Better by a HAIR: Hardware-Amenable Internet Routing

Firat Kiyak* Brent Mochizuki* Eric Keller† Matthew Caesar*
*University of Illinois at Urbana-Champaign †Princeton University
{fkiyak2,mochizu1,caesar}@illinois.edu ekeller@princeton.edu

Abstract—**Routing protocols are implemented in the form of software running on a general-purpose microprocessor. However, conventional software-based router architectures face significant scaling challenges in the presence of ever-increasing routing table growth and churn. Recent advances in programmable hardware and high-level hardware description languages provide the opportunity to implement BGP directly at the hardware layer. Hardware-based implementation allows designs to take advantage of the parallelization and customizability of the underlying hardware to improve performance. As a first step in this direction, we design and implement a hardware-based BGP architecture. To understand the challenges in doing this, we propose an architecture and logical design for the core components of BGP running as a logical circuit in an FPGA. We then enumerate sources of complexity and performance bottlenecks, and derive modifications to BGP that reduce complexity of hardware offloading. Our results based on update traces from core Internet routers indicate an order of magnitude improvement in processing time and throughput.**

I. Introduction

The Internet is a very large and complicated distributed system. Selecting a route involves a computation across millions of routers spread over vast distances, multiple routing protocols, and highly customizable routing policies. To perform this computation based on up-to-date information, the state of network paths is propagated across ISPs through the use of the Border Gateway Protocol (BGP). While BGP has performed this job well for many years, offering highly configurable operation and control over propagation and selection of routes, it is facing tremendous scaling challenges in modern networks. BGP-speaking routers must process millions of updates daily over hundreds of thousands of prefixes. Furthermore, update arrivals are quite bursty in nature, providing a highly variable workload for routers, and to avoid triggering outages and routing loops routers must be provisioned for handling the *peak* load. Worse still, the number of Internet routes and their churn is steadily increasing, with predictions that current router architectures may be unable to keep up [1].

To cope with these loads, some BGP implementations leverage *timers* to rate-limit update traffic, and only periodically exchange deltas (differences) of routing state. However, timers slow routing *convergence* by extending the time it takes for routers to learn the current state of a route. Worsening convergence leads to black holes, routing loops, and other anomalies, increasing the potential for packet loss. Alternatively, network operators incorporate *flap damping* into route selection, by forcing less-stable routes to be artificially "held-down" and withdrawn from use. While flap damping reduces routing instability, it also harms availability, by removing working paths from use. In fact, conventional wisdom is to disable flap dampening, as it can be inadvertently set off during path exploration and can sometimes leave a router with no route for some periods of time, leading to black holes [2]. Unfortunately, the availability and convergence issues introduced by damping and timer-based approaches is becoming even more serious, with the increasing levels of Internet traffic and deployment of applications with real-time requirements (gaming, virtual worlds, VoIP).

Traditionally, BGP is implemented in the form of a software daemon running on a commodity microprocessor. However, history has shown that when software designs hit a performance wall, the wall can be overcome by hardware offloading, as it allows the design to take advantage of the parallelization and customizability of the underlying hardware. While processing protocols in hardware introduces several challenges, recent advances in hardware technologies vastly simplify the process of hardware-based implementation. Hardware description languages like Verilog, SystemVerilog and VHDL, tools for synthesizing transactional models such as from Bluespec [3], and tools for compiling high-level languages such as from Synfora [4] ease the ability to prototype and debug hardware. Advances in special-purpose hardware with customized instruction sets for specific applications (e.g., GPUs) are gaining increased acceptance and demand. Reconfigurable hardware (e.g., FPGAs) enables hardware to be updated and patched without physical replacement.

At the same time, implementing protocols in hardware also offers some benefits. Direct implementation of hardware allows the designer to leverage *parallelism*. Protocol messages may be pipelined within the same circuit, and functional blocks may be replicated to create multiple pipelines. Implementation in hardware allows the designer to leverage domain knowledge of the protocol or workload to make common-case operations traverse few gates or execute in few cycles. Finally, we believe that with the advent of multi-core processors, the increasing popularity of graphics processor programming [5], and the increasing deployments of resource constrained and embedded devices, programmers will need to be increasingly aware of constraints of the underlying hardware.

Unfortunately, higher-layer network protocols like BGP were not designed with hardware in mind, complicating the ability to offload them into hardware and reducing efficiency of the resulting design. To address this problem, our work provides two key contributions:

1) To characterize complexity of hardware offloading, we provide an empirical analysis of the challenges associated with offloading BGP into hardware. First, to study inefficiencies and bottlenecks, we propose an architecture and logical design for implementing a *hardware-based* version of the BGP protocol (i.e., in the form of a circuit running directly on a semiconductor device). We provide circuit designs for the core components of the BGP protocol, including memory management, route selection logic, and packet parsing. Then, based on this design, we enumerate features of BGP that increase complexity of hardware-based implementation.
2) To mitigate complexity of BGP offloading, we propose protocol changes to make BGP more amenable to hardware-based operation. In particular, we design a *hardware-amenable* variant of BGP (which we call *Hardware Amenable Internet Routing (HAIR)*), which simplifies offloading into hardware, while retaining BGP's semantics. Our design works by making three key simplifications to BGP: (i) replacing variable length fields in packets with fixed-length *labels* to simplify parsing, (ii) replacing trie-based lookups by allowing routers to directly *index* into routing state, and (iii) simplifying decision logic by enabling routers to provide a fixed ranking over routes (thereby avoiding issues that can arise with use of MED).

While conventional wisdom is that higher-layer protocols such as BGP must be implemented in software, we examine the extreme design point of an all-hardware implementation and find that modern technologies and several small protocol changes make a hardware-targeted protocol design a better option, due to significant performance improvements. In particular, due to the increased parallelism achievable in hardware, processing rate and latency are improved by multiple orders of magnitude, and our hardware-amenable design improves performance by an additional order of magnitude.

Roadmap: This paper proceeds as follows. Section II describes some limitations of hardware offloading, and overviews modern hardware design technologies that may mitigate some of these limitations. We then describe our design architecture for BGP in hardware in Section III, along with a description of key performance bottlenecks. We then propose design guidelines and a variant of BGP (called HAIR) that is more amenable to hardware offloading in Section IV. We then evaluate performance in Section V, briefly summarize related work in Section VI, and conclude in Section VII.

II. Background

In this section we give some background to address some common objections to the use of hardware offloading and to understand the capabilities of modern FPGAs in order to better understand the design.

A. Limitations of hardware-based implementation

The traditional arguments against offloading higher layer protocols (some of which are outlined in [6]) are threefold: the performance benefits might not outweigh the costs, it harms the ability to update protocols or deploy new ones, and it complicates implementation work. We discuss these issues below. Although these objections still hold for many protocols, we discuss why new technologies address these traditional objections in certain circumstances.

1) Performance benefits and Moore's law: General-purpose CPUs have steadily increased in clock speed over many years, with conventional wisdom that new CPUs double their processing rate every two years. In the context of hardware, a similar increase in performance has been observed, with ASIC-based designs keeping up with the clock speeds of conventional CPUs. However, while FPGAs have undergone a similar performance improvement rate over the years, they have consistently lagged behind CPU clock speeds by a factor of roughly 10x. This gives general-purpose CPUs an advantage in processing power.

However, modern CPUs no longer undergo regular increases in processing speed, but instead are becoming more parallel by offering multiple processing cores. Additionally, FPGAs are still increasing in speed and hardware-based implementations of protocols can be made extremely *parallel*. Thus, considering hardware-based implementation can reveal how to best leverage parallelism when designing protocols.

2) Complicates implementation: A second challenge that has traditionally faced hardware-based implementations, is the complexity of carrying out the implementation. Programming hardware has traditionally required knowledge of operation at the gate-level, making building large designs a time-consuming and error-prone process. However, the advent of modern *hardware description languages*, and more recently the capability to "compile" a high level language, such as C, to an FPGA implementation alleviates the need to implement designs via low-level mechanisms. Moreover, implementing a new design no longer requires starting from scratch: just as software programming allows use of *libraries* of code, hardware description languages make *reuse* of code simple to do with open interfaces. Open-source implementations of hardware design libraries are increasingly made publicly available for commonly-implemented logic [7], [8]. Finally, the cost of programmable hardware, and hardware simulators, has dropped low enough for system builders, from students to commercial programmers, to prototype and experiment with their designs in realistic environments.

3) Worsens protocol ossification: The difficulty of changing deployed protocols is one of the roots of many of the Internet's problems. This makes it hard to deploy new designs with advanced features and functionality. It also makes it difficult to fix problems and bugs in existing protocols.

Implementing protocols in hardware makes this problem worse. While low-level network protocols such as Ethernet and 802.11 are relatively fixed and undergo few changes, and hence are typically implemented in hardware, higher-layer protocols undergo more innovation and suffer from a wider array of bugs and vulnerabilities. Changing already-deployed hardware is an expensive proposition. While ASICs can operate at high speeds, taping out a design and burning it into silicon costs millions of dollars. Furthermore, once the design is deployed, updating hardware traditionally required physical changes to equipment, further increasing cost.

However, modern semiconductors called *field-programmable gate arrays* (FPGAs) can be modified by a customer after manufacturing. FPGA-based designs can be remotely updated and patched in the field, just like software-based systems. FPGAs are flexible enough to support any logical functions that can be implemented in traditional silicon. However, field-programmability comes at a cost—FPGAs have typically been a factor of 10 times slower than ASIC-based designs. That said, FPGAs are being increasingly used, even for high-volume applications traditionally dominated by ASIC designs. This is happening due to the very high upfront costs of ASIC development, and lowering R&D resources (and hence capacity for high-grade quality-assurance to weed out errors prior to deployment—finding a bug in an ASICs require refabricating the entire design at the cost of millions of dollars and remanufacture of all produced components). Moreover, even though FPGAs are slower than ASICs, they can be orders of magnitude faster than general-purpose processors due to the ability to leverage parallelism. Finally, enormous strides have been made in the area of *hardware-software codesign*. For example, the field of *configware* is concerned with co-compilation of hardware and software designs across both an instruction-stream based microprocessor and attached reconfigurable hardware. While in this paper we consider the extreme design point of offloading the entire network protocol into hardware, several of our design's functional components may be offloaded into software if desired. We describe more details of reconfigurable hardware in the next section.

B. Reconfiguring hardware with FPGAs

An FPGA is a semiconductor device that can be configured (potentially multiple times) after manufacture. An FPGA consists of an array of configurable logic blocks connected together via a programmable interconnect, allowing a user to program the device with customized hardware designs. Modern FPGAs come in an array of sizes (supporting varying complexities of designs), clock speeds and the number of "pins" that allow communication with external components such as a commodity processor or memory.

FPGA designs are commonly implemented in a hardware description language (HDL) such as Verilog or VHDL. The HDL is then processed by a series of tools into a form that can be loaded onto an FPGA. Commonly, FPGAs are components in complex systems. Because of this, they are often coupled with a general purpose processor. This may be done in several ways. First, the FPGA may reside on a peripheral bus attached to the main processor. For example, within a conventional PC, an FPGA may reside on a network interface card reachable by the processor over a peripheral bus (e.g.,PCI)[8]. Second, the FPGA may be used as a co-processor, where an FPGA is used in a processor socket of a multiprocessor system [9]. Third, the FPGA may be a standalone system, but may run an embedded processor coded as logic within the FPGA. Finally, the FPGA can exist with a processor running as a separate component on the board [10].

Future trends in semiconductors can further extend the usefulness of FPGAs. Modern FPGAs are increasingly integrated with a wider variety of components on-chip, including memory, Ethernet MACs, and general purpose processors. One key bottleneck that limits parallelism in such designs is bandwidth to off-chip components, as all communication must take place over a fixed number of *pins* attaching the FPGA to the rest of the board. To address this, the use of high-speed (e.g., multi-gigabit) serial transceivers will free up pins for additional peripheral devices. Additionally, stacked chip technology where SRAM memory or other components may be layered atop and directly connected to the FPGA's configurable logic. Finally, modern FPGAs offer increasing levels of efficiency for application-specific support, by implementing common functions (e.g.,the carry chain of an adder) directly within individual logical units on the chip. In fact, specialized DSP blocks (e.g.,for performing efficient multiply-accumulate operations) are appearing on newer FPGAs.

III. Offloading BGP to hardware

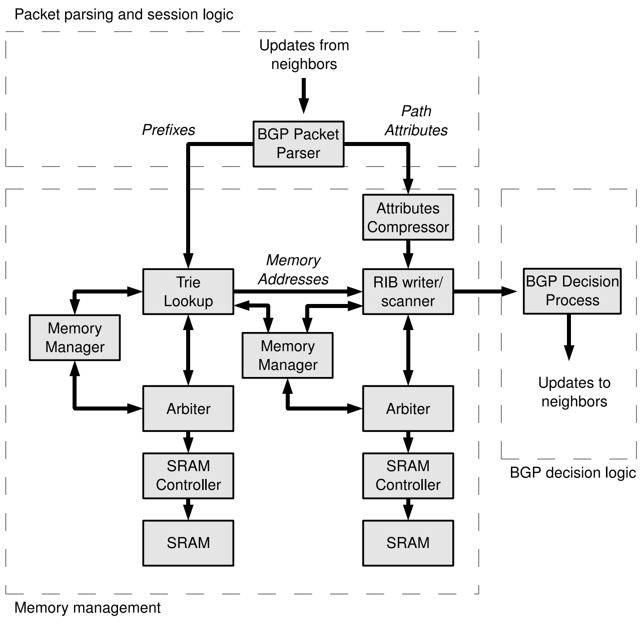

Fig. 1. FPGA-based architecture for BGP.

BGP is a distributed routing protocol that to date has been implemented exclusively in software. In this section we give an overview of the architecture of our BGP implementation in hardware. Our design (Figure 1) is intended to be used with a TCP offload solution [6], and its structure consists of three main components: packet parsing and connection logic to communicate with neighbors, memory management logic to lookup and maintain the trie data structure, and policy and decision logic used to decide on which routes to use and propagate. The figure shows a simplified version of our architecture that supports a single peering session (in practice, the parsing component is replicated for each peering session, with an arbiter component multiplexing their accesses to memory). We describe the details of these components below:

Packet parsing and session logic: This component maintains BGP sessions to the neighboring router, and parses BGP updates into a concise representation that is used internally. It consists of 2 sub-modules: a *packet parser* and a *connection finite state machine (CFSM)*. The packet parser is responsible for parsing incoming packets. For non-update messages it generates internal events, and for update messages it generates prefixes and attribute information. The CFSM module inspects the message type, advances the state machine as appropriate, and either requests the outgoing message generator to generate a message if necessary (e.g., if a BGP Open message is sent requiring a response). In the case of a received update message, the BGP packet parser extracts path attributes and prefixes and forwards them to the Trie Lookup and Attribute Compressor modules. As an optional extension, we could improve performance by cancelling processing of earlier updates if later updates to the same prefix arrive, since the output of a BGP router for a prefix only depends on the currently advertised state of that prefix.

Memory management: This component maintains two structures: the *routing table (RIB)*, which contains the set of currently advertised routes by neighbors, and a *trie* structure which contains references to locations in the RIB. The Trie Manager uses an IP prefix to traverse the trie structure, resulting in a location in the RIB which contains a set of received routes for that prefix (one from each neighbor that advertised a route). In particular, each element of the set contains a list of attributes, and the interface number of the neighbor that advertised the route. The RIB writer/scanner then writes the advertised route to the RIB (or, in the case of a withdrawal, deletes the route from the RIB) given the location produced by the Trie Manager and scans the RIB for other previously advertised routes to the same destination. As each route is written to or scanned from the RIB, it is sent to the BGP decision logic component.

Decision logic: This component implements logic to choose the best route from all advertised routes to the same destination prefix. For each advertised or withdrawn route, the RIB writer/scanner will send it multiple routes to the same destination in sequence. Whenever a new route is driven to the module, it compares the current best route with the new route and updates the current best route if the new route is better. The comparison between two routes is done using the standard BGP decision process (preferring lowest localpref, then lowest AS Path Length, etc.). When all routes to the same destination are finished being sent from the RIB scanner/storer, the best route is output from this module. If the newly-selected best route differs from the previously-selected one, the best route is sent to the forwarding table (FIB) in the data plane (not shown in our figure), and the outgoing message generator module to advertise the new best route to neighbors.

Example: To clarify how the components work together, we trace an example routing update through our design. Suppose an advertisement for a single prefix with a corresponding set of attributes arrives (if an update contains multiple prefixes, this process would be repeated for each prefix). First, the parser reads in the packet, parses the prefix and attributes, and forwards them to Trie Lookup and Attribute Compressor modules. The trie lookup module then looks up the location of the set of advertised routes to the destination prefix and sends this to the RIB scanner/storer, which first stores the new route (given by the Attribute Compressor) to the RIB, then scans the RIB for additional routes to the same destination prefix. It then sends the set of routes, including the newly received route, to the decision logic one by one. The decision logic maintains the current best route and outputs it when all the routes have been processed. The new best route is then sent to the FIB for storage. Similarly, if needed, the best route is forwarded to the outgoing message generator module to generate new advertisements.

IV. A HARDWARE-AMENABLE BGP

While our hardware-based BGP implementation offered improved processing speed over a software-based implementation, it also had some downsides. Its implementation consists of a moderate level of complexity, and it continues to suffer from several bottlenecks – it requires multiple clock cycles to parse update messages, to traverse the trie for the locations of routes in the RIB, and to look up current routes maintained in the RIB. To further improve performance, we relax our design goals by considering a new question—*is it possible to make BGP more amenable to hardware offloading, by allowing modifications to the protocol?* To do this, we enumerate the bottlenecks and develop a new protocol which we term *Hardware-Amenable Internet Routing* (HAIR). We aim to design HAIR to reduce complexity of hardware offloading, to improve performance gains achieved by hardware offloading, yet also to retain the same semantics and features as traditional BGP.

A. Challenges of hardware-based BGP implementation

In this section, we enumerate three key design properties of BGP that complicate offloading to hardware.

No total ordering of routes: When a new route is advertised in BGP, the BGP router needs to determine if the new route is better than its current best route. It does this by scanning over all routes (including the new route) that have been advertised, and selecting the best one. However, the Multi-Exit Discriminator (MED) attribute, used to signal to an immediately-adjacent neighboring AS which ingress link should be used to send traffic to the local AS, prevents each router from having a total ordering over all possible candidate routes [11], and hence advertising a new route requires a complete scan of all existing routes.

Complex lookup: Routers must maintain a data structure to look up the set of advertised routes associated with an IP prefix. This is often done by use of a *trie* data structure, which allows lookup of keys of length n in $O(n)$ time. However, implementing a trie in hardware has some disadvantages. First, implementing data structures with pointers in hardware is complex and requires advanced memory management. Second, a single IP prefix lookup takes a substantial number of cycles since a traversal for IP prefix visits multiple nodes of the trie, requiring each step a separate lookup from the memory in sequential order. While software routers also suffer from lookup delay, it becomes much more apparent in hardware where other parts of the design are running much faster.

Simplify assumptions on transport: The BGP RFC [12] requires protocol messages to be exchanged using TCP, to provide resilience to loss and packet reorderings. However, TCP provides numerous features that complicate its design and hence increase complexity of hardware implementation: for example it performs congestion control, requires de-encapsulation and segmentation logic, and must remain backward compatible with existing implementations. Since BGP itself does not strictly require these features, to simplify offloading, we eliminate TCP and instead use a lightweight procedure that directly acknowledges HAIR messages.

Long and variable-length attribute strings: BGP update messages have dependencies between fields which can introduce complexity in update processing. Each update message contains a list of *unfeasible* (withdrawn) prefixes followed by a list of *reachable* (advertised) prefixes, and a set of attributes. Each of these fields is variable length, may consist of multiple-subfields which in turn can appear at arbitrary positions within the packet (and some of which are optional and need not appear at all). For instance, path attributes section includes a list of attributes and each attribute is in the form of $<$attribute type, attribute length, attribute value$>$ triple. The size of the attribute value field is variable and depends on the attribute length field. Variable length, dependent fields clearly limit the performance since these fields must be parsed in sequential order and the hardware implementation cannot take advantage of its parallel processing capabilities.

Moreover, BGP messages provide for high levels of expressiveness and flexibility by providing extensible attributes. Some examples are community attributes, in which operators may write arbitrary strings, and control operation based on these strings, and AS-Paths, which provide a list of ASes along the path to the destination. While these extensible fields allow expressiveness, they present two problems: their contents are highly redundant (the same field is often sent multiple times in different update messages to the same peer) and the fields themselves are overly verbose (the information within the field can be expressed in a more compact form). This results in wasted bandwidth, which limits the rate at which update messages can be processed.

B. Hardware-amenable protocol overview

In this section, we redesign BGP to make it amenable to hardware offloading. To do this, we propose a new protocol (HAIR), which addresses the three complexities discussed in the previous section:

Optional total ordering: In BGP, a total ordering of routes is not possible. Because of this, to determine the best route, each currently advertised route must be considered. This requires a complete scan of all advertised routes for a prefix, increasing processing time. To address this, our design provides a configurable flag to enforce a total ordering across routes. A network operator may enable this flag to achieve faster processing, yet still enable typical uses of MED.

Simplify lookup: The processes to lookup the set of currently advertised routes associated with an IP routers requires traversing a data structure. The data structure itself requires advanced memory management and the lookup process takes several cycles and several memory accesses. To address this, instead of propagating IPv4 routes, HAIR operates in a *virtual address space* where each destination network is enumerated with a fixed identifier. Here, we assume that each host on the Internet has an address in the form of (virtual supernet ID, virtual subnet ID). HAIR-speaking routers propagate routes to supernets, and HAIR border routers use an IGP to reach hosts internal to its attached subnets. This addressing scheme has the advantage that our hardware implementation uses the virtual supernet ID to directly index into memory for relevant routing information, without need to traverse a trie. This is possible because there would not need to be a longest prefix match and the number of unique virtual supernet IDs would be small enough to allow the routing table to be directly addressed by these IDs and still be small enough to fit in a moderately sized memory. Hence, the lookup can be done in constant time since there is no need of a search or traversal for the information. Moreover, this reduces complexity of the hardware design. While changing the Internet's addressing structure would require substantial work to deploy, several next-generation routing techniques propose routing on fixed network identifiers rather than prefixes (including AIP [13], HLP [14]), and our virtual address space can be directly translated to AIP or HLP's network identifiers. If changing Internet addressing is not desirable, virtual supernet IDs may be translated to IPv4 prefixes. This is done through use of an auxiliary protocol that propagates this mapping, and having routers store this mapping in a local table (in a manner similar to HLP's AS-to-prefix mapping protocol [14]).

Fixed-length, independent fields: The fields in a BGP update are of variable length, which limits performance since these fields must be parsed in sequential order and the hardware implementation cannot take advantage of its parallel processing capabilities. Additionally, the overly verbose encodings used in BGP are wasteful, in that less-redundant more-compact encodings may be read and processed in fewer cycles. To address these inefficiencies, we modify BGP to replace variable-length fields with fixed-size labels. A fixed length attribute may be used to replace a variable-length field when a fixed, finite set of values are used for the field assignment. If variable-length attributes are desired, our design supports them by allowing adjacent routers to run a protocol that periodically propagates mappings from variable-length attributes to fixed length labels. Finally, labels are set within the packet in fixed well-known locations, and in a fixed well-known order, to simplify processing.

However, using fixed-length labels introduces some new challenges: some information in the update messages is variable length by its nature (e.g., the AS Path attribute), some attributes in update messages are optional and may appear as desired (e.g., community attributes), and update messages contain a substantial number of dependencies across different fields. We address these in the following section.

C. Protocol details: using labels

To deal with the challenges of using fixed-length labels, we decouple label assignment from routing updates. The resulting protocol consists of three steps. First, a unique fixed-size label is generated for each unique set of values of a set of variable length fields. Second, the label and the corresponding values of the set of variable length fields are advertised to the receiver. Finally, the label can be used in the following update messages instead of variable length fields. The main observation of this scheme is that variable length fields have to be processed only once, but the corresponding fixed size label is used multiple times in different update messages.

In order to maximize the reusability of the labels, for a given set of attributes we have decided to use two labels: AS Path Label to represent the AS path attribute and Attribute Set Label to represent the remaining attributes. The main intuition behind this separation is that attributes except the AS path define a local policy between peers and the same policies are used over and over again in multiple update messages, whereas AS Path is only used in loop detection and best route decision process. Hence, the new protocol defines two new message types for advertising AS Path Labels and Attribute Set Labels. AS Path Label messages are used to advertise AS Path Labels and the corresponding AS Path to a HAIR-speaking router. Similarly, Attribute Set Label messages are used to advertise a label for the remaining set of attributes. The overall protocol has three key steps:

Computing labels at edge routers: Here we describe how edge routers must redistribute from IPv4 routes received from BGP to HAIR (IP routes received from an IGP are handled in a similar manner). Upon receiving a BGP update message with a new AS Path, the edge router first selects an unused label from the list of free labels. Second, the router creates a two-way mapping between the AS Path and the corresponding label in its internal tables. Third, the edge router sends out AS Path Label advertisement messages to its HAIR neighbors. Finally, the edge router sends out HAIR update messages including that AS Path Label. Redistribution from HAIR to BGP is performed in a similar fashion. Upon receiving a HAIR update message, the edge router replaces all labels with the corresponding variable length fields and sends out BGP update messages if needed. However, when a HAIR AS Path Label advertisement message is received, the edge router simply creates a two-way mapping between the AS Path and the corresponding AS Path Label.

Computing labels at internal routers: Routers within a HAIR network propagate advertisements with labels. To avoid fragmentation of the label space, labels only have local meaning between routers, and *label swapping* [15] is used to translate labels across routers. In more detail: routers receive Label advertisement messages (which propagate label to attribute mappings) or Update messages (which propagate route changes with attributes represented as labels). Once an AS Path Label advertisement message is received, the internal router first creates a one-way mapping from AS Path Label (i.e. inbound label) to AS Path in its internal tables. In addition, the router selects one unused label and sends out AS Path Label messages to its neighbors including the selected label (i.e. outbound label) and the corresponding updated AS Path. Finally, the internal neighbor creates one-way mappings from the inbound AS Path Label to outbound AS Path Labels. Upon receiving a HAIR Update message, the internal router replaces all inbound labels with the corresponding outbound labels for each neighbor and sends out updated HAIR Update messages if the best route information is updated.

Up to this point, we have assumed backward-compatibility with traditional BGP. However, the network operator may instead directly assign policies in terms of the fixed-size labels used in our protocol. Alternatively, labels may also be computed by an RCP, which can directly write these label mappings into routers. In order to do that, the edge routers directly sends HAIR Label Advertisement messages to the RCP instead of internal routers, and then the RCP computes the label mappings and writes those into the internal routers.

D. Design

In this section we briefly give an overview of the HAIR architecture. Our HAIR design (Figure 2) consists of three components: packet parsing and session logic, memory man-

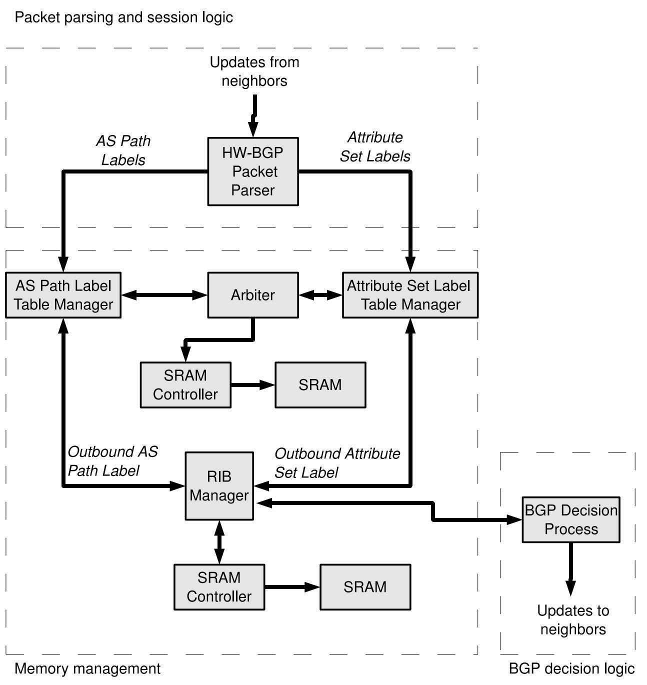

Fig. 2. FPGA-based architecture for hardware-amenable BGP (HAIR).

agement logic, and the BGP decision logic. Messages exchanged by HAIR contain labels in place of variable-length fields. To perform this function, the memory management logic maintains *label tables*. We maintain two separate label tables, one for AS Paths and one for Attribute sets. When an update message is received, the AS Path and other attributes are extracted by the parser. Next, these inbound labels are converted to outbound labels via a label-swapping step, in which a lookup is performed by label table *managers*. Then, the RIB manager receives these outbound labels and updates the RIB and sends out update messages including the outbound labels if needed.

E. Deployment considerations of HAIR

Supporting standard routing policies: Most BGP policies can be described as performing an ordered ranking over the set of advertised routes. To increase processing speed of our design further, we can embed this ranking information within assigned labels. For example, the label itself may correspond to its placement within the ranking of routes, and route selection then simply becomes a matter of performing a numeric comparison to determine the lowest-labeled route. While enumerating routes in this fashion presents challenges, this process is simplified by having an RCP [16] compute an enumeration over the set of historically visible routes, applying a ranking, and installing label maps into routers. Then, only newly visible routes that were not assigned rankings need to undergo the full decision process. From parsing BGP updates, we found these historical rankings to be quite stable, with less than 1 percent of routes in a week not appearing within the previous weeks' advertisements.

Interoperating with standard BGP: Another downside of HAIR is that, unlike the design given in Section III, it cannot directly peer with existing BGP routers. This complicates incremental deployment, especially since our design may only ever be deployed on certain routers or regions of the network where cost concerns or processing requirements are the highest. However, translation between routing protocols has been a widely-studied problem in the context of traditional protocols, through techniques known as *redistribution*. Here, routes from one protocol are re-advertised into another protocol. HAIR routes can be simply redistributed into standard BGP and vice versa since they use the same addressing structures and protocol formats. The main challenge is in converting protocol messages, which requires translation from labels to BGP update contents, and vice versa. Finally, our design is amenable to other deployment strategies, like *tunneling* (e.g., forwarding updates through GRE tunnels over domains that do not support HAIR), and *dual-stack* (e.g., routers maintain processing engines for both HAIR and traditional BGP, and demultiplex message to the appropriate engine based on the version number in the update header).

V. Evaluation

Methodology: To understand the performance properties of our design, we implemented a prototype and replayed BGP update traces against it. We then measured *processing time*, the amount of time required to process a routing update, and *throughput*, the number of routing updates processed per given unit of time (note that processing time is not the inverse of throughput, since our design is pipelined, where multiple updates are processed at different stages at the same time). To evaluate our design, we conducted two kinds of experiments. First, we performed *black-box* measurements, where we simulated loading our design onto a NetFPGA board [8], and monitored the time from when updates arrived at the inputs to when the resulting best route was advertised at the output (using the methodology given in [17]). Second, we performed *microbenchmarks*, where we instrumented our design with counters to determine (for each update) the amount of time it spent in each module of our design. To collect these results, we ran our design on the ModelSim FPGA simulation environment. To evaluate performance under realistic workloads, we replayed Route Views traces [18] against our design. We did this by randomly selecting four vantage points to act as neighbors to our router. We replayed traces collected during October 2008, removing all time between updates such that all the updates arrived at the router simultaneously. To eliminate cold-start effects, we preload routing tables before replaying updates. In addition to evaluating our BGP and HAIR FPGA-based designs, for comparison purposes we also collect black-box results for the Quagga open-source software router. Overall, our design consists of 5239 lines of Verilog code.

Throughput and processing delay: Here we measure the throughput (update processing rate) and per-update processing delay of our design. It is important for protocol designs to have high throughput and low processing delay, as this allows them to handle sudden bursts of updates, to accelerate the convergence process, and to reduce cost of hardware (allowing cheaper and lower clock cycle components). We measure throughput as the number of updates that are processed within a single cycle. We compare *FPGA-BGP* (our design of the standard BGP protocol, running on an FPGA), *HAIR* (our hardware-amenable routing protocol), against *SW-BGP* (the Quagga [19] open-source software router, running on a single core on a 3GHz Intel Core2 Duo processor). Comparing our design against Quagga introduces two key challenges. First, Quagga contains timers which reduce update throughput at the expense of slowed convergence. To address this, we optimized Quagga to immediately pass-through updates, by disabling timers (*SW-BGP-opt*), thereby improving its throughput. Second, it is misleading to directly compare cycles, since commercial CPUs scale to multiple GHz (billions of cycles per second) while FPGAs scale only to hundreds of MHz. To address this we also plot a normalized line showing the performance of Quagga if it were sped up by the factor difference in clock speed between the NetFPGA (125MHz) and the 3GHz processor. These two changes improve Quagga's performance results, to provide a more fair comparison. Figure 3 shows throughput (update processing rate) and Figure 4 shows processing delay.

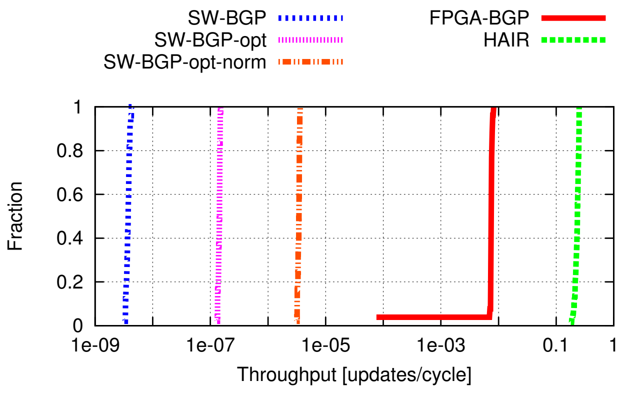

Fig. 3. Performance Results: Update processing rate.

Overall, we find that offloading BGP to hardware provides more than an order of magnitude improvement in throughput over the Quagga software router, and our HAIR implementation improves upon this by another order of magnitude. Similarly, offloading BGP improves per-packet processing delay, and a HAIR implementation reduces delay further. In addition, we found that our FPGA-BGP design reduced delay variability by over an order of magnitude, and HAIR completely eliminates variability by attaining a fixed processing delay across all updates.

To localize the bottlenecks, we instrumented our design with counters (Figure 5) to measure the amount of time

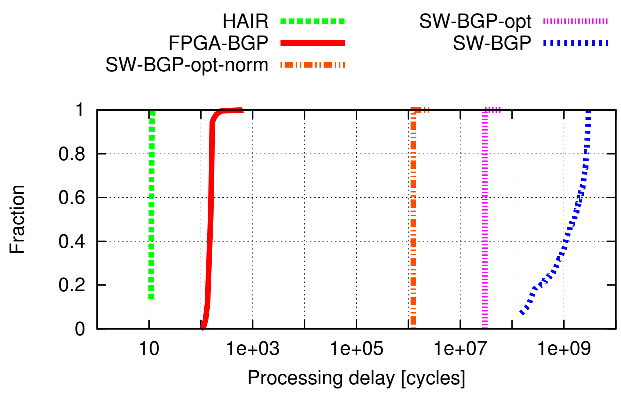

Fig. 4. Performance Results: Per-update processing time.

updates spent in each component. For FPGA-BGP, we found that the memory management component that manages the trie data structure was the greatest source of delay, with the parser, which deals with variable-length fields, close behind. We found that HAIR attains its performance gains by mitigating bottlenecks in all three components.

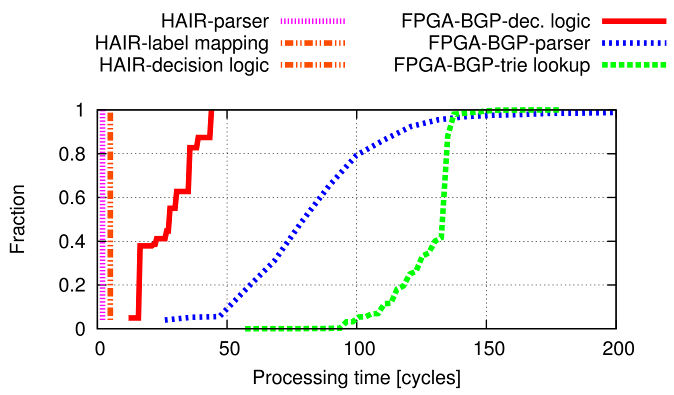

Fig. 5. Performance Results: Microbenchmarks.

Sensitivity to workload changes: To evaluate whether our results held across a variety of workloads, we replayed different update traces against our design. First, we varied the year in which the trace was collected, by replaying a trace of the same length from April 2001, 2004, and 2008. We found a slight increase in processing delay in traces from later years in FPGA-BGP due to an increased trie size (we observed a negligible increase in SW-BGP, as this effect was masked by the magnitude of processing delay). We found that HAIR underwent no increase in processing delay, as the trie was replaced by a constant-time lookup. Next, we varied the number of neighbors (peers) attached to the router (Figures 6 and 7). We found that all designs undergo some decrease in performance with more neighbors, due to the larger number of routes being processed. However, in the hardware-based versions this effect is small, incurring for example only 1 additional cycle per neighbor in HAIR per withdrawal (and zero additional cycles per advertisement).

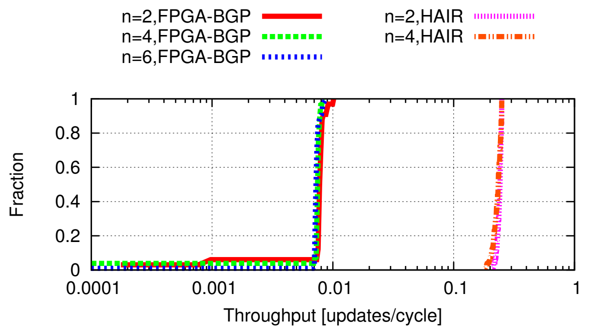

Fig. 6. Sensitivity to workload: Effect of varying number of neighbors on throughput.

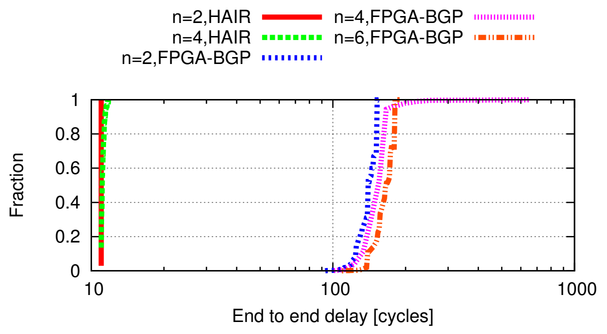

Fig. 7. Sensitivity to workload: Effect of varying number of neighbors on delay.

Properties of workload: Understanding the fundamental level of parallelism achievable in a protocol is important, as hardware-based technologies such as multicore enable the ability to perform multiple computations at the same time. To evaluate this, we analyzed update traces and computed the number of updates that could be processed simultaneously, where two updates can not be simultaneously processed if they read/write the same prefix (Figure 8).

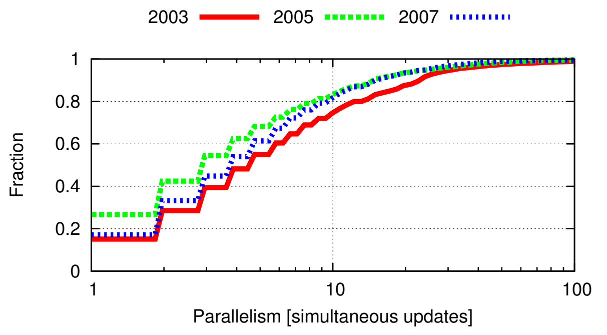

Fig. 8. Properties of workload: Level of parallelism in update traces.

Interestingly, we found that when "spikes" of updates were received at the router, parallelizability increased a large amount. This is shown by the long tail in Figure 8, which extends far beyond the right side of the plot shown. In this case, the 50th percentile is less than 10, but the average over the entire trace is 217. This is important as processing speed is most crucial to avoid worsening convergence during times of elevated load, which happen because link failures cause large numbers of prefixes to be simultaneously withdrawn or advertised, but do not typically trigger multiple updates to the same prefix. While the design presented in this paper performs parallel processing across updates only to the extent of its pipeline, we can leverage the high degree of parallelism present in BGP data by replicating our design within a single FPGA, and balancing updates across the replicas.

Finally, our design allows for an RCP-like system [16] to compute label assignments to routers, to further improve performance. To evaluate feasibility of this approach, we study workloads to capture the number of *label changes* that occur over time, which corresponds to the number of times the RCP would need to refresh label mapping tables within routers. We find that over a 10 day trace, out of 3.2 million updates, only 85,895 unique label mappings need to be published.

VI. Related Work

The question of where the boundary should be between software and hardware has been a long-standing and widely-investigated question in the field of computer science. The field of hardware-software *codesign* focuses on generating designs for systems that are composed of both a microprocessor and a hardware-based logic circuit [20]. *Co-compilation* techniques are used to automatically transform a high-level language into software modules running atop the processor with the rest compiled into logic circuits [21]. While vast advances have been made in this area, additional gains are often attained by leveraging domain-specific information and techniques. We believe our work is complementary to codesign, by considering a simplified version of this problem specifically within the context of networking protocols.

Within the realm of network protocols, hardware offloading may reduce computational costs and speed throughput. First, hardware offloading for TCP is argued to be useful to reduce data copy costs in systems where the host bus is the main bottleneck [6]. Several vendors are beginning to provide network equipment to support TCP offloading, including Broadcom, Chelsio, and Neterion. Second, hardware technologies are commonly used for monitoring workloads. Hardware-based counters are used for monitoring aggregate statistics of data traffic [22], and characterizing anomalies. Third, a variety of protocols at lower layers of the protocol stack are implemented directly in hardware or firmware, such as MAC and physical-layer protocols. This is done to improve processing speed, to reduce reaction time to outages, and to reduce component cost. There has also been work on offloading web server traffic [23] and spam email processing [24] to FPGAs. However, there has not been widespread investigation of offloading routing protocols into hardware. While traditionally there was little need to do so, the ever-increasing scale and churn of networks coupled with rising demands of new applications may require consideration of new architectures. In this work we characterize the implementation of a routing protocol (BGP) in hardware and propose protocol changes to simplify offloading.

Performance of network protocols may also be improved by other means. Computation time of processing may be reduced by using more efficient algorithms and caching results of previous computations. Networks may reduce timers and exchange messages at higher rates to improve convergence time and keep state more up to date [25]. System-wide bottlenecks may be reduced by increasing bandwidth between devices, incorporating more powerful hardware, or configuring the system to reduce unnecessary processing and eliminate bottlenecks. Router load can be decreased by giving certain updates higher priority processing [26]. Techniques such as metarouting [27] reduce likelihood of implementation errors by mapping high-level descriptions to code, and may be extensible to hardware implementation. We believe these works are synergistic with hardware offloading, and may be used in concert with the techniques proposed in our work. Moreover, in addition to performance benefits, we believe that new developments, such as the increasing pervasiveness of multicore technologies [28], graphics processing technologies [5], and resource constrained network elements demonstrate the need for greater awareness of hardware issues when designing network protocols.

VII. Conclusions

In this paper we challenge the conventional wisdom that higher-level protocols such as BGP should be designed for a software-only implementation. We start by designing a circuit that executes BGP directly in hardware. While this design leads to significant performance improvements, hardware implementation is not considered when designing network protocols like BGP. This limits achievable benefits and complicates implementation. Given the ever-increasing loads on routers, we believe future routing protocols should be developed with hardware in mind. As a first step in this direction, we redesigned and implemented a replacement for BGP that simplifies design and offers further performance improvements.

However, our work is only one early step towards developing more hardware amenable network protocols. In future work, we plan to investigate application of co-compilation [21] to determine which parts of BGP computation attain most gains from offloading. It may also be interesting to evaluate a wider array of networking protocols (e.g., storage/filesystem protocols, spam/email and other application services), and to investigate commonalities as a step towards developing a set of shared primitives to simplify hardware offloading.

References

[1] T. Li, "Router scalability and Moore's law," in *Internet Architecture Board meeting (presentation)*, October 2006, http://www.iab.org/about/workshops/routingandaddressing/Router_Scalability.pdf.

[2] Z. Mao, R. Govindan, G. Varghese, and R. Katz, "Route flap damping exacerbates Internet routing convergence," in *Proc. ACM SIGCOMM*, August 2002.

[3] "Bluespec, Inc." http://www.bluespec.com.

[4] "Synfora, Inc." http://www.synfora.com/.

[5] S. Tomov, M. McGuigan, R. Bennett, G. Smith, and J. Spiletic, "Benchmarking and implementation of probability-based simulations on programmable graphics cards," *Computers & Graphics Journal*, 2005.

[6] J. Mogul, "Tcp offload is a dumb idea whose time has come," *Proc. HotOS*, May 2003.

[7] "OpenCores," opencores.org.

[8] J. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA - an open platform for gigabit-rate network switching and routing," *IEEE Microelectronic Systems Education*, December 2007, netfpga.org.

[9] "DRC Computer Corp." drccomputer.com.

[10] "Virtex-5 FXT FPGA ML507 Evaluation Platform," xilinx.com/products/devkits/HW-V5-ML507-UNI-G.htm.

[11] N. Feamster and J. Rexford, "Network-wide prediction of BGP routes," in *IEEE/ACM Trans. Networking*, April 2007.

[12] Y. Rekhter and T. Li, "A Border Gateway Protocol," RFC 1771, March 1995.

[13] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable Internet Protocol (AIP)," in *Proc. ACM SIGCOMM*, August 2008.

[14] L. Subramanian, M. Caesar, C.-T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica, "HLP: A next-generation interdomain routing protocol," in *Proc. ACM SIGCOMM*, August 2005.

[15] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," January 2001.

[16] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and K. van der Merwe, "Design and implementation of a routing control platform," in *Proc. NSDI*, April 2005.

[17] A. Feldmann, H. Kong, O. Maennel, and A. Tudor, "Measuring BGP pass-through times," *Proc. Passive and Active Measurement*, April 2004.

[18] "University of Oregon Route Views," archive.routeviews.org.

[19] "Quagga software routing suite," http://www.quagga.net.

[20] W. Wolf, "Hardware-software co-design of embedded systems," *Proc. of the IEEE*, July 1994.

[21] M. Baleani, F. Gennari, Y. Jiang, Y. Patel, R. Brayton, and A. Sangiovanni-Vincentelli, "HW/SW partitioning and code generation of embedded control applications on a reconfigurable architecture platform," *International Workshop on Hardware/Software Codesign*, May 2002.

[22] Q. Zhao, J. Xu, and Z. Liu, "Design of a novel statistics counter architecture with optimal space and time efficiency," *Proc. ACM SIGMETRICS*, June 2006.

[23] T. Sproull, G. Brebner, and C. Neely, "Mutable codesign for embedded protocol processing," *International Conference on Field Programmable Logic and Applications*, August 2005.

[24] E. Gawish, M. W. El-Kharashi, M. A. El-Yazeed, and A. Salama, "Design and FPGA-implementation of a flexible text search-based spam-stopping firewall," *National Radio Science Conference*, March 2006.

[25] C. Alaettinoglu, V. Jacobson, and H. Yu, "Towards millisecond IGP convergence," in *IETF Draft*, November 2000.

[26] W. Sun, Z. M. Mao, and K. G. Shin, "Differentiated bgp update processing for improved routing convergence," *Proc. International Conference on Network Protocols*, October 2005.

[27] T. Griffin and J. Sobrinho, "Metarouting," *Proc. ACM SIGCOMM*, August 2005.

[28] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Kuetzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick, "The landscape of parallel computing research: A view from berkeley," *Technical Report, UCB/EECS-2006-183*, December 2006.