CAREER: Expanding the Functionality of Internet Routers

Project Summary

The inability to change core functionality beyond a simple packet forwarding service has hindered innovation in the Internet. Although the architectural simplicity of the Internet has likely led to its massive success, it has also had unintended consequences. For example, there is little intrinsic support to measure and monitor the Internet, which makes effective management challenging and has confounded efforts to gain a deep understanding of the Internet's structure and behavior. In addition, real-time applications often rely on overlay networks and application-level optimizations to deliver acceptable performance to users. Moreover, various network security devices are deployed as extrinsic, isolated systems to detect and respond to attacks; methods to detect and react to security threats could be much more effective if there were coordinated detection and response mechanisms integral to the Internet.

Intellectual Merit:
This project develops a new approach for enhancing the functionality of Internet routers, toward the goal of enabling the development of future applications and services. The aim of the proposed research is to provide programmable mechanisms on routers and similar Internet devices to enable service providers, application developers, and researchers to harness in-network capabilities. A new framework for programming network routers will be developed based on primitive functions that enable and expose new and useful capabilities in the Internet. These primitives will be designed to provide in-network support for measurement and monitoring of the Internet, real-time applications, and network security.

Broader Impact:
A reference design and implementation of the programming framework will be made openly available to researchers and deployed and evaluated in the Global Environment for Network Innovations (GENI) experimental network. In this setting, researchers will be able to use it to test research ideas that require or could benefit from additional in-network capabilities. In addition, tutorials will be held to introduce researchers to the system. Undergraduate students will participate in the research activities of this project. Moreover, new course materials will be developed and tailored to undergraduates at smaller institutions. Two sets of coursework will be developed: one for use with a popular reconfigurable hardware system (the NetFPGA) and another for use with the router programming framework to be developed. The educational materials will be made openly available to other undergraduate institutions and workshops will be held to disseminate the materials to the educational community.

1 Project Description

In this CAREER development plan, we first discuss our *research plan*, which focuses on a framework for enabling new classes of applications and services by providing new functionality in the network. We then describe our *educational and outreach plan* to expand undergraduate course offerings and research opportunities in networking.

1.1 Research Plan

1.1.1 Overview and Goals

An astonishing breadth of Internet applications has been developed over the past several decades. Fueled by the continued physical expansion of the network, growth in bandwidths to end hosts [**?**], and wide variety of devices that connect to the Internet, users continue to develop innovative and compelling services. Despite (and perhaps because of) this growth, fundamental functionality *within the Internet* has ossified and remains generally limited to a simple best effort packet forwarding service.

While the simplicity of the end-to-end architecture of the Internet [**?**] and the simple packet forwarding ("stupid network" [**?**]) model have likely enabled the massive success of the Internet, they have also had unintended consequences. For example, there is little intrinsic support for measurement and instrumentation in the Internet, which has made effective network monitoring and management challenging, and has thwarted efforts by researchers to gain a deep understanding of the Internet's dynamic structure and behavior [**?**, **?**, **?**]. Furthermore, developers of real-time applications such as interactive games and multimedia often rely on overlay networks and application-level optimizations in order to deliver acceptable performance to users [**?**, **?**, **?**, **?**]. Lastly, network intrusion detection and prevention systems and deep packet inspection firewalls have pervaded the Internet as security threats have escalated and malware has become more damaging. Whereas these systems are typically deployed as isolated middleboxes, methods to detect and react to emerging threats could be significantly more effective if there was support within the Internet to deploy coordinated detection and response mechanisms, *e.g.*, as proposed in other work [**?**, **?**, **?**]. That these problems have become acute has not been lost on the research community, and there have been a number of recent efforts to develop "clean slate" network architectures [**?**, **?**] and experimental testbeds [**?**].

Another way to address the ossification problem is to *expand network functionality* to better support new and existing applications and services. Routers are potentially ideal targets for new functionality since they dictate how traffic is forwarded between end hosts and because of their view of numerous, diverse traffic flows [**?**, **?**, **?**, **?**]. For example, mechanisms to provide better control over the quality of service might be more easily facilitated within the network. Overengineering at the application level might be obviated if there were better mechanisms within the Internet to support these kinds of applications. Similarly, ISPs struggle to adapt to traffic dynamics and outages in order to satisfy contractual commitments made in service level agreements [**?**]. Improved in-network functionality could be harnessed to enable the network to measure and adapt to changing network traffic patterns, for example by adjusting routes or forwarding priorities. The key challenges in this approach include functional specification and implementation, and how the resulting router-based capabilities can be presented to ISPs, application designers, researchers, and other stakeholders in a simple, flexible, safe, and scalable fashion.

We propose a new approach for enhancing the functionality of routers, toward the goal of enabling the development of future Internet applications and services. Our main goal is to provide programmable mechanisms on routers and similar Internet devices to enable service providers and application developers to harness capabilities within the Internet. Such an approach poses many challenges and is potentially of high risk, since there are serious security and complexity implications and the potential to negatively impact traffic flows. Nevertheless, the thesis of our proposed research is that exposing in-network functionality has

tremendous potential to benefit applications and services that, in essence, *require* an in-network perspective for correct and accurate functioning, *e.g.*, network measurement, real-time, and security applications.

To address our main goal, we propose a framework for expanding inherent capabilities of routers that is based on identifying a set of *functional primitives* that are accessed through a domain-specific programming language. We define the requirements for our proposed framework as follows:

1. that it provide *flexible* access to router capabilities, including the abilities to modify packets in flight, emit new packets, query and modify router state, and programming constructs that facilitate construction of a broad range of applications;

2. that the performance impact on routers be *isolated* and bounded, such that any computational or memory requirements of the functional primitives be statically analyzable and/or dynamically monitored and controlled, with limited and measurable interference to basic packet forwarding capabilities;

3. that it be *safe*, in the sense that access to the primitives is enabled only for authorized users or delegates, and that router mechanisms cannot be harnessed to launch network attacks; and

4. that use of the framework not impose too heavy a burden of complexity, and be as *simple* as possible while still enabling development of rich in-network functionalities.

We propose to apply our framework toward defining primitives to address specific problems in network measurement, real-time applications, and network security. In these contexts, our aim is to design router-based mechanisms that enable improvement of the performance, management, and usability of the Internet for ISPs, application designers and users, researchers, and other stakeholders. While we intend to focus on specific problem areas as a way to analyze the operational capabilities, performance, and potential of our primitive-based approach, our objective is to design the framework and primitives to be applicable to a broad class of problems in networking.

The ability to control and modify in-network functionality by a variety of stakeholders has great potential to revolutionize the way that we think about designing Internet applications and services. For example, this capability could be used to radically change the kinds of measurements that can be collected from Internet routers. Rather than forcing network operators to adapt their analysis techniques to the measurements available from routers, the measurement facilities themselves could be reprogrammed to dynamically adapt to observed traffic patterns in order to provide operators with contextually useful information. They could provide accurate, detailed diagnostic information for *ex post facto* analysis of network events, and adaptive online information for emerging security threats. Not only could expanded functionality improve network operations tasks for ISPs, application designers could work in concert with ISPs to embed network support for applications requiring features beyond a best-effort packet delivery service, such as emerging real-time medical applications which may require strict quality of service assurances. Moreover, our framework could provide networking researchers in an experimental setting such as the Global Environment for Network Innovations (GENI) [**?**] the ability to quickly prototype new and potentially disruptive ideas and to deploy them at scale. We envision our work as having the potential to provide operators with better tools to monitor, manage, and troubleshoot their networks, to enable an Internet that better balances applications with different service demands, and to create an Internet in which attacks can be quickly traced and silenced.

The research initiatives described in this proposal are aimed at developing technology for routers and network devices for advancing the future Internet. The specific research goals of our work are as follows:

- **Goal 1:** Design and evaluate a programming framework for expanding router functionality and interacting with router subsystems, including a runtime model, an integrated security model and tools and methods for analyzing and controlling program resource consumption.

- **Goal 2:** Define and analyze a set of functional primitives appropriate to the application domains of network measurement, real-time applications including games, and network security.
- **Goal 3:** Design, build, and evaluate a reference implementation of the framework with the NetFPGA [**?**] as a target platform.
- **Goal 4:** Deploy the reference implementation in the GENI experimental testbed in order to evaluate the framework in a live and realistic setting and to inform future development of this and related research.

We intend to focus our research efforts in the following two areas:

- **A framework for expanding router capabilities based on functional primitives.** Efforts in this area will first focus on the design of the language, runtime, and security framework for our proposed system. In our preliminary work, we designed a basic prototype of the language and runtime in the context of the Click modular router [**?**], however many challenges remain. For example, how should users of this framework gain access to router hardware resources, namely memory, computational resources, and network bandwidth, and how should access be arbitrated across multiple users? Similarly, we intend to investigate appropriate analytic, programming language, and system-level techniques for monitoring and controlling program resource consumption, both statically and on-the-fly.

  There is a symbiosis between the framework design and development of new primitive functions in routers. Thus, in parallel with the design of the framework, we will develop a library of functional primitives that can be used in network measurement, real-time applications, and network security and accessed through a domain-specific programming language. In prior and preliminary work, we examined a basic set of network measurement primitives [**?**, **?**] and primitives to support real-time applications [**?**]. We propose to continue and expand this effort by including measurement-based network security applications that could benefit from direct network support, such as port scan and heavy-hitter (flooding) detection [**?**, **?**, **?**]. A related challenge that we intend to address is how to compose services across a network of routers. Our initial work has focused on a single router, but security applications in particular could benefit greatly from coordinated functionality across a network.
- **Design, deployment, and evaluation of a reference implementation.** Efforts in this area will focus on developing and evaluating an implementation of our framework on the NetFPGA programmable hardware platform [**?**]. A prototype implementation has been initiated using the Click modular router [**?**] in our preliminary work [**?**]. Our experience with this initial implementation suggests the promise of our approach, but has many limitations, and its performance does not permit evaluation in a setting with realistic traffic demands. We propose to design, develop, and evaluate a NetFPGA or network processor-based reference implementation of our framework and primitive library. This effort can proceed as aspects of the framework are defined, thus there is likely to be significant overlap between the design of the framework and development of our reference implementation. We plan to make our hardware functional design (*e.g.*, in Verilog HDL) available to the research community (note that there is no intent to develop new hardware for the proposed research). Next, we plan to deploy and evaluate our reference implementation in the GENI shared research infrastructure. We intend for this deployment to enhance existing programmable infrastructure in GENI, affording researchers a new set of malleable in-network capabilities. We also intend to use our deployment in GENI as a key venue for performing experiments to analyze the performance, security, and utility of our approach.

The PI has had extensive experience in the Wisconsin Advanced Internet Laboratory (WAIL) [**?**] with a variety of networking equipment while designing and carrying out experiments as a graduate student and

more recently as a remote user of WAIL. The PI's strengths in designing, developing, and evaluating systems in a hands-on laboratory setting will also be brought to bear on the proposed project.

The rest of this proposal is organized as follows. In Section 1.1.2 we describe our prior and proposed work in developing a primitive-based framework for expanding router functionality. In Section 1.1.3 we describe our accomplished and proposed work toward creating and evaluating a reference implementation of our primitive-based framework. In Section 1.1.4 we discuss related work. In Section 1.1.5 we elaborate on the expected impact of our proposed research. In Section 1.2 we describe our educational plan and the opportunities to enhance undergraduate research and education. Finally, Section 1.3 summarizes results from prior NSF support.

1.1.2 Framework for Expanding Router Functionality

The idea of introducing programmability into network devices, *i.e.*, methods for *changing* functionality rather than configuring existing functionality, has been of interest to the networking research community for some time. While modern routers have significant built-in capabilities, these functions are generally fixed in the firmware. Research in the area of "active networking" in the late 1990's and early 2000's sought to overcome this limitation, with the goal of enabling ISPs, researchers, and other interested parties to modify the ways in which packets were treated by routers, and to enable access to router-based computation, storage, and other mechanisms [**?**]. While not the exclusive focus of prior work, a major focus was on treating packets themselves as containing program text (also referred to as "capsules") [**?**].

Much was learned from this prior work, and it can be seen as having set the stage for the current generation of network processor- and FPGA-based systems, *e.g.*, [**?**, **?**, **?**, **?**, **?**]. Still, the fundamental problem that active networking research attempted to address remains unresolved, namely, how can computation and memory resources within the network be more effectively utilized for developing next generation Internet services? Moreover, the problem of *ossification* of the Internet infrastructure has continued to loom large [**?**], but researchers still lack any mechanisms for affecting router functionality in the core of the Internet.

We propose a new approach for enhancing router functionality toward the goal of enabling new services and applications in the Internet. We intend to focus our efforts on developing a framework for running programs on routers that is both powerful and deployable.

More specifically, in [**?**] we propose an approach for enabling programmability of routers that focuses on the functional requirements of different application and service domains such as gaming, network measurement, and security, and decomposes these requirements into a set of *primitive functions* that could be built into routers. In preliminary work [**?**], we describe how these functional primitives are accessed through a new router primitive-aware programming language that we developed called *Morpheme*. Our objectives in the design and development of Morpheme are to create a language that is efficient, easy to use and easy to analyze. We accomplish this through a prototype implementation that reflects the primitive functions directly and that has a minimal set of capabilities beyond the invocation of primitive functions.

There are four components of our framework for enabling programmable functionality on routers:

1. *Functional Primitives:* We specify a set of primitive functions that extend the basic capability of a router. While some of these functions may already exist on some commercial routers, we assume that in many cases they will require reengineering. Our process for defining primitives focuses on specific application and service domains, resulting in capabilities that are both effective and potentially attractive for developers and network operators. We consider well-known examples within each domain and then identify how sources of uncertainty or complexity can be accommodated by in-network functions. Our objective is to keep the primitives conceptually simple, and thereby keep implementation feasible.

2. *The Morpheme Programming Language:* The Morpheme programming language coordinates and gives access to the functional primitives. Our objectives with Morpheme are to develop a language that is

expressive, easy to use, and easy to analyze. The language syntax that we define relates directly to the primitive functions and how they can be parameterized. We also include constructs that enable simple but important programmatic operations such as loops, along with network-specific functions such as generating, inspecting, and modifying packets. An additional design objective is to make the language restrictive so that programs can be analyzed effectively and will run on systems with limited resources.

3. *Runtime Environment:* The main tasks of the runtime environment are to enable Morpheme programs to run efficiently and correctly within the context of a running router. There should be no impact on data plane traffic, and minimal impact on traffic processed by the Morpheme runtime (beyond what the program is intended to accomplish). Specifically, the runtime must handle resource sharing among multiple programs, admission control to avoid overcommitting resources, and resource arbitration to handle any feature interactions of primitives.

4. *Router Access Control:* In our router access control model, programs can only be loaded and run on routers by users who have the proper authorization. We assume that user credentials are established in an offline fashion. A properly credentialed user will be granted access to some set of routers in a network. The user will download programs onto those nodes, which will begin executing immediately. The user will have the opportunity to stop programs and load new programs as desired.

1.1.2.1 Router Primitives and Morpheme Language

In our work to date, we have focused on two target application domains: real-time applications (*e.g.*, games and VoIP) and network measurement. There are essentially two types of primitives in our design: *events* and *actions*. The basic idea is that an action, or set of actions, can be associated with the occurrence of an event. Invoking an event or action is specified in terms of Morpheme statements.

Events trigger the execution of Morpheme code blocks which contain action primitives. Events may be associated with a *timer expiry* or *packet receipt*. Primitives exist for setting timers that will trigger the execution of Morpheme actions.

Besides timers, an action or sequence of actions can be associated with the arrival of a packet that matches certain criteria. In our work thus far, we assume that all packets pass through the Morpheme runtime environment. While this assumption is convenient for our initial implementation, it should not be considered necessary (or perhaps even desirable). More generally, one could consider a mechanism external to Morpheme (*e.g.*, an access control list) that that could perform the necessary function of selecting packets with specific characteristics for Morpheme processing.

Action primitives form the core functionality of Morpheme. In our preliminary work we have considered primitives useful for both real-time applications and for network measurement.

Target Domain 1: Real-time Applications. The key requirement for real-time applications is for predictable performance. This typically translates into ensuring low delay and loss in the face of uncertain competing traffic demands.

- **Traffic classification and marking.** A `field` primitive is provided to enable access to arbitrary elements in a packet's headers or payload. A user of the `field` primitive is required to specify the header name (or `payload`) and a field mnemonic (or the size of the data to access, and its offset in the specified header or payload). The `field` primitive can also be used for modifying a packet's headers or contents. Thus, `field` could be used for basic classification of a packet as well as marking it, *e.g.*, modifying the IP TOS bits or ECN bits. For example, if we wanted to set the IP TOS bits to 0x10 ("low delay") for all UDP traffic, we could use the following Morpheme code:

```
isudp = field ip proto == 17
if isudp:
    field ip tos |= 0x10
```

- **Expedited forwarding.** We define a **forward...queue** primitive to specify a numeric output queue to which a packet should be directed and how the packet should be queued (at the head or tail of the queue). For example, the following would forward all UDP packets to the head of the default output queue (defined as queue 0):

```
isudp = field ip tos == 17
if isudp:
    forward next-hop queue 0 head
```

- **Explicit discard of packets.** For low-priority flows or for flows that are excessive consumers of bandwidth, we may wish to explicitly drop packets; a **drop** primitive is provided for that purpose.

- **Timestamp manipulation.** Monitoring bandwidth consumption of flows is useful for ensuring superior performance for real-time traffic (and for possibly degrading the performance of low priority traffic). In order to evaluate bandwidth consumption over time, the ability to request the current time is necessary. A straightforward **now** primitive is provided for that purpose.

With the above primitives, along with other basic Morpheme features, a basic rate-limiting application can be created. The following Morpheme program attempts to cap all TCP traffic to 500 kb/s by periodically computing an exponentially weighted moving average of TCP bandwidth consumption. The moving average is recomputed every ¼ second. If the rate exceeds 500 kb/s, all traffic during an interval is dropped. A more sophisticated implementation might mark ECN-capable flows with the CE bits or discriminate more finely. Such functionality could be easily implemented Morphemically.

Listing 1: Simple rate-limiting application in Morpheme.

```
; attempts to cap all TCP traffic at 500 kb/s.  due to periodic computation of
; traffic rate, more than 500 kb/s may be allocated over short durations.
ratecap = 500000.0
rateewma = 0.0
lastts = now
alpha = 0.1
istcp = field ip proto == 6 ; extract the protocol and test
plen = field ip len         ; extract the packet length
if istcp:
    if toohigh:
        drop                ; summarily dismiss the packet
    else:
        bytes += plen
periodic 0.25:              ; periodically recompute EWMA traffic rate
    ts = now
    ratenow = bytes / (ts - lastts)
    rateewma = ratenow*alpha+rateewma*(1-alpha)
    toohigh = rateewma > ratecap
    bytes = 0
    lastts = ts
```

Target Domain 2: Network Measurement. For network measurement, the key requirements are to be able to create and send probes for active measurement, to annotate packets that are forwarded through the router with passive measurement data, and to have a minimal impact on other data plane traffic if desired. We define the following set of primitives for use with network measurement tasks.

- **Create and send a packet.** The **probe** action is used to construct and send a new packet. Parameters to this primitive can be used to specify the header and payload contents:

```
probe 10.10.2.1 udp dport 3000 payload {42/4B}
```

  The above example creates and sends a UDP packet with a particular destination address and port, and with a payload consisting of the number 42 stored as a 4-byte quantity.

  Using the **probe** primitive along with a timer, a wide range of active network measurement algorithms can be implemented in Morpheme. For example, we experimented with an implementation of the somewhat sophisticated Badabing active packet loss measurement algorithm [**?**] in our prior work [**?**].

- **Annotate a measurement probe.** Another measurement primitive enables annotation of packets with passive measurement data as they are forwarded through a router. For example, high-resolution timestamps could be added to a packet both on ingress and egress from a router. Interface addresses could also be added to a packet. More generally, any passive measurement data that is readily available (e.g., SNMP MIB variables) could be added to a packet as it is forwarded through a router. We define a set of primitives for each of these functions:

```
input-timestamp
output-timestamp
input-address
output-address
input-mib octets
output-mib octets
```

  In the example above, packets are annotated with timestamps, interface addresses, and the `ifHCInOctets` counter (from the Interfaces Group MIB [**?**]) on both ingress and egress.

- **Conditional packet forwarding.** The **forward...when** action can be used to specify that a probe should be held until a condition involving the output queue becomes true. For example, the follow statement specifies that a packet should be forwarding once the output queue becomes less than ½ filled:

```
forward next-hop when outputqueue < 0.5
```

  This statement implies that additional buffer space and processing is needed at router egress. Packets that are processed with **forward...when** are held until the specified condition becomes true. Because buffer space is finite, packets held awaiting a **when** condition may be eventually dropped.

  Interestingly, using **forward...when** could be used to implement a record-route feature that has minimal impact on other data plane traffic:

```
input-address
output-address
forward next-hop when outputqueue < 0.1
```

  Note that the destination specified in the **forward** statement can be a specific IP address rather than the default **next-hop**. Using this feature could enable a user to send a packet out an interface that is not on the standard forwarding path, thereby enabling active measurement of arbitrary links and paths. An example application of this facility could be to test router ACLs by forwarding probe packets to a router that should drop them, and testing whether the packets are indeed dropped.

Proposed Research Activities In our work to date [**?**, **?**], we have identified a small set of primitives for real-time and network measurement domains, along with a minimal set of language features to be able to create useful programs. Many challenges remain to make our set of primitives useful beyond these domains, and to include language features that enable more sophisticated applications.

1. Evaluate the existing primitives in the real-time and measurement domains, and design new primitives to enable network security applications, *e.g.*, primitives to match patterns against incoming packets while leveraging router hardware capabilities.

   Furthermore, we intend to investigate the question of how to develop criteria and mechanisms to evaluate the "completeness" of a given set of primitives. A related issue is to develop criteria to evaluate the complexity of a given primitive, and to evaluate performance and functionality tradeoffs with different primitives. We will consider both simulation and analytic modeling approaches to these questions. These issues are important for static resource consumption analysis and for limiting the complexity of Morpheme as a whole.

2. Design a set of primitives to enable access to control plane functions. For example, in our set of target application domains, determining the control plane-computed route of a given packet may be important for determining how to process the packet. Moreover, with appropriate control plane primitives, a Morpheme-based routing protocol could be deployed, *e.g.*, the Pathlet routing proposal [**?**].

3. Examine and evaluate primitive mechanisms to enable coordination of programs running on multiple routers. We will consider simulation as an evaluation approach in this task.

4. Design new language features such as built-in data structures like lists and associative arrays. Moreover, Morpheme currently lacks functions and other abstraction mechanisms for developing more complex programs. We will continue working with collaborators in the programming languages community to develop the capabilities of Morpheme.

5. Investigate, develop, and test higher-level abstractions or environments to enable building complex programs. For example, we might consider a visual Scratch-like environment [**?**].

1.1.2.2 Runtime Environment

The runtime environment is mainly responsible for executing Morpheme programs efficiently, allocating and monitoring program resources, and interacting with the rest of the running router. In our initial execution model, for each primitive event that is fired, a virtual thread of execution is spawned to process the sequence of action primitives.

The set of action primitives will be designed for safety with respect to memory and CPU resource usage, precluding arbitrary-length or infinite loops and disallowing access to arbitrary memory locations. The goal is for the resource demands (CPU and memory) of any primitive to be statically analyzable (*i.e.*, offline); this allows the use of an admission control procedure for accepting requests to install Morpheme code.

Accounting is an important capability of the runtime environment. For example, traffic generation quotas should be supported as well as processing time quotas. Practically, it may be very difficult to support *hard* limits, *i.e.*, ensuring that no limits are ever exceeded, across an entire network. On the other hand, *soft* limits, in which a user may temporarily exceed a threshold, may be possible and still provide the key benefit of rapidly detecting user errors or deliberate misuse.

Some amount of memory will need to be reserved for packets held due to `when` clauses; a conservative maximum can be placed on this amount of memory in order to bound this cost. A control interface to query and/or flush packets held due to `when` clauses would provide the ability to test whether any probes are being held pending a condition becoming true. Furthermore, we note also that memory costs related to the `store` action would likely be far less than is used for current passive flow measurement capabilities in routers.

Depending on traffic conditions, there may be situations in which packets waiting on a `when` condition consume memory resources for an extended period of time. In such situations, there may not be enough storage for an arriving packet that requires `when` processing. Generally speaking, the program responsible for

the probes should be quickly notified, and may need to be paused or stopped while the situation is resolved. We have not yet defined the mechanisms for performing such error notification and exception handling.

Proposed Research Activities We propose to design and evaluate a runtime environment. This activity will be challenging since the runtime forms the boundary between our proposed framework and an existing router environment. Our main evaluation methods in these activities will be simulation and testing of limited prototypes within the context of the Click modular router [**?**]. Specifically, we propose to:

1. Design and evaluate runtime algorithms to track and control static and dynamic resource consumption. We also plan to investigate the potential problem of feature interaction among primitives, *e.g.*, if primitives can change the state of the router, how should other router configuration or monitoring mechanisms be made aware of these changes?

2. A simple design assumption could be that programs will have access to all traffic flowing through a node. However, in a live setting this will likely not be the case for security, privacy, and scalability reasons. We intend to investigate the tradeoffs between full and limited access to packets, and to design and evaluate runtime mechanisms for enabling a scalable and robust runtime environment.

3. We propose to develop algorithms and runtime mechanisms to convey exceptional program state to the users affected. For example, should a user be immediately notified using, *e.g.*, a mechanism similar to an SNMP trap? If not, should some other method be used to propagate exceptions?

1.1.2.3 Security and Access Control

The router primitives available through our framework should not be accessible to users without proper credentials. Furthermore, an ISP may wish to allow some users access to certain primitives but not others; the ISP should have the flexibility to assign different rights to users. For example, the network operations center for a friendly peer ISP may be given access to certain primitives in order to troubleshoot customer problems. However, use of all primitives might be disallowed so that the peer cannot gain "too much" information.

In order to gain access to a router for adding or removing an event trigger and associated actions, a user must obtain a *ticket* from a central authentication and authorization broker. The ticket has a limited lifetime and is encrypted using a secret key that is shared between the auth system and given router, *i.e.*, the user cannot modify the ticket. The ticket includes timestamp information (*i.e.*, when the ticket expires), a unique user identifier, the source address for the user, a unique identifier for the router, a *static capability* set and a *dynamic capability* set. The unique user identifier can be used in the probe packet to allow demultiplexing of probe packets on arrival at a router and to match an incoming probe to its associated event actions. The static capability identifies the action and event primitives that the user is allowed access to for the router. For example, a user might not be given access to actions that enable gathering of MIB data, or to specify an address other than `next-hop` for the `forward` action. The dynamic capability contains information regarding quotas and other limits that must be monitored over time by the runtime environment for adherence. For example, a user might be limited to some maximum bandwidth for probe traffic.

A user must obtain separate tickets for each router in which the user wishes to install events and actions. Because a ticket contains the user's source address and a router identifier, it cannot be transferred to another user or used with a different router. The central broker must keep track of a user's currently active tickets in order to possibly issue any capability revocations or to deny issuance of a ticket in the case that a user already has exceeded some threshold of outstanding tickets. Each router with events and actions installed for a given user must maintain accounting information such as processing and memory usage in order to monitor dynamic capabilities and also to monitor system-wide usage of the measurement primitives. If, in processing user actions, a dynamic capability threshold is exceeded, the router informs the central broker

which issues ticket revocations to any affected routers. Events and actions for the user are then temporarily disabled. User access may be automatically re-enabled when the ticket expires, or upon manual intervention by an administrator. Note that each router must maintain a ticket revocation list so that any future attempts to use the ticket (within its lifetime) are disallowed.

Fine-grained capabilities allow the possibility to create, *e.g.*, policies that allow a friendly peer ISP limited access for collecting some types of measurements in order to troubleshoot problems. VPN customers could also be delegated limited access in order to measure performance across a provider's network and to verify compliance with SLAs. Events and actions could be installed for anonymous users in order to enable some types of measurements by outsiders. If too many probes were observed, the dynamic capability processing could cause the access to be temporarily disabled.

Proposed Research Activities Much work will be required to develop and evaluate the security and access control features of our framework. Our main evaluation methods in these activities will be analytic modeling and experiments with limited prototypes. Specifically, we propose the following.

1. Develop a threat and impact analysis model to evaluate the security implications of the primitives and runtime mechanisms.

2. Design and develop a prototype for gaining credentials for accessing Morpheme-enabled routers, including a static and dynamic capability set for evaluation by the runtime environment.

3. Investigate and design delegation and credential proxying mechanisms.

1.1.3 Development and Evaluation of a Reference Implementation

The second focus of our research is to develop, test, and carry out experiments with a reference implementation of our framework. There are two aspects to this work: (1), design, development, and testing of a reconfigurable hardware-based reference implementation, and (2) deployment and experiments with our system in the GENI testbed.

In our preliminary work [**?**], we implemented a minimal prototype runtime environment for Morpheme in the Click modular router [**?**], and evaluated it using a set of microbenchmarks. The functionality of the Morpheme runtime is split across two elements within our Click implementation. Figure 1 indicates how our two Morpheme elements might fit into a Click configuration (note that the figure does not show a complete Click configuration, it is merely shown for illustrative purposes). The key element is called `MetaMorphic` and implements the bulk of the functionality required to support the various Morpheme primitives and language features. Timer handling (to support **`after`** and **`periodic`**) is handled here, as is probe creation and emission, field access and marking, and ingress passive data annotation (*e.g.*, input timestamps or input MIB data).

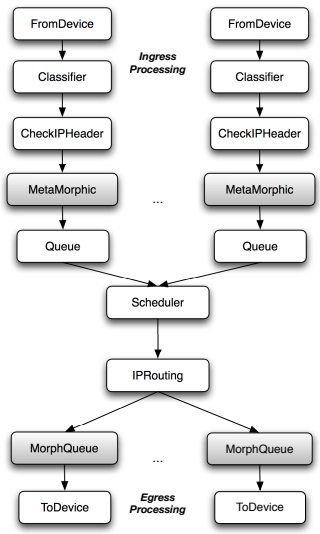

Figure 1: Click element flow in Morpheme prototype. Elements used in the Morpheme runtime are shaded.

All functions that are logically related to router egress are *preprocessed* in the `MetaMorphic` element. In essence, metadata are added to a packet to indicate that it needs a certain type of egress processing by the Morpheme runtime. For example, metadata are added to indicate any passive data that need to be written to the packet on egress. The metadata are added so that a back-end Morpheme element (`MorphQueue`) can perform any remaining run-time tasks prior to packet egress. We note that any passive data to be added

to a packet in `MorphQueue` are written just before the packet is passed on to be sent to the network. Thus, timestamps and other data are applied as close as possible to the packet leaving the router.

A side-effect of our division of front-end and back-end processing implies that the ordering of statements in a Morpheme program may not be the same order in which the primitives are executed within the runtime. For example, it is valid to specify **`output-timestamp`** before **`input-timestamp`** in a Morpheme program. Within the runtime, the first call to **`output-timestamp`** would result in the addition of metadata to the packet. The second call would result in an ingress timestamp added to the packet. The output timestamp would be added only *after* the packet had been forwarded to the back-end `MorphQueue` element.

Besides application of egress passive measurement data, processing of **`forward...when`** and **`forward...queue`** statements are handled in the `MorphQueue` element. Metadata are added to a packet in `MetaMorphic` to indicate the type of back-end processing that needs to take place. A fixed-size circular queue is used to store packets waiting for a *when* condition to become true. As the packets are added and removed from the queue, the Boolean functions associated with held packets are executed. If the function returns true, the packet is released and forwarded. If the queue is filled and a new packet arrives for *when* processing, the packet at the head of the queue is dropped to create space for the new arrival.

Proposed Research Activities The limited prototypes designed and evaluated in the research activities described in Section 1.1.2 will inform the design, development, deployment, and testing of a complete reference implementation based on the NetFPGA platform. Specifically, we propose to:

1. Design, develop, and evaluate a full NetFPGA-based reference implementation. Experiments will initially be performed in a controlled laboratory setting and will focus on micro- and application-level benchmarks and inform future development of our proposed system. We expect this work to be a significant milestone toward demonstrating and evaluating the potential of our proposed framework. The hardware functional specifications for our reference implementation will be made openly available, which will facilitate use in commercial systems.

2. Deploy and perform experiments with the reference implementation in GENI. The GENI testbed environment represents an excellent opportunity to evaluate the reference implementation in a live network, and to make our system available to others. Experiments will focus on research questions specific to the three application domains studied in this proposal, *viz.*, network measurement, real-time applications, and network security.

3. Create and test a library of applications that use the reference implementation for users to learn from and to modify for their own purposes. This library will be made openly available and will facilitate sharing of applications by researchers.

1.1.4 Related Work

Our work is informed and inspired by prior research on *active networks*. These studies investigated the idea of enabling customized computation on packets as they flow through network devices such as switches and routers [**?**]. A key objective of active networks was to enable new applications and new capabilities in existing domains such as network management. Our objectives are similar. Two conceptual similarities are the notion of "primitives" in network devices that can manipulate packets [**?**], and the PLAN programming language for active networks [**?**]. However, the context for both of these ideas was primarily focused on "capsule packets" which carry program fragments (in essence, the packet is the program) that are executed by active network routers and can dynamically extend router and network functionality. In contrast, in our approach packets do not carry program fragments, and only built-in primitives can be accessed by Morpheme programs. Finally, Gao *et al.* describe a programmable router architecture designed to extend control plane

functionality in [**?**]. While this study is also conceptually similar to ours, our focus is on defining specific primitives in both control and data plane contexts.

There have been a number of studies on programming environments for network devices. These studies largely fall into two categories: software development kit (SDK)-based approaches and entirely new programming languages. SDK-based environments enable functionality available in the devices to be manipulated through application programming interfaces [**?**, **?**, **?**]. The primary advantage of this approach is that it enables programs to be written in languages such as C or Java, which are familiar to a large number of users and have well-known development tools. Recent work by Davie and Medved describes an API for creating an overlay to enable service providers to add new functionality to the network and experiment with it in a flexible fashion [**?**]. Built-in router functions are key to enabling this service layer which is accessed through APIs using web-based services.

A number of entirely new languages for programming network devices have been developed, typically with the goal of simplifying the operation of specialized devices (*e.g.,* [**?**, **?**]. While many of these languages have limited scope, an excellent example of a general purpose language is *nes*C [**?**]. *nes*C was designed for use in sensor network environments on tiny devices with severe resource constraints. Morpheme has some similarities to *nes*C in terms of our interest in efficiency. However, our approach is different since it includes defining primitive functions for routers to potentially support a broad range of applications.

The notion of including specialized primitive functions in routers has been an important focus of vendors for many years. Many of these functions have been focused on improving the manageability or performance of these devices as opposed to application support. A quick review of the data sheets for several network processors [**?**, **?**, **?**, **?**] shows built-in support for queue management, filtering, crypto, a variety of interfaces, etc. For example, in commercial routers today, there exist various built-in functions to facilitate both passive and active network measurements. While a range of configuration options are available for these facilities, there are no *programmable* capabilities. For active measurements in particular, there are facilities to generate probes according to preset emission processes, and perform basic statistics on these measurements [**?**, **?**]. While our focus is also on router capabilities, we propose a set of primitives that can be composed by a user to achieve a much wider variety of application objectives.

Also in the area of network measurement, there have recently been a number of proposals for programmable *passive* measurement systems. Ramaswamy *et al.* [**?**] describe a programmable passive measurement system based on network processor-based systems deployed in a network. More recently, Yuan *et al.* describe the ProgME system which is based around the notion of capturing *flowsets* (arbitrary collection of flows) [**?**]. In [**?**], Khan *et al.* examine an FPGA-based *query-driven* collection engine that can be reprogrammed based on user/application requirements. Our focus is different in that we propose a set of primitives to extend router capabilities rather than on capabilities of an external passive measurement system.

1.1.5 Impact of the Proposed Research

Our proposed research has three main areas of impact.

1: De-Ossifying of the Internet. The concepts, techniques and results from the proposed research will provide new in-network capabilities to network operators, application developers, and researchers. Presently, the lack of this type of capability has led to stagnation in the protocols deployed in the Internet, the inability to deploy "disruptive" research technologies, and the necessity for developers to use overlay facilities and employ algorithmic tricks to obtain acceptable application-level performance.

2: Enabling New Kinds of Internet Applications. In a similar vein, research enables researchers, operators and application developers to envision and develop entirely new ways of using and understanding the Internet. Other researchers could use our techniques to measure characteristics of the Internet that are presently poorly understood. Application developers could better control and use Internet resources to provide better user experience and more efficient use of the Internet.

3: Enhancing Shared Research Infrastructure. The proposed deployment of our reference implementation in the GENI testbed enables researchers to test ideas that require new or different functionality from routers. The existence and deployment of our framework in GENI will help to spur new ideas that might otherwise not be pursued. We will develop a library of applications that use our system and hold tutorials at GENI Engineering Conferences to make the research community aware of this capability and to promote its use as a platform for conducting research that requires changing router functionality. We intend also to build a technical community around sharing Morpheme applications.

1.1.6 Schedule and Milestones

The proposed research will be conducted over a five year period. Undergraduate students employed during the academic year and during the summer will be closely managed to perform significant portions of the proposed work. Pairs of students will be employed in order to facilitate continuity and shared knowledge. The PI will assist students to work on implementation and testing-related activities, while still engaging them in the broader questions of the proposed research. The schedule and milestones for this project are as follows:

- **Year 1:** *Framework:* Perform design work in the primitives, new language features, runtime capabilities, and security model, focusing on the primitives and language features. Evaluate and test the design through limited prototypes, simulation, and modeling.
- **Year 2:** *Framework:* Continue design and evaluation work, focusing especially on the runtime and security capabilities; investigate control plane primitives. *Reference implementation:* Begin design work for FPGA-based implementation.
- **Year 3:** *Framework:* Investigate coordination primitives; complete design and evaluation of framework. *Reference Implementation:* Continue design of FPGA-based implementation, begin development of basic runtime environment, design laboratory benchmark tests.
- **Year 4:** *Framework:* Examine higher-level design tools to enable development of primitive-based programs in simple, perhaps visual, manner. *Reference Implementation:* Continue implementation of reference implementation and perform benchmark tests. Develop experiments for GENI-deployed system. Hold a GENI tutorial to introduce researchers to the capabilities of the present and eventual system.
- **Year 5:** *Framework:* Complete and evaluate higher-level design tools. *Reference Implementation:* Deploy and perform experiments with reference implementation in GENI. Develop a library of example applications for users to learn from, extend, and contribute to. Hold additional GENI tutorial.

1.2 Educational and Outreach Plan

The educational component of this proposal is tightly integrated with the proposed research, emphasizing enhancement of undergraduate education and research. Indeed, undergraduates will be actively involved in our proposed research. The objectives of our educational plan are to engage undergraduate students in research, to enhance undergraduate networking curriculum, and to disseminate our enhancements to a wider audience.

1.2.1 Undergraduate Research and Curriculum

Hands-on laboratory-style activities can be fun and motivating learning experiences. Especially for undergraduate students, until they can *do* something with their knowledge, that knowledge remains lifeless

and somehow unreal. We propose two directions for developing and disseminating enhancements for an undergraduate curriculum, described below.

1. Reconfigurable hardware platforms such as the NetFPGA have gained popularity as an implementation platform to test research ideas in a very real way. As a result, a growing community has formed around this hardware to contribute software and courseware [**?**]. Unfortunately, much of the existing course materials are generally geared toward graduate students or students with significant engineering background [**?**]. As these platforms continue to rise in popularity, it is important to make them accessible to a wider range of students, including undergraduates are smaller institutions, *e.g.*, liberal arts institutions. We propose to adapt existing educational materials and to tailor them for students in a small college setting. Although students in such a setting may not be able to build a fully functional Internet router in one semester [**?**], there are more modest projects that could provide critical learning experiences with this platform. We intend to create additional assessment activities (besides development projects) that enable potential adopters to more easily make use of our work. Our materials will be made openly available to the community.

   A related and important aim of this effort is to build expertise in undergraduates at our own institution in order to enable students to significantly contribute to the development of our reference implementation and other aspects of the research project described above. We plan to assess the effectiveness of our modified educational materials in helping students to learn the NetFPGA platform and to be able to apply their knowledge in a meaningful way. Our hope is that this feedback will enable us to enhance the materials and activities so that others can take advantage of our improvements.

   Timeline: Since later development work will greatly benefit from having local student expertise, we will initiate this effort in Year 1, and expect it continue into Year 2.

2. Our proposed framework for enabling new functionality of routers opens opportunities for new types of laboratory experiences for students in which they can modify the behavior of a router. Thus, our second aim is to develop a set of laboratory modules for students to learn how to make and deploy substantive changes using Morpheme. For each laboratory module we will develop learning goals, learning activities, and assessments to help make them more easily adoptable by others. The assessments will also enable us (and others) to determine how well students learn and apply networking concepts compared to other learning activities. Examples of modules that we intend to develop are (1) to design and develop a routing protocol, *e.g.*, XL [**?**], (2) to implement the network-assisted XCP congestion control algorithm [**?**], and (3) to implement a passive measurement capability similar to Cisco's Netflow [**?**]. Our materials will be geared toward an undergraduate networking course, but could easily be adapted for an introductory graduate level course.

   To reach beyond our own students and institution, and to further the goals of our educational plan, we plan to make our laboratory modules openly available to the community. We also intend to hold a workshop at an ACM SIGCSE conference or similar venue in order to make other educators aware of our work, and to provide them with background knowledge and some experience with using our materials. Assessment activities will be created and implemented in order to determine the success of the workshops and educational goals of our laboratory modules and other materials.

   Timeline: Since these laboratory modules are dependent on a fully functional prototype (or the full reference implementation), we will initiate this effort in Year 4, completing and disseminating this work by the end of the project in Year 5.

1.2.2 Impact of the Proposed Educational Plan

Our educational plan is designed to have the following effects:

1: Engage undergraduate students in research and the research community. Our research project engages students directly in research activities, and in presenting their work at research conferences. Students will also be involved in carrying out the proposed tutorials and workshops. Since all our materials will be openly available, students at other small institutions will be able to engage in research-like activities. Our hope is that these efforts will help to excite students and prepare them to apply to top graduate programs.
2: Modification, development and dissemination of effective teaching and learning materials. Our proposed modifications to the NetFPGA learning materials and our laboratory modules will be made openly available to the community. The proposed materials will make reconfigurable hardware technologies more accessible to institutions that do not traditionally have strong (or any) engineering focus. Our proposed materials will also assist in incorporating research ideas and methods into undergraduate teaching. We note that there are hundreds of small colleges that could potentially take advantage of these materials.

1.2.3 Summary of Prior Teaching Accomplishments

Since arriving at XYZ University (Fall, 2007), the PI has taught the CS1 course each semester, upper-level undergraduate networking and operating systems courses, and a CS0-type course in the liberal arts core curriculum. Over the past three years, the enrollments in our CS1 course have nearly doubled, and we have significantly increased the number of computer science majors. The CS1 course was recently revised to be more accessible to students who enter without any background in computer science (which is quite common at XYZ University). The networking and operating systems courses have been designed from the ground up, including laboratory and programming exercises and other activities.

Specifically for the CS0-type course, it was designed to introduce students to scientific practices from the perspective of computer science (citation removed) (in accordance to curriculum guidelines at XYZ). An important part of the effort has been to develop several in-class laboratory activities in which students learn about computer science by *doing it*. The key goal for these sessions is for students to learn to approach problem solving from a computational perspective [**?**,**?**]. An important design decision for the labs is that there is no direct programming involved. Instead, students work in pseudocode and bring other resources to bear (*e.g.*, a combination of net-accessible and custom software tools) when investigating particular phenomena and when attempting to formulate questions and hypotheses. Qualitative data have been collected thus far on student experiences and whether students feel that the lab exercises have contributed to their understanding of course material. We have been encouraged by the fact that student feedback has been overwhelmingly positive. Indeed, several students have since enrolled in the CS1 course. We believe, moreover, that our course structure could serve as a blueprint for institutions with similar curricular requirements.

1.3 Results from Prior NSF Support

(Removed for release as example proposal.)