniques to improve resource sharing, which results in a resource planning tool. Field Programmable Gate Array (FPGA) - a natural choice for high-speed packet processing applications - is used as the target platform. By using the proposed techniques, we show that up to 4 complex lookup schemes can be hosted on a single FPGA consuming only 20 Mbit on-chip memory and 750 pins for external memory communication.**

I. Introduction

With the advent of shim layers such as Multi-Protocol Label Switching (MPLS), 802.1Q, etc., [9], [1], [18], [17] modern routers and switches are required to perform complex sequences of packet lookup operations. These sequences of lookup operations, hereafter referred to as *lookup schemes*, implement different packet forwarding technologies and protocols which assure security, functionality, performance, etc., of a network. A single such lookup scheme may contain multiple individual field lookups. To exacerbate the process, these complex lookup schemes may contain non-trivial dependencies between field lookups (e.g. IP version check, MPLS/Ethernet). Depending on the lookup schemes adopted in a particular networking environment and the level in the network hierarchy at which the router/switch operates (e.g. core, provider/customer, etc.,), a packet may have to go through multiple lookup operations before making a forwarding decision. Further, the lookup operations performed on a given packet depends on the header information available.

A network equipment manufacturer is interested in making their products versatile so that a single switch/router supports various packet forwarding technologies (e.g. Q-in-Q [4], MAC-in-MAC [16], Ethernet bridging, etc.,). While adding more lookup functionality is an important goal, an equally important goal is providing these diverse packet forwarding functionality on a single chip. This is challenging since the amount of hardware resources available on a chip is limited [12]. Therefore, careful resource planning is critically needed in order to exploit the limited on-chip resources in such a way that multiple lookup operations can be accommodated on a single chip. This leads to economical, yet versatile packet forwarding solutions.

While other hardware platforms exist, Field Programmable Gate Arrays (FPGAs) have become the natural choice for high-speed networking applications due to various reasons such as abundant resources (memory, logic, etc.,), reconfigurability, high-performance, etc., [13], [20], [6], [3], [11], [2]. Various algorithmic solutions can be mapped onto FPGA, and pipelining is often employed to achieve high-performance. In addition, by exploiting the vast parallelism offered on these devices, performance can be further improved. Due to these reasons, in this work we use FPGA as the target hardware platform. However, note that the solution proposed in this work can be seamlessly adopted in other platforms as well.

To the best of our knowledge, there is no existing solution for organizing multiple packet lookup schemes on hardware, other than manual identification of a potential solution. With increasing number of lookup schemes, variety of search techniques, and storage locations (on-chip/off-chip), hand/manual optimization becomes infeasible and may not guarantee optimal allocation. As a remedy, we propose an Integer Linear Programming (ILP) [14] tool for hardware resource planning in network applications that maps multiple lookup schemes on to hardware platforms under a given hardware budget by improving resource sharing. To summarize, we make the following contributions in this paper:

- Abstraction of the problem from the network, design and architecture perspectives: We model the resource usage and latency of various table lookup techniques, based on the high-level details of the lookup schemes.
- Identification of the optimal lookup strategy for a given scheme: A modified version of Branch-and-Bound algorithm [15] is used for this purpose to resolve the non-trivial dependencies between field lookups.
- Extension to multiple lookup schemes: A heuristic ap-

proach to identify and improve potential resource sharing for multiple lookup schemes.

The combination of the above three contributions forms the resource planning tool, which produces the design for a static pipelined lookup architecture to perform packet lookup, under limited hardware resource budgets.

II. Background

Wire-speed packet forwarding has become a prominent concern with the dramatic growth of the Internet [6], [3], [11], [1]. While at the early stages of the Internet packet forwarding used to be simple Ethernet/IP inspection, since the debut of Virtual Local Area Network (VLAN) and MPLS technologies, packet forwarding has become a complex operation [8]. This is further complicated by the existence of different versions of protocols such as IPv4 and IPv6 [5].

In order to get a sense of the types of lookups a router/switch has to perform, we list a set of widely used packet lookup schemes here. Note that several custom schemes can be found depending on various network requirements. We list only the prominent techniques due to space limitations.

- MAC-in-MAC
- Q-in-Q
- MPLS forwarding
- IP lookup - IPv4, IPv6
- Multi-field packet classification

Depending on whether the router/switch operates at edge/core level, a networking device may have to implement multiple of these techniques to correctly forward packets incident on its network ports. Such implementations store lookup tables that contain forwarding information using a chosen data structure. The lookup table is scanned through to identify the matching entry and the corresponding forwarding information is used either for further processing or to actually forward the packet. The most widely used data structures to perform such lookup operations are:

- Binary/Ternary Content Addressable Memory (BCAM/TCAM)
- Hash table
- Binary Search Tree (BST) or B-tree
- Uni-bit/multi-bit trie

These techniques can be implemented on FPGA using the available logic resources and on-chip/off-chip memory. Current research in packet forwarding mainly focuses on implementing MPLS/IP forwarding and multi-field packet classification. While they are noteworthy problems, accommodating multiple packet lookup schemes on a single chip is even more challenging due to the hardware resource constraints imposed by the target platform and the resource requirements of lookup schemes. Further, organizing individual packet lookup operations under such constraints leads to an interesting optimization problem that deals with various aspects of networking as well as lookup engine design.

III. Multi-field Packet Lookup on FPGA

The main goal of this research is to optimize the resource usage of modern router/switch hardware to accommodate multiple packet lookup schemes in a shared lookup engine. In doing so, we add versatility and cost effectiveness to network equipment by identifying the optimal organization of different header field lookups under a limited hardware budget.

A. Metrics and Assumptions

The effectiveness of a packet lookup engine is often characterized by 1) resource efficiency, 2) throughput, and 3) latency. In the context of FPGA, a resource can be on-chip memory, logic resource or external input/output (I/O) pins. With the advancements in FPGA technology, achieving 40+ Gbps per pipeline packet forwarding rates using existing search techniques (e.g. hash, CAM, tree, trie) has become commonplace [7], [10], [2], [3], [13]. Therefore, we mainly target resource efficiency and latency as the performance evaluation metrics. Even though resource efficiency can be measured in many ways, in this work we compute it as the number of field lookups that can be performed with the provided resources, mainly on-chip memory and external I/O pins.

We make the following assumptions in this work:

Assumption 1: Target packet lookup architecture is a linear pipelined engine. An incoming packet traverses through all the pipeline stages despite the lookup scheme a packet performs lookup in.

Note that we make Assumption 1 for the sake of simplicity. More complex, yet interesting, architectures can be designed by allowing packets to exit the pipeline when header processing is complete, instead of traversing all the stages of the pipeline. Each pipeline stage performs a lookup algorithm action rather than an individual packet lookup. To complete a single packet lookup, traversing multiple stages may be required (see Section III-B3). Under this assumption, every packet experiences a latency of *number of pipeline stages* $\times$ *clock period.*

Assumption 2: Packet header extraction can be performed at wire-speed and does not require complex processing, hence is integrated as a part of a lookup stage.

In [9] and [1] packet header parsing is performed at 40G and 400G+ rates and the complexity of header extracting is low compared with packet lookup operations.

B. Understanding the Context

Before defining the problem, to help understand it better, we provide some insight into the details. The considered problem can be described from four perspectives and we discuss them in the following sections.

1) Network Perspective: A network administrator or an Internet Service Provider (ISP) specifies the technologies and functionality that a particular network ought to have in order to facilitate correct, efficient, and secure operation of a network. In order to achieve this, the network equipment has to have the capability to process the packets in the desired fashion. As mentioned earlier, depending on various factors, a switch/router may have to perform different sequences of lookup operations, which we refer to as lookup schemes. These lookup schemes can be represented as a graph, which describes the order in which these operations need to be performed and any dependencies that exist between two operations. Therefore, we name this graph representation as Lookup Flow Graph (LFG) throughout this paper. Note that a LFG may contain *field nodes* as well as *decision nodes*. From the network standpoint, a LFG field node specifies the individual header field, width of each field, size of the corresponding lookup table, and the type of lookup that needs to be performed.

2) Design Perspective: While a LFG provides the network-related specifications, it lacks design parameters for it to be implemented as a packet processing engine. Therefore, the designer has to elaborate the LFG by adding the design considerations for each individual field search. In an elaborated LFG a field node has to have the structure shown in Figure 1, which contains both network and designer perspectives. This gives adequate information required to implement such a lookup scheme on the desired implementation platform. An example sequence of packet header fields and its potential elaborated LFG are shown Figure 2.

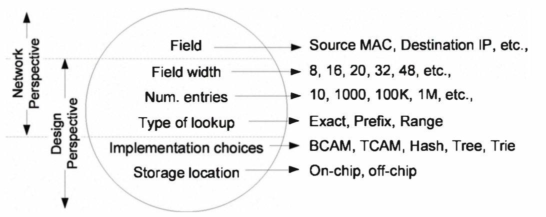

Fig. 1. Structure and potential field values of a field node

3) Architecture Perspective: As shown in Figure 1, a certain field search can be implemented using one of the implementation choices specified in the LFG. In this work, we consider 5 such lookup techniques: 1) Hash, 2) Binary CAM, 3) Ternary CAM, 4) Tree and 5) Trie. These are the prominent search techniques used to perform field search in the context of networking [10], [13], [20], [6], [3], [11]. In order to express the above techniques in hardware terms, we abstract the resource consumption and latency of operation of these search techniques, considering both on-chip and off-chip storage options. To assist this, we use the notations listed in Table I. The hardware abstraction of each search technique is given in Table II. Figure 3 illustrates a potential pipeline design that may be produced by the tool to perform packet lookup for the LFG given in Figure 2.

In this abstraction, we make several assumptions on the search techniques and hardware. These assumptions can be modified according to the design considerations for a given problem.

- A hash lookup operation completes in at most two cycles. First access for hash table lookup and second for collision resolve (if required).
- Hashing increases lookup table size by a factor of μ.

TABLE I
NOTATIONS

Symbol	Description
N	Num. of entries in a lookup table
W	Field width in bits
M	Total amount of allowed on-chip memory
E	Total num. of allowed I/O pins
E_{mgt}	Num. of pins for ext. memory management
T	Bits per trie node
μ	Table scale factor of hashing
α	Resource scale factor of CAM
β	Node scale factor of trie

- Due to the cost of implementing a CAM on FPGA, a resource scale factor α is introduced.
- Building a trie for an N entry table results in a trie with βN nodes.
- Off-chip memory is either Static or Reduced-Latency Dynamic Random Access Memory (SRAM/RLDRAM)
- Off-chip memory has three times the latency of on-chip memory; memory controller introduce latency.

When the implementation choice of a LFG node is identified, depending on the type of lookup, size of the lookup table, etc., completing the lookup operation for a single LFG node may require traversal of multiple pipeline stages (e.g. tree/trie lookup). The number of stages required per lookup is equal to the latency of operation given in our hardware abstraction in Table II. In this paper, we assume that operation for each pipeline stage is statically assigned. The possible operations for a pipeline stage are:

- Hash computation + on-chip/off-chip hash table access
- Hash collision resolve
- On-chip/off-chip CAM access
- On-chip/off-chip tree/trie node access

As stated earlier, all on-chip operations are performed in a single cycle while off-chip operations require three cycles.

4) Hardware Perspective: Even though modern FPGAs have abundant resources available on-chip, in applications of this nature, such resources are easily exhausted. In this work, we assume that the maximum amount of allowed resources is specified. These resources can be the amount of on-chip memory, logic blocks, and external I/O pins of a chip.

C. Problem Definition

Even though several variants of this problem may exist, we define our problem as follows: *Given a LFG and the maximum allowed hardware resources - on-chip memory M, and external I/O pins E - find:*

- Implementation choice and storage location for each LFG node
- A design of a linear pipeline corresponding to the LFG
- The operation for each pipeline stage

such that the number of pipeline stages is minimized, under the constraints:

- The total amount of on-chip memory consumed by the pipeline $\leq M$ and
- The total number of external I/O pins consumed by the pipeline $\leq E$

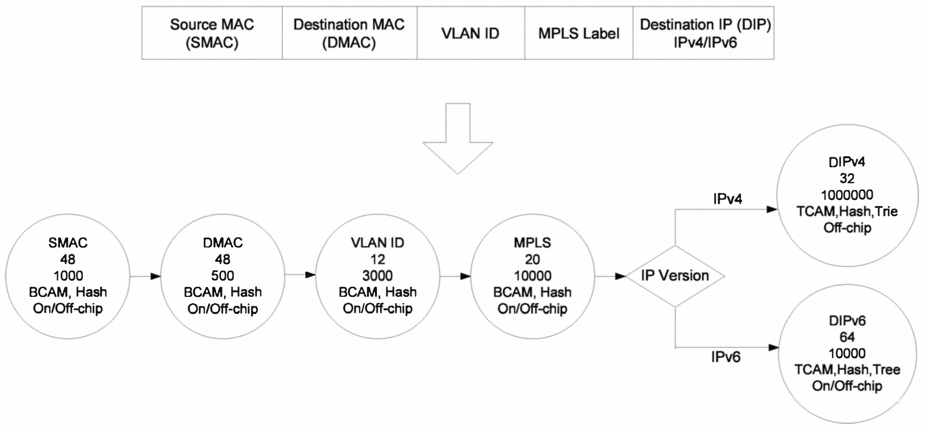
Source MAC (SMAC)
Destination MAC (DMAC)
VLAN ID
MPLS Label
Destination IP (DIP) IPv4/IPv6
SMAC 48 1000 BCAM, Hash On/Off-chip
DMAC 48 500 BCAM, Hash On/Off-chip
VLAN ID 12 3000 BCAM, Hash On/Off-chip
MPLS 20 10000 BCAM, Hash On/Off-chip
IP Version
IPv4
IPv6
DIPv4 32 1000000 TCAM,Hash,Trie Off-chip
DIPv6 64 10000 TCAM,Hash,Tree On/Off-chip

Fig. 2. An example but realistic lookup scheme and corresponding LFG

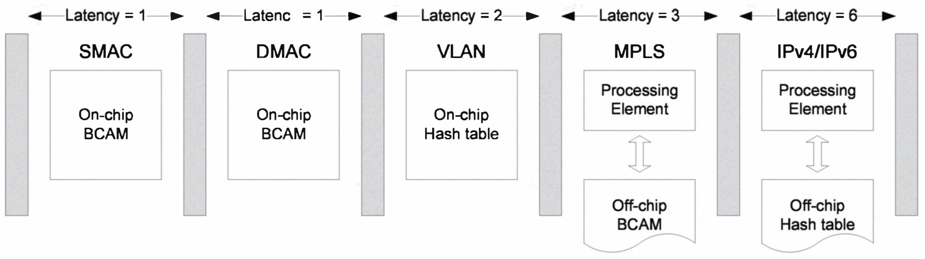

Fig. 3. A potential design for pipelined packet lookup for the LFG given in Figure 2

TABLE II
HARDWARE ABSTRACTION OF SEARCH TECHNIQUES

Search Technique	On-chip			Off-chip		
	On-chip mem.	I/O pins	Latency	On-chip mem.	I/O pins	Latency
Hash	μNW	0	2	0	$W + E_{mgt}$	6
CAM	αNW	0	1	0	$W + E_{mgt}$	3
Trie	βNT	0	W	0	$W(T + E_{mgt})$	$3W$
Tree	NW	0	$\log N$	0	$\log N(W + E_{mgt})$	$3 \log N$

IV. RESOURCE OPTIMIZATIONS

A. Branch-and-Bound Based Resource Optimization

While manual identification of the implementation choices and storage location is possible for a simple and small scale LFG, with a collection of larger and complex LFGs, the task becomes infeasible. Further, with the number of implementation choices and storage locations, brute force identification of the optimal organization of lookups also becomes inefficient. For example, a h level LFG with m implementation choices and n storage locations will yield a tree of $O(m \times n)^h$ nodes when fully generated. In a realistic scenario, where $h = 7, m = 5, n = 2$, number of tree nodes becomes 10 million. Therefore, we propose a resource planning tool that automates the process and identifies the optimal choices for the LFG nodes in a time efficient manner.

In order to solve this optimization problem, we use Branch-and-Bound (BnB) [15]. While other algorithms such as greedy algorithms or dynamic programming could have been employed, considering the nature of the problem, BnB produces the optimal result with less computational complexity. The input to the BnB algorithm is a single LFG input by the user. The extension to multiple LFG scenario is discussed in Section IV-B. The goal of BnB is to complete the assignment of implementation choice and storage location for each field node of the input LFG under the given resource constraints, optimized for the user specified parameter. The resource constraints have to be realistic in order for the algorithm to find a solution. If the resources allotted are not adequate for the input LFGs, the algorithm output will be, simply, no possible solution.

The BnB algorithm expands each LFG node with its implementation choices and storage location and picks the node with the lowest latency out of the expanded nodes as the next potential expansion node, based on a distance heuristic. When a node is expanded, depending on the choice of search technique and storage location, latency and resources are allocated for that node based on Table II. The BnB algorithm is an optimal search algorithm due to the expansion criterion.

Since only the lowest latency node is expanded at any point, if a solution is found by the algorithm, it is an optimal solution. Further, the search space of the BnB algorithm can be reduced with the help of an admissible heuristic[2]. In the context of our problem, since we consider minimizing the number of pipeline stages as our optimization goal, the heuristic has to be chosen to achieve this goal. The lowest pipeline length is achieved if the implementation choice is picked as on-chip BCAM/TCAM (Table II), in which case the latency will be 1. Therefore, if all lookups were performed using on-chip CAM, then the lowest possible latency becomes the depth of the LFG. This stands as the lower bound for the latency for any LFG. Hence, we choose our admissible heuristic for a node x, $h(x)$ as:

$$h(x) = D - d(x)$$

where D is the maximum depth of the LFG and $d(x)$ is the depth of the considered node x in the LFG. Therefore, $h(x)$ is simply estimating how costly the rest of the path will be if the remaining nodes were to be implemented using on-chip CAMs. Hence, $h(x)$ is admissible since it never overestimates the lowest cost to goal.

Due to the dependencies and certain conditions that exist between lookup operations, a LFG as it is cannot be fed as input to the BnB algorithm. The reason behind this is, as shown in Figure 4, certain nodes (nodes 4 and 7 of the LFG) may have multiple parent nodes. This causes the BnB algorithm to ignore some paths, albeit all the paths need to be considered, since the original BnB algorithm terminates when a leaf-node (i.e. node 7) is reached. For example, traversing the path $1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 7$ in the LFG causes the nodes 3 and 5 to be ignored. Therefore, we apply a preprocessing step called *straightening* on the LFG, which guarantees the property that no child node will have more than one parent node. Figure 4 illustrates the straightening process.

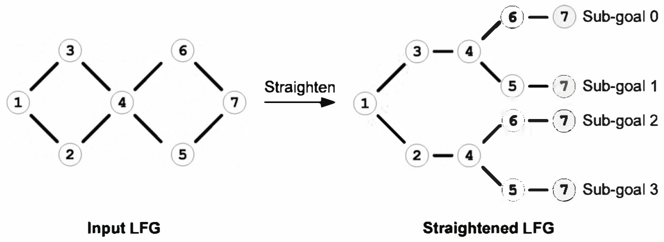

Fig. 4. Straightening process to assist the operation of Branch-and-Bound

In the straightened LFG, each leaf-node is a sub-goal. Reaching all the sub-goals may not be necessary since all the distinctly numbered nodes may be visited before reaching all the sub-goals. In order to reach all the distinctively numbered nodes to complete the search, we modified the original BnB algorithm accordingly. The modified algorithm is given in Algorithm 1. The original BnB algorithm generates potentially safe nodes and traverses the tree recursively until a sub-goal is reached. The modification is invoked when a sub-goal is reached and when the algorithm recursively returns until it reaches a split node in the straightened LFG. A split node is a node that has multiple children. When the split node is found, the currently inspected child is dropped and the original BnB algorithm is called to reach the rest of the sub-goals. Upon completion of the algorithm, the optimal implementation choice and the storage location of each node is identified for all the field nodes of the input LFG.

[2]A heuristic is admissible if it estimates no more than the lowest-cost path to the goal.

Algorithm 1 Multi-goal Branch and Bound

```
Require: LFG, BnB() - basic branch and bound algorithm
  if $node \rightarrow goal \,||\, goal$ then
    if $node \rightarrow LFGchildren.size() > 1$ then
      $node \rightarrow LFGchildren.pop\_front()$;
      $goal = false$;
      $goal$ = BnB(node);
    else
      return $true$;
    end if
  end if
```

B. Extension to Multiple LFGs

The above BnB based optimization is only for a single LFG. As mentioned in Section I, a single router/switch may require to perform lookup operations for multiple lookup schemes. For such a scenario, when multiple LFGs are input, the BnB algorithm is able to find the optimal choices for each individual node, but does not exploit the possible resource sharing among the LFGs. In the context of this problem, resource sharing can take two forms: 1) Sharing the same external I/O pins or 2) Sharing the same memory module while accessing different partitions of memory. To achieve these, resource sharing has to be identified among the LFGs. In order to assist this, we propose an extension to handle the multiple LFG scenario.

The order in which the lookup operations are performed has to be respected in each LFG. Therefore, the flexibility in reordering nodes is limited. However, due to the layered operation in routers/switches, the packet headers are organized in such a way that similar lookup order can be observed in different lookup schemes. Therefore, we propose a heuristic that scans through the LFGs and organizes the LFG nodes in such a way that resource sharing is maximized.

The above problem can be seen as merging the multiple LFGs into a single LFG such that, the resultant LFG can be mapped on to the linear pipeline architecture. During the merging process, nodes from the LFGs may be moved and/or merged. Initially, the heuristic starts with an empty pipeline (zero stages) and serially scans through all the LFGs output by the BnB algorithm. When inserting a node from a LFG to the pipeline, the existing nodes in the pipeline are scanned to see whether any sharing is possible with the existing pipeline stages. If possible, the two nodes are merged and if not, then the node is implemented as a standalone stage either parallel

TABLE III
RESOURCE UTILIZATION COMPARISON FOR INDIVIDUAL LFG IMPLEMENTATION

Lookup Scheme	Tool/Manual		
	On-chip mem.	I/O pins	Latency
Ethernet Bridging I	15.8	148	11
Ethernet Bridging II	11.7	186	10
Mac-in-Mac	11.7	0	7
Q-in-Q	8.6	0	5

or serial to the existing pipeline. In this work, two nodes are merged if and only if the the implementation choice and data width are the same. When a node is inserted serial to the existing pipeline, depending on where the node resides, the rest of the stages may have to be shifted to create space for the new stage.

During the above process, the storage location of some nodes might be changed. In the BnB based algorithm, it is assumed that initially, all the LFGs have the same user specified maximum hardware budget. When combining the LFGs, the available amount of on-chip memory may become insufficient. In such cases, the storage location of such nodes are changed to off-chip and the resource requirement is adjusted properly. Note that sharing can be further enhanced by manual inspection after the above technique identifies the available sharing. Various other methods to improve sharing may be explored, however, in this preliminary study we propose a basic technique to perform this task.

V. PERFORMANCE EVALUATION

In order to evaluate the resource planning tool being proposed, we consider a state-of-the-art FPGA platform and a set of realistic lookup schemes. A Xilinx Virtex-6 XC6VLX760 [19], with 33 Mb of on-chip memory and 1200 external I/O pins, was chosen based on its performance and resources. For the experiments, we assume that 20 Mb of on-chip memory and 750 external I/O pins are available for the implementation of the lookup pipeline.

For lookup schemes, we considered four realistic lookup schemes that are widely used by network manufacturers. Some of these lookup schemes represent the requirements of specific customers and therefore, the details of the lookup schemes and details about the lookup tables cannot be made publicly available. Such lookup schemes are listed as, "Ethernet Bridging" while others are well known schemes. The Ethernet bridging schemes I and II are of depths 4 and 5, respectively, including combinations of MPLS, MAC address, and VLAN lookup. In all the schemes, the lookup table sizes range from 4k - 256k. Implementing such customized lookup schemes on a single chip can be challenging when multiple implementation and storage choices are available and when multiple lookup schemes need to be implemented.

- Ethernet Bridging I
- Ethernet Bridging II
- Mac-in-Mac
- Q-in-Q

We conducted experiments using our tool by generating the LFGs for the above schemes. The LFG field nodes were tagged with appropriate implementation choices and both on-chip and off-chip storage locations. The appropriateness of a certain implementation choice for a certain field lookup depends on the characteristics of the table values. For example, for a prefix type lookup similar to IP lookup TCAM, Trie, Tree and hash are good candidates, while BCAM is not.

Manual identification of implementation choice and storage location was considered as the baseline and no sharing was assumed. While manual identification can be inefficient, for fair comparison purposes, the optimal choice identified by the tool was considered as the manual identification as well, for the single LFG case. However, in real-life scenarios, the optimal choice identification can become similar to brute force method discussed in Section IV-A, which has significantly higher time/space complexity than that of the proposed technique.

For the resource modeling portion, the scaling factor were chosen as follows: $\mu = 1.5$, $\alpha = 1.2$, $\beta = 1.7$. These values represent reasonable approximations, especially for FPGA implementations. Table III summarizes the results when individual LFGs are considered for implementation. Table IV shows the benefit of using the proposed technique in identifying efficient organization and resource sharing for the cumulative lookup schemes, compared with the baseline implementation. The manual method requires over 200 external I/O pins compared with the proposed solution. The device itself has only 1200 pins available and the manual method leaves 240 pins for interfacing with the network, which might become insufficient. Therefore, resource sharing becomes critical when complex and multiple lookup schemes need to be implemented on a single chip.

The lookup schemes considered for the evaluation exist in Layer 2 and Layer 2.5 of the TCP/IP protocol stack. Therefore, most of the lookups can be performed using BCAM and hash. Hence, the packet latency is low. However, when IP lookup and packet classification are considered, tree and trie search will also be employed in which case, the latency may increase. From Tables III and IV, it can be seen that the proposed tool is able to host all four lookup schemes on a single chip while the manual approach consumes more resources than what is allotted for the task.

In order to quantitatively evaluate the amount of sharing, we take the ratio of number of lookups performed using the given amount of hardware. This metric stands as a measure of the amount of sharing achieved using the tool. We observed that, using 20 Mb of on-chip memory and 750 external I/O pins, the proposed technique is able to perform 20 lookups while the manual approach can perform only 15 lookups. The benefits of the proposed approach becomes more significant when multiple LFGs are considered. This is because having multiple LFGs increases the chances of sharing.

In addition to the benefits, the tool can be used to identify whether a given design can be fit on the FPGA without the use of any external I/O at all. This is desirable as the use

TABLE IV
RESOURCE UTILIZATION COMPARISON FOR MULTIPLE LFG IMPLEMENTATION

Lookup	Tool			Manual		
Scheme	On-chip mem.	I/O pins	Latency	On-chip mem.	I/O pins	Latency
Ethernet Bridging I	15.8	148	11	15.8	148	11
Ethernet Bridging II	17.4	434	14	17.4	508	14
Mac-in-Mac	19.5	634	17	19.5	792	17
Q-in-Q	19.5	744	17	19.5	960	17

of external memory introduces latency due to the memory controller operation which ultimately may degrade throughput. A case may exist in which, the latency of the pipeline design produced by the proposed tool is more than that of the manual design. This is possible due to the movement of LFG nodes when trying to map all LFGs on to a single pipeline. The performance of the pipeline with respect to throughput has not been discussed in this work in detail. However, with current state-of-the-art FPGA devices, achieving $200 - 300$ MHz clock rates has become possible without significant engineering effort. While this paper puts foundation for this research, future extensions may consider elaborate discussions on throughput and power metrics.

VI. CONCLUSION

In this paper, we proposed a resource planning tool for organizing multiple packet lookup schemes on to resource constrained hardware platforms. Due to the increasing complexity of packet header lookup and the necessity to perform different types of lookups based on the forwarding technologies adopted, the task of performing single chip lookup has become challenging due to the resource limitations. Hence, efficient allocation of available hardware resources become critical when implementing multiple lookup schemes.

We adopted a hierarchical approach to solve this problem by first identifying the optimal search strategy for a single lookup scheme using an Integer Linear Programming (ILP) based optimization. An extension was proposed to handle multiple lookup schemes, which combines the solutions for individual lookup schemes and produces a static linear pipelined architecture to perform packet header lookup on hardware. By using a Xilinx Virtex-6 device, we showed that up to 4 complex lookup schemes can be implemented using only 20 Mbit of on-chip memory and 750 external input/output pins.

In this work, we assumed the pipeline architecture to be linear. However, using a non-linear pipeline architecture, greater benefits in resource efficiency can be achieved at the expense of throughput. This will require feedforward and feedback connections across pipeline stages to route the packet along the pipeline and in order to guarantee conflict-free operation, packet scheduling has to be taken into consideration. Further, the tool can be extended to automatically generate the pipeline design code in a chosen hardware description language based on the generated pipeline design. These will be considered in the future developments of this work.

REFERENCES

[1] M. Attig and G. Brebner. 400 gb/s programmable packet parsing on a single fpga. In *Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*, ANCS '11, pages 12–23, Washington, DC, USA, 2011. IEEE Computer Society.

[2] M. Bando, N. Artan, and H. Chao. Flashlook: 100-gbps hash-tuned route lookup architecture. In *High Performance Switching and Routing, 2009. HPSR 2009. International Conference on*, pages 1 –8, june 2009.

[3] M. Bando, Y.-L. Lin, and H. J. Chao. Flashtrie: Beyond 100-gb/s ip route lookup using hash-based prefix-compressed trie. *Networking, IEEE/ACM Transactions on*, PP(99):1, 2012.

[4] Huawei. Q-in-q. www.huawei.com/products/datacomm/pdf/view.do?f=556.

[5] I. E. T. F. (IETF). Ipv6 addressing architecture. http://tools.ietf.org/html/rfc4291.

[6] G. Jedhe, A. Ramamoorthy, and K. Varghese. A scalable high throughput firewall in fpga. In *Field-Programmable Custom Computing Machines, 2008. FCCM '08. 16th International Symposium on*, pages 43 –52, april 2008.

[7] W. Jiang and V. Prasanna. A memory-balanced linear pipeline architecture for trie-based ip lookup. In *High-Performance Interconnects, 2007. HOTI 2007. 15th Annual IEEE Symposium on*, pages 83 –90, 2007.

[8] J. Kempf, S. Whyte, J. Ellithorpe, P. Kazemian, M. Haitjema, N. Beheshti, S. Stuart, and H. Green. Openflow mpls and the open source label switched router. In *Teletraffic Congress (ITC), 2011 23rd International*, pages 8 –14, sept. 2011.

[9] C. Kozanitis, J. Huber, S. Singh, and G. Varghese. Leaping multiple headers in a single bound: Wire-speed parsing using the kangaroo system. In *INFOCOM, 2010 Proceedings IEEE*, pages 1 –9, march 2010.

[10] H. Le, T. Ganegedara, and V. Prasanna. Memory-efficient and scalable virtual routers using fpga. In *Field Programmable Gate Arrays (FPGA), 2011 International Symposium on*, 2011.

[11] H. Le and V. Prasanna. Scalable high throughput and power efficient ip-lookup on fpga. In *Field Programmable Custom Computing Machines, 2009. FCCM '09. 17th IEEE Symposium on*, pages 167 –174, april 2009.

[12] C. Neely, G. Brebner, and W. Shang. Flexible and modular support for timing functions in high performance networking acceleration. In *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, pages 513 –518, 31 2010-sept. 2 2010.

[13] H. Song and J. W. Lockwood. Efficient packet classification for network intrusion detection using fpga. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, FPGA '05, pages 238–245, New York, NY, USA, 2005. ACM.

[14] M. Trick. A tutorial on integer programming. http://mat.gsia.cmu.edu/orclass/integer/integer.html.

[15] Wikipedia. Branch and bound algorithm. http://en.wikipedia.org/wiki/Branch_and_bound.

[16] Wikipedia. Ieee 802.1ah-2008. http://en.wikipedia.org/wiki/IEEE_802.1ah-2008.

[17] Wikipedia. Ieee 802.1q standard. http://en.wikipedia.org/wiki/IEEE_802.1Q.

[18] Wikipedia. Multiprotocol label switching. http://en.wikipedia.org/wiki/Multiprotocol_Label_Switching.

[19] Xilinx. Virtex-6 lxt fpgas. http://www.xilinx.com/products/silicon-devices/fpga/virtex-6/lxt.htm.

[20] S. Yusuf, W. Luk, M. Sloman, N. Dulay, E. Lupu, and G. Brown. Reconfigurable architecture for network flow analysis. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(1):57 –65, jan. 2008.