A Programmable, Generic Forwarding Element Approach for Dynamic Network Functionality

Ran Giladi
Department of Communication Systems Engineering, Ben Gurion University
Beer-Sheva, Israel
ran@bgu.ac.il

Niv Yemini
Department of Communication Systems Engineering, Ben Gurion University
Beer-Sheva, Israel
nivye@bgu.ac.il

ABSTRACT

Communication networks are growing exponentially, and new services and applications are being introduced unceasingly. To meet the demands of these services and applications, current network systems have to be modified, replaced or supplemented. Various technologies, such as reconfigurable devices or active networks, have attempted to address this problem. In this paper, we introduce a programmable, generic forwarding element (GFE), which can be used as a platform for a flexible and reconfigurable network system. This platform and the resulting network system enable on-the-fly definition of adaptive and dynamic network functionalities, so that the demands of new services and applications can be met. Additionally, specific service instances or traffic flows can be handled by this platform on a temporary and locality basis, according to traffic patterns, application demands, and provisioning decisions. The proposed GFE complies with today's standards and can easily be adopted for future standards. A network processor is used to implement this platform, so that frame processing is achieved at wire speed, even though each frame is analyzed and processed by a meta-program. An XML-based definition of the forwarding element is used to describe frame processing, based on the frame contents and ingress port, and on various system and network parameters. [1]

Categories and Subject Descriptors

B.4.1 [**INPUT/OUTPUT AND DATA COMMUNICATIONS**]: Data Communications Devices;
C.2.1 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Architecture and Design.

General Terms

Design

[1]The research was funded by the European Community's Seventh Framework Program [FP7/2007-2013] under grant agreement number 215462.

Keywords

Network Systems, Programmable netwroks, Forwarding element

1. INTRODUCTION

Communication networks are growing exponentially, both in aggregate bandwidth and in number of users. (IP traffic has doubled approximately every 2 years and will probably surpass 16.8 Tbps by 2012 [1]. As of 2008, the Internet serves more than 1.4 billion users [2].) Researchers are therefore working intensively on new network architectures, the Internet of the future, and novel technologies for new services. New network applications, services and technologies are materializing at an ever-increasing rate and require different network behaviors and functionalities. Current networks (Internet, Ethernet, mobile networks, etc.) limit potential applications and services, since network systems – the building blocks of the networks (e.g., routers and bridges) – are designed to support specific, predefined sets of operations and standards, with predetermined functionalities. To support new services, a long process of standardization – followed by modifications of network systems – is required. It is impossible to shorten this process, since it involves both integration of many network entities (some of which do not even exist, and others are yet to be programmed) and massive investment in network equipment. Such a massive undertaking can obviously not be accomplished on the fly.

A flexible and dynamic network system could relieve the problem described above and provide an answer to the required redefinition of network systems and programming of network functionality. Various approaches have been proposed for flexible and programmable network systems. Among them, the Click Project [8] proposed a software architecture for building flexible and configurable routers and can be considered as a data path programming and virtualization tool on PC platforms [7]. Once programmed, however, the network system cannot change its functionality. Another approach – active networks – which was suggested as early as the mid-1990s [14], enables active, programmable network behavior by injecting code into the network. Various network elements then execute the injected code capsules [3] for per-frame processing and other computations for various specific tasks. Active networks are based on the use of general-purpose processors for executing the injected code and implementing the software re-configuration algorithms. Active networks are focused on programming arbitrary and complex tasks of upper layer functions, and the forwarding elements (FE) of the active network systems are not changed or redefined.

Separation of the management or control plane from the forwarding plane (or data-path) functionalities has been used as a basis for various programmable network systems. Such an approach was presented in the NEon project [11]: Entities in the control plane configure pre-defined tasks of the network system (e.g., shaper or forwarder) by policy rules and actions that define the operation of various network services of a NEon system on a per-frame basis. 4D [5] defined a network architecture with four planes: decision, dissemination, discovery and data: the data plane (e.g., switches and routers) forwards packets according to what the decision plane dictates and disseminates. Ethane [4], OpenFlow [10] and Forwarding and Control Element Separation (ForCES)[15] also separate management and control from forwarding functionalities and allow managers to remotely define network-wide policy in a central controller. Ethane and OpenFlow represent a popular approach that simplifies the data path and lifts out complexity into the control plane. As a result, however, a performance issue may arise, since complex forwarding functions have to be carried out in a general-purpose processor, rather than in a network processor that can nowadays cope with 10 to 100 G-bps wire speed processing.

Software [6] and reconfigurable [13] routers have also been suggested to define low-level behavior of a network node. These suggestions are limited to the configuration of routing tasks and require pre-installation of software on the network node for supporting new applications. However, such tasks cannot be done on the fly and require knowledge of the underlying programmable components. A more general reconfigurable network hardware platform (not only for routing) has been suggested [12] to reconfigure specific network tasks by using a field programmable port extender (FPX) [9]. The FPX is an FPGA designed to support different network applications, such as IP routing, per flow queuing, and flow control algorithms. The FPX is, however, limited to specific tasks (specifically, configuration of network flows) and should be pre-programmed, i.e., it is not dynamic solution.

Most of the research described above has focused on programming and reconfiguration of specific, pre-defined network services [6, 13, 12] or sets of rules and actions [11, 10] rather than on the complete scope of frame flow through a network system. In addition, most of these approaches require knowledge of the internal implementation of the FE and are dependent on it.

In this paper, we address the problems described above by introducing a heterogeneous, multifunctional, generic forwarding element (GFE) that serves as a platform for a flexible network system. This platform inherently supports new and on-demand network services, various network architectures, and multiple management protocols. This platform's functionality is fully programmable, on the fly, using an XML-based API. This API is used to retrieve information from the network system and to upload data for its meta-program. The uploaded XML data contains the relevant information that is used by the network systems to: define forwarding behavior and decisions; adapt classification, forwarding and processing rules; execute modifications in frames' contents and format; encapsulate and decapsulate frames; obey quality of service (QoS) criteria; and shape, learn, and filter traffic flows, frames and fields, and more. Per port characterization can be applied using these XML definitions, and a default characterization at bootstrap can be used.

The GFE is based on the ForCES framework, which is currently being standardized by an IETF working group [15]. The network system's functionalities are logically separated into: a management plane that configures and manages the network system; a control plane, including routing and signaling; and a forwarding plane, where per-packet activities, such as forwarding and queuing, occur. We use ForCES principles and mechanisms, albeit in the opposite way to their conventional means of application: in ForCES, a forwarding element model *describes the existing FE* [16], so that the control plane will know how to communicate with and control the FE, whereas in our GFE architecture, we use the FE model *to define how the FE will work and generate its functionality.*

ForCES was developed in parallel to network processors with the aim to utilize the rich forwarding capabilities that network processors offer. Although the GFE can be based on a variety of FE models and communication protocols, we find ForCES simple and adequate for our requirements. In addition, it allows designing of a generic definition of forwarding functionalities.

The proposed GFE is not limited to specific services, and, as mentioned above, users (e.g., vendors, operators, administrators) can use ForCES and XML API to define and program the GFE functionality. The XML data and meta-data are translated to internal structures and mechanisms, which are used by a meta-program to define the GFE behavior. No code writing or machine specific programming language (or programming at all) is required to program the GFE, and the GFE is configured in an abstract manner. The GFE thus can be used for building a generic forwarding plane that can: change the classification rules of packets, change forwarding behavior, modify frames' contents and format, support multicast and OAM functions, and change queuing policies, shaping mechanisms, learning and filtering algorithms, and encapsulation and decapsulation rules.

The GFE supports a pre-defined library of many functions that are used for an on-the-fly definition of forwarding functionalities (as described above, e.g., mapping, classifying, or traffic management). These functions constitute the basic building blocks for the required forwarding plane's functionalities; additional functions can be written, either by extending the library or by re-directing the frame that requires a specific forwarding functionality to a service agent that resides in the FE or the control element (CE).

The GFE architecture is described in the next section, followed by a section that defines how to program the GFE. The fourth section describes the GFE prototype and its performance, and the fifth section concludes the paper.

2. ARCHITECTURE

The proposed GFE is based on concepts of the ForCES framework, which is used to separate control and forwarding entities into different planes and to define uniform interfaces and protocols for exchanging information between the planes. Specifically, ForCES defines two types of elements:

- The FE is responsible for the actual, per-packet processing functionality, which includes classifying, metering, shaping, queuing, and tasks alike. The FE is typically implemented by specific network hardware such as FPGAs, ASICs, or network processors;

- The CE controls FE functionality in a master/slave mode, using routing and signaling protocols (e.g., RSVP, BGP) and is typically implemented by general purpose CPUs.

ForCES also defines an FE model [16] that contains Logical Function Blocks (LFBs), which are the basic building blocks of the FE and are connected in the FE by using a directed graph. Each LFB defines a set of packet processing operations; when a packet flows through the FE, it is served by one or more LFB instances.

The GFE takes the ForCES model one step forward. Originally, specific functionalities were assumed by the FE (regarding supported frames, protocols, and other network services) and were described by the FE's LFBs. The details of implementations of data-paths inside an FE (i.e., LFB topology, and the operational capabilities and attributes of the LFBs) are communicated to the CE at the association stage of the ForCES protocol. The CE then uses the FE-supported features (according to the LFBs) to configure and define the FE operation. The GFE, on the other hand, has no specific functionality upon bootstrap but to accept "instructions" from the CE for defining its functionalities. In other words, the GFE's functionality is programmed from scratch by the CE; thus the CE decides how the GFE should behave. A meta-program in the GFE is used to implement the forwarding functionalities and frame processing, according to XML-based LFBs that are received from the CE, using API and the ForCES protocol.

The flexibility level of the GFE regarding radically new services depends on the ability to define the new service using LFBs and on the extent to which the ForCES/XML description of LFBs is sufficient. As noted at the end of the Introduction, pre-defined LFBs can be used for an on-the-fly definition of the required service, or, if the existing LFBs are not sufficient, a new LFB or service agent must be programmed.

2.1 GFE modules

The GFE contains two groups of modules (see Figure 1): management modules and run-time modules. The management modules interact with the CE, receive the LFB information from the CE, analyze the information, and create internal data structures that define and control the way the run-time modules process each of the frames. The three main GFE modules are the *ForCES agent*, the *XML parser*, and the *Low Level Rule Enforcer*. The ForCES agent and the XML parser are management modules, and the Low Level Rule Enforcer is a run-time module.

The *ForCES agent* communicates with the CE and implements the ForCES protocol, which governs FE to CE communications. The *XML Parser* parses XML information that programs different LFB functionalities and translates this information into relevant LFB specific data structures that reside in the GFE. The *Low Level Rule Enforcer* executes the GFE meta-program and implements the various LFBs that were defined by the CE and communicated through the ForCES agent and the XML parser. As noted above, these LFBs are the building blocks of the GFE; some of these LFB classes are common to all frames (i.e., each frame passes through at least one instance of these LFB classes). The common LFB classes include the *LFB Classifier*, which defines how to identify a frame and that frame's classification rules, and the *LFB Forwarder*, which includes the required information for defining a forwarding decision. The LFB Forwarder is also used as a reference to additional LFBs, such as the LFB Mapper, and the LFB Auto-Learner.

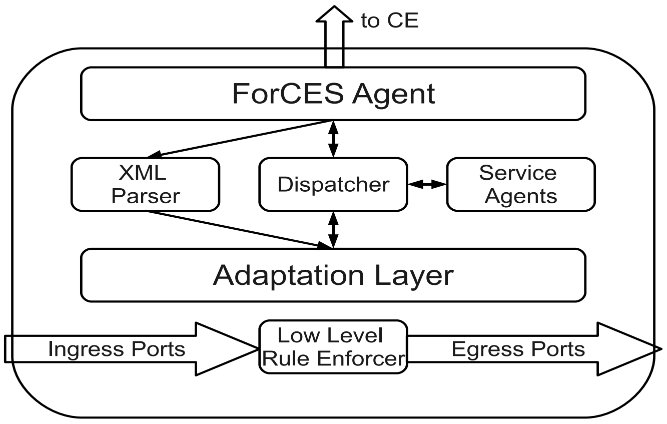

Figure 1: The GFE Architecture

The ability to map any LFB to the *Low Level Rule Enforcer* is not trivial and depends both on the complexity level of the LFB, and on hardware limitations (e.g., memory or search engines).

An *Adaptation Layer* module is used for abstracting the hardware implementation of the *Low-Level Rule Enforcer* to the management modules, as detailed in the next subsection.

Two other module types are included in the GFE to support additional services. Some LFBs can divert specific in-band traffic frames, if programmed to do so, to the *Dispatcher* module. The *Dispatcher* then sends these frames to various *Service Agents*, according to the frame type. These service agents are implemented either in the FE or the CE, depending on the service functionalities. One obvious service agent in the FE is the *ForCES agent* to which the dispatcher sends ForCES frames that arrive from the GFE ingress ports as in-band traffic. Another example of such a service agent in the FE is an OAM processor, which has to be implemented in the forwarding plane. The frame is re-directed to the CE for services that are implemented in the control plane. In addition, a *Frame Generator* is another module that can be used to generate frames in the forwarding element when required (e.g., ARP requests).

2.2 Implementation

The management modules can be implemented by general purpose processors, using high-level language, since the GFE programming procedure that is performed by these management modules can tolerate some latency. On the other hand, per-frame processing must be carried out at wire speed. The run-time module (i.e., the *Low Level Rule Enforcer*) must thus be implemented by embedded programmable network hardware (e.g., ASIC or network processor). The run-time modules have to be written in the most efficient way (i.e., in low-level machine language), when they use the data structures (which were created by the management modules) to execute the required procedure on each of the frames, according to the frame's header and payload information.

The *Low Level Rule Enforcer* is therefore hardware dependent but can be implemented by various hardware platforms;

```
<LFBs>
<LFB_Classifier>
  <synopsis> LFB used to classify frames. It include all frame definitions </synopsis>
  <frameDefs>
    <frame>
      <frameTypeId>0x01</frameTypeId>
      <name>802.1Q</name>
      <synopsis> Ethernet frame with tag </synopsis>
      <fields>
        <field type=”forwarding−key” length=”48" name=”Dest address”/>
        <field type=”regular” length=”48" name=”Src address”/>
        <field type=”identifier−key” length=”16" name=”Ether Type>0x8100<field/>
        <field type=”QoS−key” length=3 name=”priority”/>
        <field type=”regular” length=1 name=”format indicator”/>
        <field type=”forwarding−key” length =”12" name=”VID”/>
      </fields>
    </frame>
    <frame>
      <frameTypeId>0x02</frameTypeId>
      <name>foobar</name>
      <synopsis> Sample frame </synopsis>
      <fields>
        <field type=”identifier−and−forwarding−key” length=”16" name=”Sample field”/>
      </fields>
    </frame>
  </frameDefs>
</LFB_Classifier>
</LFBs>
```

Figure 2: LFB Classifier

it may differ from one GFE implementation to another. The most flexible hardware, which can still meet performance demands, is a network processor. To cope with various hardware platforms, the Adaptation Layer abstracts the hardware and provides a separation between the high-level functionalities and the low-level implementation of the GFE. In other words, the adaptation layer interfaces and handles the interactions between the specific hardware implementation and the management modules. When a specific hardware implementation is replaced by another embedded hardware, the adaptation layer should be modified.

3. GFE CONFIGURATION AND PROGRAMMING

Every frame flows though two parts of the run-time modules of the *Low Level Rule Enforcer*. The first part is the *Classification Phase*, which determines the type of frame and how to make a forwarding decision. A per-port classification can be defined, so that each frame can flow through a port-specific classification, based on its ingress port. Any frame that cannot be classified (not recognized or not supported) is dropped. The classification phase sends a message to the next part of the *Low Level Rule Enforcer*, the *Forwarding Phase*, which uses the message to construct keys in internal tables that define its forwarding behavior. The two logical independent parts enable the reclassification of frames for additional processing and support multiple instances of the same original frame to flow through the forwarding phase (enabling multicasting).

Each of the *Low Level Rule Enforcer* phases is constructed from LFBs (the basic building blocks of the GFE's run-time modules). The LFBs are defined by the CE and communicated to the GFE using an XML-based definition as part of the ForCES protocol payload. Therefore, to configure and program the GFE, we need to take a closer look at how the CE defines and sends LFBs.

3.1 LFB Classifier

This is the first LFB, which is mandatory in the classification phase of the *Low Level Rule Enforcer*. The CE defines first, with this LFB, the various frames that should be handled by the FE. The frames' fields are described by tags, which include a name and length (bit wise). Some fields can be defined as special tags that represent keys in the frame. The keys are:

- Identifier key (mandatory for each frame): Each frame is assumed to contain a unique identifier that distinguishes it from other frames (e.g., Ethernet type, protocol type);
- Forwarding key (optional): A field in the frame that is used for forwarding decisions (e.g., destination address, VLAN tag);
- QoS key (optional): A field in the frame which is used to represent the QoS level;
- Metering key (optional): A field in the frame that is used for metering decisions.

There is no limit on the header length or the number, lengths and location (offset) in the frame of the keys. After defining the frames, the CE can then use the ForCES protocol to associate the frames with several GFE attributes (such as ingress ports that support specific frame types, token bucket rate, or shaping and metering policies). An example of a frame structure definition is illustrated in Figure 2.

3.2 LFB Forwarder

This is the first and mandatory LFB in the forwarding phase of the *Low Level Rule Enforcer*. The LFB Forwarder defines a unique forwarding behavior for a frame instance. The forwarder can be considered as a table containing entries of a key and results. The *key* is constructed from several parameters:

- The frame type (identifier);
- The ingress or virtual port, which enables different behaviors that depend on the ingress port, even for the same frame types (virtual ports are used for loopback purposes, as described below);
- The forwarding key values, which were defined in the LFB Classifier for forwarding decisions; not all forwarding keys must be used, and if a forwarding key is not used, it will be ignored in the GFE (enabling wildcard forwarding keys, such as any VLAN Tag).

Since the forwarding key usage is optional, the forwarding behavior can be associated solely with a frame type (identifier) or a port and a frame type combination. A default forwarding behavior, which is the result of the forwarding table, can also be defined if no match occurs. The *result* contains the following information:

- Reference to additional LFBs – referred by a unique identifier; additional LFBs are described by an XML-based definition;
- Outgoing egress ports, virtual ports, or the dispatcher (for frames designated to the GFE itself); if additional LFBs are called, these ports are used after the called LFBs are executed;

- QoS (for a specific frame type);
- Priority list of operations that replace fields' contents in the frame;
- Multicast or unicast treatment.

Virtual ports are used for loopback, i.e., to send frames for reclassification and forwarding after changes are made in the frame format or contents, thereby enabling additional operations for the modified frames. The virtual port can then be used as a key to the forwarder. Multicasting is supported simply by defining multiple results for one key. Each result defines a different forwarding behavior. In addition to sending a frame multiple times to several ports, this mechanism also enables the forwarding of the copied frame in several processing versions, i.e., using different format or field contents for some ports. The operations that replace fields in the frame can be immediate values (i.e., constants), either copied from other fields or computed by arithmetic/logic operations (e.g., incrementing the TTL value). Figure 3 describes the LFB Classifier XML.

3.3 LFB Mapper

This LFB is an example of an optional LFB in the forwarding phase. Changes in the frame format are performed by the *LFB mapper*, using a simple API for defining the required changes. The LFB mapper is required when an ingress frame format has to be converted to a different frame format, encapsulated or decapsulated. Each mapping has a unique identifier, and the LFB mapper maps two frame formats, source and target, and defines the association between fields in the source frame to fields in the target frame.

4. GFE IMPLEMENTATION AND RESULTS

Flexibility is, almost always, achieved at the expense of performance. In a system that deals with ultra-high forwarding speed and extreme packet processing performance, it is important to evaluate the generic and flexible offering in performance measure. To this end, a prototype was built, programmed, and tested in various scenarios. This section provides initial results for the performance-flexibility tradeoff.

4.1 Prototype

The GFE was implemented, using a network processor, on development platforms that contain EZchip's network processors (NP-1 and NP-2). These platforms enable ten 1 Gbps Ethernet ports and 10-Gbps full duplex packet processing. EZchip's NP is a highly integrated network processor that is based on pipeline architecture for packet processing and therefore fits the logical phases of the run-time modules, as presented in section 3. The development platform also contains a general-purpose CPU that allows high-level programming (e.g., C++) and functionalities that are integrated with the network processor via the EZdriver. We built a prototype that contains the basic module of the GFE, i.e., the *Low Level Rule Enforcer* meta-program (programmed in micro-code), the adaptation layer (using the EZdriver), the ForCES agent and other agents (using C++). We also implemented a simple CE that programs the GFE using the out-band ForCES protocol (above TCP/IP) and the basic XML LFBs (classifier, forwarder, and mapper).

```
<LFBs>
<LFB_Forwarder>
  <synopsis> LFB holding the forwarding database </synopsis>
    <entries>
      <fdbntry active="yes" id="0x01" frameTypeID="0x01" tableID="0x07" static="yes"/>
        <incoming_fr>
          <key>
            <ports>
              <portId>0x03</portId>
              <portId>0x09</portId>
            </ports>
            <frameFields>
              <frameField name="Dest address">0x00AABBCCDDEEFF</frameField>
              <frameField name="VID">0xABC</frameField>
            </frameFields>
          </key>
        </incoming_frame>
        <outgoing_frame>
          <isMulticast value="no"/>
          <results>
            <resultId>0x01"</resultId>
            <destPortId>0x02</destPortId>
            <qos>0x05</qos>
            <frameFields>
              <frameField name="Src address">0x001122334455</frameField>
            </frameFields>
            <mappidId>0</mappidId>
          </results>
        </outgoing_frame>
      </fdb_entry>
    </entries>
</LFB_Forwarder>
</LFBs>
```

Figure 3: LFB Forwarder

4.2 Results

Several bridging algorithms and functionalities were programmed on the GFE, using two types of network processors. First, the GFE was programmed to solely support bridging, and then it was programmed to support multiple frame formats and various network tasks, including bridging. In addition, for purposes of comparison, a normal Ethernet bridge was implemented using the same network processors. We measured the performance of both bridge implementations (generic and normal) using the same equipment and environment. Performance was evaluated in terms of the average frame latency (i.e., the average frame processing time) caused by implementation of a bridge. To calculate performance, the average traverse time of a frame was measured in a specific network topology, when frames of various sizes (from 100 to 1500 bytes) were transmitted into the network, 10,000 frames for each size. Then, by rearranging the network topology and omitting one bridge, we were able to calculate (by subtraction) the average processing time of a bridge implementation that is required to bridge a frame in the various ways described above.

Figures 4 and 5 compare the frame processing time between a normal Ethernet bridge and the GFE bridging, using NP-1c and NP-2, respectively. The GFE was tested under two scenarios. In the first scenario, it was programmed to support more than 40 frame formats and network services. In the second, it was programmed to support only the bridging functionality. The comparison shows that the GFE requires several additional micro-seconds of overhead per frame, regardless of the frame length. This fixed overhead is dependent on the complexity of the required functionality that is executed by the GFE. For plain bridging, the GFE overhead is less than 50% in most cases, and when the GFE performs more complicated tasks, this overhead increases (although complicated tasks are compared to simple bridging).

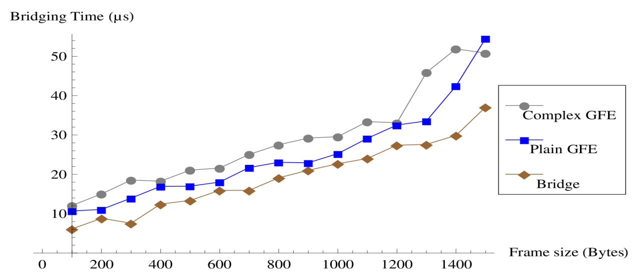

Figure 4: Bridging time - NP-1c

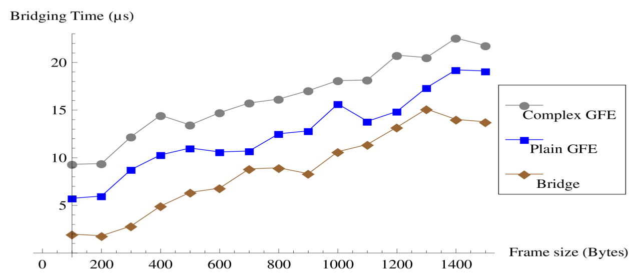

Figure 5: Bridging time - NP2

5. CONCLUSIONS

A GFE can provide a solution to the increasing requirements for dynamic, programmable high-speed network systems, which become essential for concurrent networks and services. The flexibility is provided by the suggested GFE, based on the ForCES protocol and the FE model, albeit from the opposite perspective, i.e., generating the functionality rather than using it. The GFE was architected, programmed, and prototyped, and performance comparisons were performed for evaluating the trade-offs of this flexibility. The results show that the GFE is quite efficient, and that the overhead in the case of a simple switch was less than 50% above the performance of a dedicated, inflexible, special-purpose switch.

The GFE functionality was examined by an independent group of researchers, who successfully used a CE platform to quickly define FEs that support OAM, protections, multicast and other network functionalities using ForCES and the XML API.

OpenFlow is a subset of the GFE in terms of forwarding functionalities (i.e., classifier and forwarder LFBs in the GFE can do exactly what the OpenFlow switch [10] does). Therefore, the GFE can also be used as an enabled OpenFlow switch [10], by defining classifier and forwarder LFBs instances and implementing or translating OpenFlow protocol.

Acknowledgment

The authors would like to thank Nir Amira, Kfir Damari, Jenia Feldman and Eliav Menachi for their help with the designing and programming of the GFE prototype.

6. REFERENCES

[1] *Cisco Visual Networking Index - Forecast and Methodology*, 2007-2012. http://www.cisco.com/en/US/netsol/ns827/networking_solutions_sub_solution.html

[2] Miniwatts Marketing Group,*"World Internet Usage Statistics News and Population"* http://www.internetworldstats.com/stats.html.

[3] K. Calvert *"Reflections on Network Architecture: an Active Networking Perspective"*, *ACM SIGCOMM Computer Communication Review*, 36(2), pp 27-30, 2006.

[4] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, *"Ethane: Taking Control of the Enterprise"*, ACM SIGCOMM, 2007.

[5] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, *"A Clean Slate 4D Approach to Network Control and Management"*, *ACM SIGCOMM Computer Communication Review*, Vol. 35, Issue 5, pp. 41-54, 2005.

[6] I. Houidi, W. Louati, D. Zeghlache, *"An extensible software router data-path for dynamic low-level service deployment"*. Proceedings of the 7th IEEE Workshop on High Performance Switching and Routing , Poznan, Poland, pp. 161-166, 2006.

[7] E. Keller, E. Green, *"Virtualizing the data plane through source code merging"*, ACM SIGCOMM (PRESTO), 2008.

[8] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. F. Kaashoek, *"The Click modular router"*, *ACM Transactions on Computer Systems*, 18(3), pp. 263-297, 2000.

[9] J. W. Lockwood, N. Naufel, J. S. Turner, D, E. Taylor, *"Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)"*, ACM International Symposium on Field Programmable Gate Arrays, pp. 87-93, 2001.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, *"OpenFlow: Enabling Innovation in Campus Networks"*, *ACM SIGCOMM Computer Communication Review*, Vol. 38, Issue 2, pp. 69-74, 2008.

[11] C. L. Schuba, J. G., M. F. Speer, M. Hefeeda, *"Scaling Network Services Using Programmable Network Devices"*, *Computer*, 38(4), pp. 52-60, 2005.

[12] T.S Sproull, J.W. Lockwood, D.E. Taylor, *"Control and Configuration Software for a Reconfigurable Networking Hardware Platform"*, Proceedings of Field-Programmable Custom Computing Machines, pp. 45- 54, 2002.

[13] E. Suet, H. Tse, Y. Kogan, *"Architectural designs for a scalable reconfigurable IP router"*, *Journal of Systems Architecture*, Vol. 54, Issue 1-2, pp. 197-223, 2008.

[14] D. L. Tennenhouse, D. L. Wetherall, *"Towards an Active Network Architecture"*, Computer Communications Review, 26(2), pp. 5-18, 1996.

[15] L.Yang et al., *"Forwarding and Control Element Separation (ForCES) Framework"*, RFC 3746, 2004.

[16] L.Yang et al., *"ForCES Forwarding Element Model"*, Internet Draft, 2005.