A Proposed Architecture for the GENI Backbone Platform

Jonathan S. Turner
Applied Research Laboratory
Washington University in St. Louis
jon.turner@wustl.edu

Abstract. The GENI Project (Global Environment for Network Innovation) is a major NSF-sponsored initiative that seeks to create a national research facility to enable experimental deployment of innovative new network architectures on a sufficient scale to enable realistic evaluation. One key component of the GENI system will be the *GENI Backbone Platform* (GBP) that provides the resources needed to allow multiple experimental networks to co-exist within the shared GENI infrastructure. This paper reviews the objectives for the GBP, the key issues that affect its design and develops a reference architecture that provides a concrete example for how the objectives can be met, using commercially available subsystems.

1. Introduction

The GENI initiative [GE06] seeks to create an experimental facility to enable networking researchers to develop and deploy novel network architectures that address important shortcomings of the current Internet. The proposed facility will use virtualization to enable different groups to share the available resources, including physical link bandwidth and processing resources at the network nodes.

A key objective of GENI is to enable new network architectures to be deployed and evaluated *at scale*. This means, among other things, that at least some of the experimental networks must have the potential to provide service to large numbers of end users, where large is taken to mean at least 100,000. This is considered important for two reasons. First, because many of the network characteristics one would like to be able to evaluate, only become apparent as networks get large. Second, if new network architectures are to have an impact on commercial practice, network providers and equipment vendors must have a good reason to take them seriously. The most compelling evidence for the value of a new network architecture will be its use by large numbers of people.

The *GENI Backbone Platform* (GBP) is one of the key components of the planned GENI facility. The GBP is envisioned as a flexible, high performance platform that provides the resources needed to allow multiple experimental networks to move large volumes of traffic across the country. It is expected that GBPs will be located at a few tens of sites around the country, with each site terminating a small number of fibers (typically 2-4), with possibly multiple wavelengths on each fiber. It has been suggested that the GENI backbone may use fiber facilities from the National Lambda Rail (NLR) network [NLR]. The NLR network topology has 25 sites and two major east-west routes. A GENI network with a single 10 Gb/s wavelength on each NLR fiber segment would be able to support up to 10,000 concurrent cross-country flows with an average bandwidth of 1 Mb/s, while maintaining an average link occupancy of 50%. Substantial increases in either the number of concurrent cross-country flows, or the average bandwidth per flow would require multiple wavelengths on the major cross-country routes. These observations provide a rough indication of the range of IO capacities that the GBP must provide.

In addition to IO, the GBP must provide flexible processing resources that can be allocated to different experimental networks that are implemented in GENI. To provide maximum experimental flexibility, the GBP should provide sufficient processing resources to enable networks in which the ratio of processing to IO is substantially higher than in conventional routers. Since the GBP is intended as a general experimental platform, it is also important that it provide access to a variety of different types of resources, so

This research was supported by the National Science Foundation (grants ANI-0325298 and CNS-0520778). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSF.

ANCS'06, December 3–5, 2006, San Jose, California, USA. Copyright 2006 ACM 1-59593-580-0/06/0012...$5.00.

that researchers can select the resources that best match the needs of their specific network architecture.

This paper describes a reference design for the GBP. In Section 2, we identify the overall objectives for the design. In Section 3 we describe the key abstractions that are implemented by the proposed reference design. Section 4 identifies and compares two major system architecture options for the GBP and argues that the *processing pool architecture* is the best choice for an experimental facility like GENI. Section 5 provides a detailed description of the proposed reference design, including descriptions of all the major subsystems. Finally, in Section 6, we outline a range of different system configurations.

2. Objectives

The purpose of the GBP is to enable multiple, diverse *metanetworks* to co-exist within a common shared infrastructure. To do this, it must enable sharing of backbone links and node processing resources. We expect researchers to use GENI for a wide range of different experiments, with highly diverse requirements. To enable the GBP to serve the widest possible range of objectives, it should be highly flexible and should provide sufficient resources to avoid constraining the research agendas of its users.

A GBP will host multiple *metarouters* belonging to distinct *metanetworks* (we use the term metarouter and metanetwork rather than virtual router and virtual network, because the latter terms have been heavily overloaded and are more subject to misunderstanding). The GBP will provide resources that can be used by different metarouters and the underlying mechanisms to allow each one to operate independently of the others. The term metarouter is used here in a very generic sense to mean any network component with multiple interfaces that forwards information through a network, while possibly processing it as it passes through. It can include components whose functionality is similar to that of an IP router, or components that switch TDM circuits, or components that operate like firewalls or media gateways. A given metanetwork may include metarouters of various types. It is left to the designers of individual metanetworks to define the precise functionality of their metarouters and to distinguish among different types of metarouters as they find appropriate.

The design of the GBP is distinctly different from that of conventional routers and switches. The following paragraphs summarize some important high level objectives for the GBP.

- *Scalable performance*. The experimental networks developed for GENI will have a wide range of characteristics, leading to widely differing requirements for GBP resources. Metanets seeking to support high volume data transfers for e-science applications may require multiple 10 Gb/s links, while metanets designed to transfer text messages among pagers may have little use for links above 100 Mb/s. Different metarouters will also have very different processing needs. While IPv4 forwarding requires fewer than 20 instructions executed per data byte, some experimental networks may require hundreds. The GBP should enable its resources to be allocated flexibly among many metarouters, and should support configurations suitable for a variety of performance ranges.
- *Stability and reliability*. If GENI is to provide a useful platform for experimentation and deployment of experimental network services, it must be sufficiently reliable and stable to allow researchers to work without interference from others. Because the experimental networks that run within GENI will be the subjects of on-going experimentation and modification, their stability will be highly variable. Nonetheless, the platform itself must be stable, so that researchers can focus on issues arising within their own experiments and not be concerned with the stability of the underlying substrate. The isolation mechanisms for metarouters (discussed below) are one element of the overall strategy for achieving reliable operation. However, it is also important that the hardware components used to implement the GBP have high intrinsic reliability and that the GBP as a whole be easy to manage and maintain, so that outages due to operational errors are kept to a minimum.
- *Ease of use*. Academic researchers have limited resources they can devote to development of experimental systems, making it important that it be as easy as possible for them to implement their metanetworks. There are intrinsic challenges here, since the technologies that yield the highest possible performance are often not the easiest to use. The GBP should enable use of high performance technologies, while minimizing the barriers to their use. In addition, the GBP should facilitate sharing of common software and configurable logic modules among different research groups.
- *Technology diversity and adaptability*. The GBP should enable the construction of metanetworks using a variety of different underlying technologies, including general purpose processors, network processor subsystems and configurable logic subsystems. This will allow different researchers to pursue different strategies for meeting their research objectives and will provide the flexibility for the system to incorporate new implementation technologies, as they become available.
- *Flexible allocation of link bandwidth*. Link bandwidth is a key resource. The GBP should support flexible allocation of bandwidth to different metanetworks including both reserved and shared bandwidth models. It should provide mechanisms for circuit-based management of

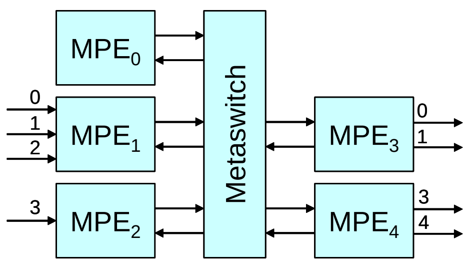

Figure 1. Example metarouter

links, allowing researchers to experiment with novel frame formats and time-domain switching techniques.

- *Isolation of metarouters.* The GBP must allow different metarouters to co-exist without interference. Ideally, each metarouter should have the illusion that it is operating within a dedicated environment. This means that resources like memory and disk space must be free from modification by other metarouters and that metarouters have the ability to reserve dedicated processing capacity and link bandwidth.
- *Minimize constraints on metarouters*. The GBP should place as few constraints as possible on the metarouters it hosts. In particular, it should not place any constraints on data formats or limit the ways in which metanetworks provide various capabilities, or constrain the way in which they use their assigned processing resources.

It should be understood that the above list is not comprehensive. It has been intentionally limited to fairly high level objectives. The remainder of this paper provides a more detailed view of the GBP.

3. GBP Abstractions

A GBP will host multiple *metarouters* that terminate *metalinks* connecting to other metarouters and to end systems. The metarouter and metalink are key abstractions that are implemented by the GBP. These abstractions are described in the following subsections.

3.1. Metalink Abstraction

The metalink is an abstraction of a point-to-point physical link. It may have a guaranteed transmission bandwidth and a maximum bandwidth, but is not required to have either. While the backbone links connecting GBPs will typically have provisioned bandwidths, we expect at least some links connecting GBPs to user sites will be implemented using unprovisioned tunnels. In addition, not all metanetworks require provisioned bandwidth. The configuration of a metanetwork will include a specification of its metalinks. For those that are specified with provisioned bandwidth, the metanet configuration software will choose an underlying implementation that can support this. A metalink may be unidirectional or bi-directional. A bi-directional metalink may have asymmetric bandwidth provisioning.

The metalink abstraction can be extended to support links with more than two endpoints. The primary motivation for supporting a multipoint metalink is to make use of the features of multi-access subnets in the LAN environment. Since this paper is primarily concerned with the backbone environment, we omit discussion of the multipoint case here.

3.2. Metarouter Abstraction

A metarouter is an abstraction of a conventional router, switch or other network component, which typically consists of three major components, *line cards*, a *switching fabric* and a *control processor*. The line cards terminate the physical links and implement specific processing functions that define a particular network. On input, this may include performing a table lookup of some sort, to determine where an arriving packet should be sent next and what special processing (if any) it should receive. Alternatively, it might involve identification of TDM frame boundaries, and time-division switching of timeslots within frames. On output, it might include scheduling a packet for transmission on the outgoing link, or the formatting of a TDM frame. The switching fabric is responsible for transferring data from the line cards where they arrive, to the line cards for their outgoing links. Switching fabrics are typically designed to be *nonblocking*, meaning that they should handle arbitrary traffic patterns, without interference within the fabric. For circuit networks this means that any set of circuits that can be supported by the external links should be supported by the fabric. For packet networks, it also means that excessive traffic to a particular output line card should not interfere with traffic going to other line cards. The control processor in a conventional router implements various control and administrative functions (such as executing routing protocols and updating tables in the line cards). These functions are generally implemented in software running on a general-purpose microprocessor.

A metarouter has a similar structure. It consists of two types of components *MetaProcessing Engines* (MPE) and a *MetaSwitch* (MS). MPEs can be used to implement data path functions within a metarouter or higher level control functions and may be implemented using various types of underlying processing resources. Metalinks terminate at *meta-interfaces* (MI) on MPEs and MPEs are connected to each other through the MS. MIs are subject to maximum bandwidth limits, as are the interfaces between MPEs and the MS. Figure 1 shows an example of a metarouter.

Metarouters with limited performance needs may use a single MPE. In this case, there is no need for a metaswitch. Another common case is a metarouter with two MPEs, one that implements the normal data forwarding path, and another that implements control functions and handles exception cases. In this case, the MIs will typically all be associated with the data path MPE (implemented on a net-

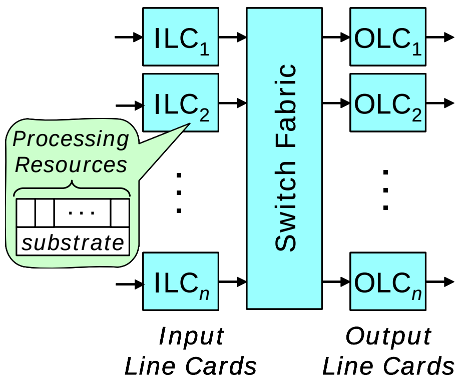

Figure 2. Virtualized Line Card Architecture

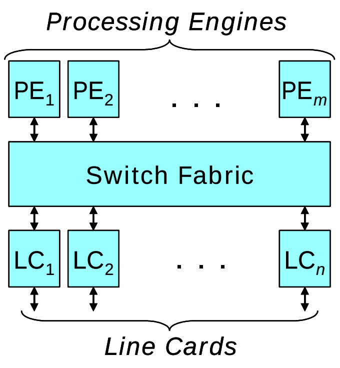

Figure 3. Processing pool architecture

work processor, perhaps) which will have a logical connection directly to the control/exception MPE (implemented on a general purpose processor). In this case, the MS reduces to the single logical connection between the two MPEs.

4. System Options and Issues

This section discusses two high level system architecture options for the GBP and issues arising from these options.

4.1. Virtualized line card architecture

Consideration of a conventional router or switch leads naturally to an architecture in which line cards are replaced by *virtualized line cards* that include a substrate portion and generic processing resources that can be assigned to different meta line cards (Figure 2). The substrate supports configuration of the generic processing resources so that different meta line cards can co-exist without interference. On receiving data from the physical link, the substrate first determines which meta line card it should be sent to and delivers it. Meta line cards pass data back to the substrate, in order to forward it through the shared switch fabric, on input, or to the outgoing link, on output.

One issue with this architecture concerns how to provide generic processing resources at a line card, in a way that allows the resources to be shared by different meta line cards. Conventional line cards are often implemented using Network Processors (NP), programmable devices that include high performance IO and multiple processor cores to enable high throughput processing. It seems natural to take such a device and divide its internal processing resources among multiple meta line cards. For example, an NP with 16 processor cores could be used by up to 16 different meta line cards, by simply assigning processor cores. Unfortunately, current NPs are not designed to be shared. All processing cores have unprotected access to the same physical memory, making it difficult to ensure that different meta line cards don't interfere with one another. Also, each processor core has a fairly small program store. This is not a serious constraint in conventional applications, since processing can be pipelined across the different cores, allowing each to store only the program it needs for its part of the processing. However, a core implementing an entire meta line card must store the programs to implement all the processing steps for that meta line card. The underlying issue raised by this discussion is that efficient implementation of an architecture based on virtualized line cards, requires components that support *fine-grained virtualization* and conventional NPs do not.

The virtualized line card approach is also problematic in other respects. Because it associates processing resources with physical links, it lacks the flexibility to support metarouters with a wide range of processing needs. Some metarouters may require more processing per unit IO bandwidth than NPs provide, and this is difficult with a virtualized line card approach. The virtualized line card approach also does not easily accommodate alternate implementation approaches for metarouters (such as configurable logic).

4.2. Processing pool architecture

The processing pool architecture separates the processing resources used by metarouters from the physical link interfaces. This allows a more flexible allocation of processing resources and reduces the need for fine-grained virtualization. This architecture, illustrated in Figure 3, provides a pool of *Processing Engines* (PE), that are accessed through the switch fabric. The line cards that terminate the physical links forward packets to PEs through the switch fabric, but do no processing that is specific to a particular metanet. There may be different types of PEs, including some implemented using network processors, others implemented using conventional microprocessors and still others implemented using FPGAs. The NP and FPGA based PEs are most appropriate for high throughput packet processing, the conventional processor for control functions that require more complex software or for metanets with a high ratio of processing to IO. A metarouter may be implemented using a single PE or multiple PEs. In the case of a single PE, data will pass through the physical switch fabric twice, once on input, once on output. In a metarouter that uses multiple

PEs to obtain higher performance, packets may have to pass through the switch fabric a third time.

The primary drawback of the processing pool architecture is that it requires multiple passes through the switch fabric, increasing delay and increasing the switch capacity needed to support a given total IO bandwidth. The increase in delay is not a serious concern in wide area network contexts, since switching delays are typically 10 μs or less. The increase in capacity does add to system cost, but since a well-designed switch fabric represents a relatively small part of the cost of a conventional router (typically 10-20%), we can double, or even triple the \capacity without a proportionally large increase in the overall system cost. In the GENI context, the switch fabric bandwidth implications of the processing pool architecture are significantly reduced, since we expect the metarouters implemented within a GBP to have a relative high ratio of processing capacity to IO bandwidth, compared to conventional routers.

The great advantage of the processing pool architecture is that it greatly reduces the need for fine-grained virtualization within NP and FPGA-based subsystems, for which such virtualization is difficult. Because the processing pool architecture brings together the traffic for each individual metarouter, there is much less need for PEs to be shared among multiple metarouters. The one exception to this is metarouters with such limited processing needs that they cannot justify the use of even one complete PE. Such metarouters can still be accommodated by implementing them on a general purpose processor, running a conventional operating system that supports a virtual machine environment. We discuss below one approach that allows such metarouters to share an NP for *fast path forwarding*, while relying on a virtual machine running within a general purpose processor to handle exception cases.

Another advantage of the processing pool architecture is that it simplifies sharing of the switch fabric. The switch traffic must maintain traffic isolation among the different metarouters. One way to ensure this is to constrain the traffic flows entering the switch fabric so as to eliminate the possibility of internal congestion. This is difficult to do in all cases. In particular, metarouters consisting of multiple PEs should be allowed to use their “share” of the switch fabric capacity in a flexible fashion, without having to constrain the pair-wise traffic flows among the PEs. However allowing this flexibility makes it possible for several PEs in a given metarouter to forward traffic to another PE at a rate that exceeds the bandwidth of the interface between the switch fabric and the destination PE.

There is a straightforward solution to this problem in the processing pool architecture. To simplify the discussion, we separate the handling of traffic between line cards and PEs from the traffic among PEs in a common metarouter. In the first case, we can treat the traffic as a set of point-to-point streams that are rate-limited when they enter the fabric. Rate-limiting these flows follows naturally from the fact that they are logical extensions of traffic flows on the external links. Because the external link flows must be rate limited to provide traffic isolation on the external links, the internal flows within the switch fabric can be configured to eliminate the possibility of congestion.

For PE-to-PE traffic, we cannot simply limit the traffic entering the switch, since it’s important to let PEs communicate freely with other PEs in the same metarouter, without constraint. However, because entire PEs are allocated to metarouters in the processing pool architecture, it’s possible to obtain good traffic isolation in a straightforward way, for this case as well. In general, we need two properties from the switch fabric. First, it must support constrained routing, so that traffic from one to metarouter cannot be sent to PEs belonging to another metarouter. Second, we need to ensure that congestion within one metarouter does not affect traffic within another metarouter. The emergence of Ethernet as a backplane switching technology provides the first property. Such switches support VLAN-based routing that can be used to separate the traffic from different metarouters. The second property is satisfied by any switching fabric that is nonblocking at the port level. While some switch fabrics fail to a fully achieve the objective of nonblocking performance, this is the standard figure of merit for switching fabrics and most come reasonably close to achieving it.

5. Reference Design

This section describes a reference design for a GBP, that attempts to meet the objectives outlined in Section 2. Wherever possible, we have identified specific components and subsystems that can be used to implement various parts of the system. This is not meant to suggest that these are the only possible choices, but to make it clear that an effective solution can be assembled largely from components that have been or are being developed for commercial use.

5.1. System Overview

The reference design uses *ATCA* components to implement the processing pool architecture discussed in Section 4. ATCA stands for *Advanced Telecommunications Computing Architecture*, a rapidly developing set of standards designed to facilitate the development of carrier-class communications and computing systems [PCMG]. ATCA defines standard physical components and some standard patterns for how to use them to construct high performance systems. It has attracted broad industry support and is has led to the development of a range of inter-operable subsystems that allow development of cost effective and flexible new communication systems.

ATCA has important implications for the networking research community and the GENI project in particular. Net-

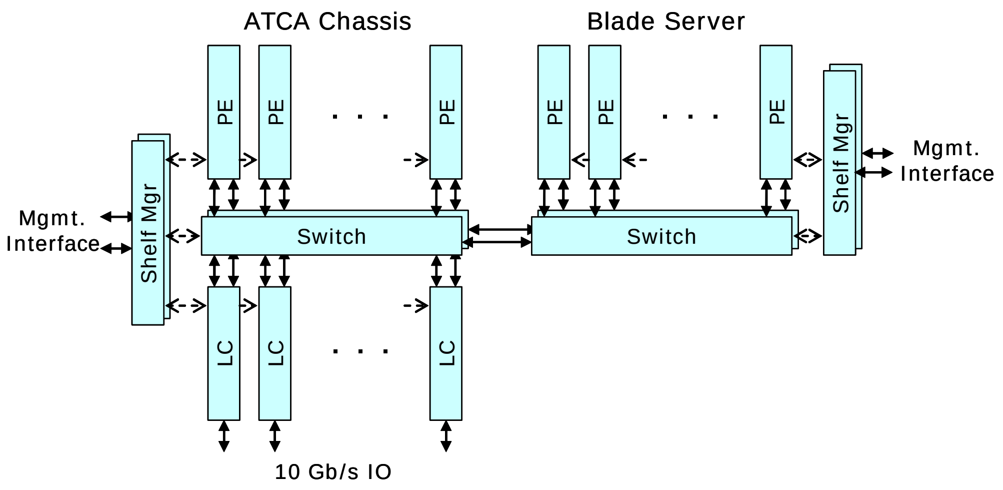

Figure 4. Baseline Configuration

working researchers interested in creating new network architectures and services have long had to content themselves with implementing experimental networks using commodity PCs. Commercial routers have been difficult to use in research contexts, because vendors have been unwilling to allow researchers to have access to the technical details needed to perform experiments and make changes. ATCA is creating an intermediate market for router subsystems that can be assembled into powerful, carrier-class systems. Subsystem vendors design their products to be highly flexible to enable their use by multiple system vendors. This gives networking researchers the tools to create high performance research systems that are built on a hardware platform that is directly comparable to the best commercial systems.

Figure 4 shows a baseline configuration for the GBP reference design. The baseline configuration includes an ATCA chassis that houses Line Cards and PEs implemented using network processors and FPGAs, and a conventional blade server that houses general purpose processing blades.

The ATCA chassis contains several primary components. The redundant *Shelf Manager* (SM) monitors the operation of the system and controls power and cooling. It provides an Ethernet interface through which the chassis can be monitored remotely and through which individual blades can be controlled (including hardware reset). The *Line Cards* (LC) terminate the external IO links and implement the substrate functions needed to multiplex/demultiplex metalinks to/from shared physical links. The *Processing Engines* (PE) provide generic processing resources for use by metarouters. The architecture supports multiple types of PEs, including PEs based on general-purpose processors, network processors and configurable logic chips. The *Switch Blades* provide high bandwidth IO linking the LCs and PEs and each has up to seven up-links, for connecting to other chasses. Each switch port provides a 10 Gb/s Ethernet interface with full VLAN support. The chassis has a total of 14 slots, two of which are reserved for the switch blades, leaving 12 for LCs and PEs (the SMs use separate, special-purpose slots). A typical GBP might use three of these for LCs and nine for PE.

5.2. General Purpose Processing Blades

To reduce overall system costs, we propose to use a commercial blade server to host the general purpose PEs, rather than using ATCA blades for this. Because the ATCA standards are still relatively new, the cost of ATCA components is not yet as competitive as those for commercial blade servers. Also, the IO capacity of the ATCA chassis far exceeds what conventional processor blades can use effectively. For this reason, it makes sense to reserve ATCA slots for PEs that can make greater use of its IO resources. The IBM Blade Center system is typical of the class of products that can be used to provide general purpose processing in the GBP. One chassis includes 14 processor blades, each with two 3.6 Ghz Xeon processors, two on-board 80 GB disks and up to 8 GB of memory. These are interconnected through redundant switch cards that plug into the rear side of the chassis and support redundant 1 Gb/s Ethernet connections to each slot. Each switch card has six 1 Gb/s up links that can be used to connect to the ATCA chassis. It is likely that switch cards with 10 Gb/s up links will be available soon.

5.3. NP Blades

Network Processors (NP) are high performance components with tens of processor cores and high performance IO. NP blades can be used in multiple contexts within the GBP. Specifically, they can be used both to implement line cards and PEs. The NP blades can be implemented using the Radisys ATCA 7010. These blades each contain two Intel IXP 2800 network processors [RA05]. Each NP has

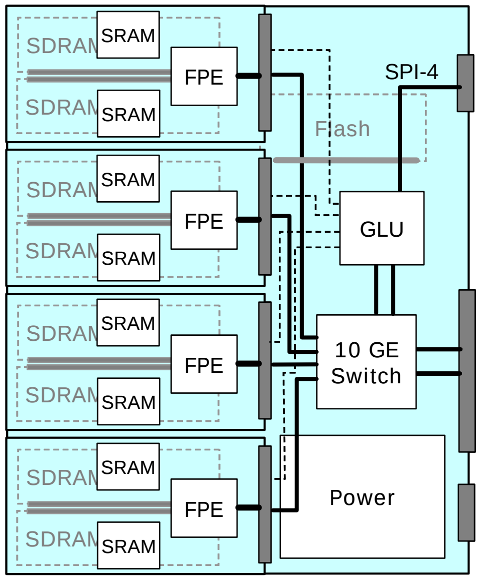

Figure 5. Configurable Logic Blade

sixteen internal processor cores for high throughput data processing, plus an Xscale processor (running Linux) for control. Each NP has three banks of RDRAM providing 750 MB of storage and four banks of QDR SRAM. The two NPs also share access to a dual-port TCAM that can be used for packet classification and other applications requiring associative lookup. The Line Card provides external IO through a Rear Transition Module (RTM). The board has several network connections. Two 10 Gb/s Ethernet connections are provided to the backplane for high speed data transfers. These connections go to the redundant switch blades. In addition, there are two 1 Gb Ethernet connections from the Xscale to the backplane and two more that come out the front panel.

5.4. Configurable Logic Blades

Figure 5 shows a block diagram of a configurable logic blade that can be used to implement hardware-based PEs. This blade includes a carrier card with four mezzanine card slots, each of which hosts a large FPGA called the FPE (e.g. Xilinx Virtex 5 LX330 or Altera Stratix II EP2S60) plus two banks of SDRAM and two or more QDR SRAM chips. The carrier card includes an on-board 10 GE switch that has two ports connecting to the backplane (one to each switch blade) and one port for each of the mezzanine cards. The carrier card also includes a GLU chip that has two key functions. First, it provides the logic to program the FPEs on the mezzanine cards remotely. New bit files are sent to it through the on-board 10 GE switch, where they are stored in a local flash memory. From there, they can be transferred to the FPEs, which are then reset. The GLU chip also provides a SPI4 interface to the *Rear Transition Module* (RTM) connector. This allows use of RTMs that provide external IO connections. A blade configured with an RTM can be used to implement Line Card functions.

5.5. Switch Blades

Figure 6 is a block diagram of a switch blade that has recently become available from Radisys (ATCA 2210). This blade includes a 20 port 10 Gigabit Ethernet switch that provides one port to each of the 12 slots designated for PEs and LCs. It also provides up to seven ports that can be used to connect to other chasses. In the baseline configuration, one of these is used to connect the general purpose blade server, leaving six available for connecting to other chasses in multi-chassis configurations. The 10 GE switch includes VLAN support, making it possible to constrain the routing of traffic from different metarouters. This can be used to provide the traffic isolation needed to keep metarouters from interfering with one another.

The board also includes a 24 port 1 GE switch intended for carrying control traffic and a Control Processor that configures the two switch components through an on-board PCI interface. The Control Processor has a front panel connection through which it can receive control messages and report status. Additional details can be found at www.radisys.com.

5.6. Line Cards

As noted earlier, the Line Cards can be implemented either using an NP blade or a configurable logic blade. However it is implemented, the LC must provide the substrate functionality needed to allow multiple metalinks to share the external physical links. On the ingress side, packets are demultiplexed and forwarded through the switch to the appropriate PEs. The LC can be configured to terminate IP and/or MPLS tunnels to facilitate reception of packets from remote sites that have no dedicated connections to the GBP. It must include a header mapping function to map arriving packets to a metarouter number, a meta-interface number and a physical destination within the GBP. Packets are labeled with their metarouter number and meta-interface number by the LC and forwarded through the switch to the specified destination. Packets going to the switch are sent through queues with a configured maximum rate, in order to prevent switch congestion.

On the egress side, packets are received from the switch, already labeled with their metarouter and meta-interface numbers. The LC uses these to map packets to outgoing queues, and to specify output formatting. The egress-side software also monitors the rate at which packets are received on each meta-interface, and raises an exception to the GBP control software, if the received rate exceeds the allowed rate for a given virtual interface. It is then up to the GBP control software to decide what action needs to be taken, if any.

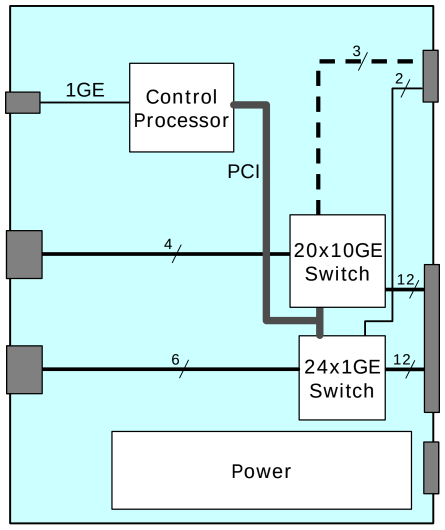

Figure 6. Simplified Block Diagram of Radisys ATCA 2210

5.7. Processing Engines

The reference design supports several types of PEs that can be used to implement metarouters. We refer to these as *General Purpose PEs* (GPE), *Network Processor PEs* (NPE) and *Field Programmable Gate Array PEs* (FPE). Users will specify the number of PEs of each type that are needed for their design and for each PE, they will specify its meta-interfaces and its interfaces to the metaswitch (see Figure 1).

The GPEs can be used in one of two modes. In *raw mode*, the entire GPE blade is under the complete control of its user. Users may run their own operating system on a GPE in raw mode and are fully responsible for its operation. There is no software to implement substrate functionality on a GPE in raw mode, making it necessary for the system to use the switch fabric and LCs to ensure the necessary isolation. In *cooked mode*, a GPE blade runs a standard GBP OS that provides substrate functionality and allows the blade to be shared by multiple metarouters. In this mode, users may reserve a portion of the blade's resources for their exclusive use, and the system will attempt to accommodate such requests by mapping MPEs to physical PEs where the needed resources are available.

NPEs and FPEs can also be used in raw mode. When an NPE is used in raw mode, the GBP control software boots it using a standard OS kernel, but then configures a user account on the NP's control processors with permissions that allow the user to reconfigure the control processors' OS kernel as well as the software running on the micro-engines. Since an NPE in raw mode contains no substrate functionality, isolation is provided by the switch fabric and LCs. FPEs are handled similarly. In this case, the user specifies a configurable logic bit-file to be downloaded to the FPE and this file is sent to the GLU component on the carrier card, which programs the FPE. Again, it is up to the switch fabric and LCs to provide isolation.

Since NPE and FPE blades have little built-in support to facilitate shared use, it is more difficult to share them in a fully general way. In the case of NPEs however, sharing is feasible if the scope of the metarouter-specific processing is limited. In the next subsection we describe a cooked mode for NPEs that we expect to be useful in certain cases that we expect to be quite common in the GBP.

5.8. Cooked Mode for NPEs

Because Network Processors lack the mechanisms to enable general shared use, it's not practical to try to provide shared usage of the NPEs, in a general sense. However, there is a particular way that NPEs can be shared among metarouters that can be useful in the GENI context. In particular, we expect many metarouters to be naturally decomposable into a *fast path* that handles routine forwarding of packets and an *exception path* that handles packets for which more complex processing is required. NPEs are well-suited to the fast-path processing and the fast-path processing can be organized into a generic framework that allows fast path processing for multiple metarouters to be implemented within a single IXP 2850 subsystem (half of an NPE blade).

The fast path can be viewed as a pipeline with five stages. In the *Demux* stage, packets are received from the switch fabric, with the *metarouter* number (MR) and *meta-interface* number (MI) already inserted into the packet header (they are placed there by the ingress LC). The Demux stage uses the MR number to identify an MR-specific control block and a pointer to an MR-specific code segment that parses the packet header and returns an opaque *Lookup Key* for use by the next stage. The second stage is the *Lookup Stage* that combines the given Lookup Key with the MR number to perform a lookup in the on-board TCAM. The first matching entry in the TCAM is returned as the lookup result, which includes an output MI and some MR-specific results. These are used in the next stage, the *Header Formatting* stage, which includes an MR-specific code segment that formats the header for the outgoing packet, which is then placed in a per MI queue. There is also a queue for exception packets, which are forwarded to a GPE for exception processing. The fast path processing pipeline is illustrated in Figure 7.

Note that the per-MR code segments in the Parsing and Header Formatting stages must be restricted to ensure that the different MRs can co-exist without interference. In particular, they are restricted in the memory they can access and they must be free of unbounded iteration or recursion. These restrictions can be enforced using a combination of static and dynamic checks. Alternatively, they can be enforced by requiring that users specify their code in a spe-

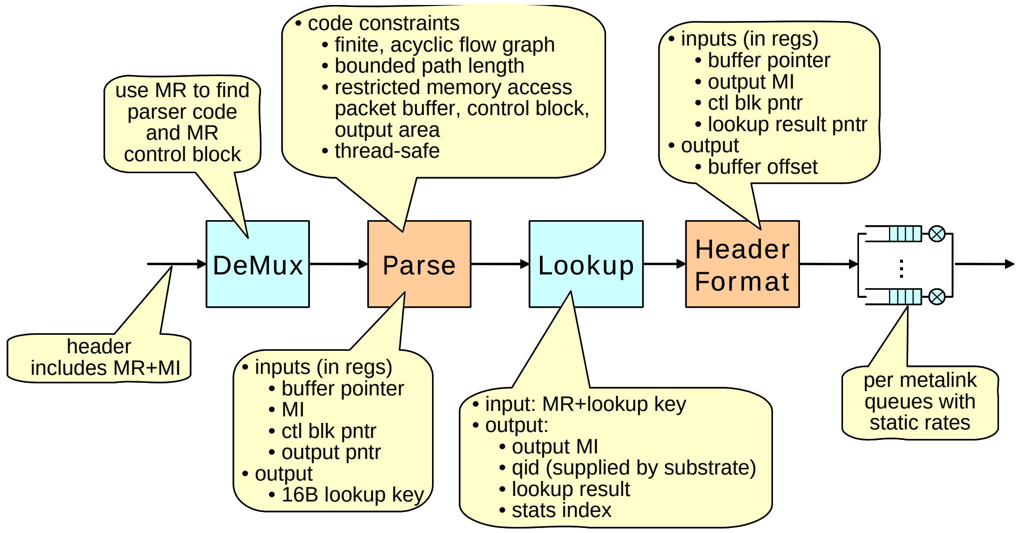
use MR to find parser code and MR control block
• code constraints • finite, acyclic flow graph • bounded path length • restricted memory access packet buffer, control block, output area • thread-safe
• inputs (in regs) • buffer pointer • output MI • ctl blk pntr • lookup result pntr • output • buffer offset
DeMux
Parse
Lookup
Header Format
header includes MR+MI
• inputs (in regs) • buffer pointer • MI • ctl blk pntr • output pntr • output • 16B lookup key
• input: MR+lookup key • output: • output MI • qid (supplied by substrate) • lookup result • stats index
per metalink queues with static rates

Figure 7. Cooked mode NPE Processing pipeline

cially designed language that enforces the necessary constraints by construction. Since the purpose of these code segments is very limited, these restrictions pose no serious constraints on the MRs. Note that MRs that cannot live with the constraints imposed by the cooked mode always have the option of using a raw NPE blade.

6. System Configurations

The baseline configuration shown in Figure 4 comprises two chasses, an ATCA chassis and a general purpose blade server chassis. The ATCA chassis has 14 slots, two that are for the switch blades and 12 that can be used for either LCs or PEs. A typical blade server has a comparable number of slots that can each be equipped with general purpose processing blades, often with dual processors operating in a shared memory mode. The system is designed to be very flexible and can support different mixes of cards of the various types. For the GBP application however, we expect most of the slots in the ATCA chassis to be devoted to PEs of various types rather than LCs. This is to allow users to experiment with networks that do more extensive processing than is typically done in conventional routers, and to relieve researchers of the need to highly optimize their designs to get the maximum possible performance. With this in mind, we expect a typical configuration to include three times as many slots for PEs as for LCs, so the ATCA chassis might include 3 slots for LCs and 9 for PEs. Of the PE slots, we would expect most to be used for NPEs with perhaps 1 or 2 for FPEs. We expect all users to require general purpose processing resources for control purposes, and many GENI users will likely use GPEs for data forwarding as well, since GPEs offer a more familiar development environment in which it is easier to develop and test experimental systems. Because the IO capability of GPEs is relatively limited (1-2 Gb/s per blade), having a fully configured blade server to go along with the ATCA chassis makes sense.

6.1. Directly Connected Multi-Chassis Systems

The simplest way to scale up the baseline configuration to is to replicate it and connect the ATCA chasses to one another using direct connections, as illustrated in Figure 8. Here, we have three subsystems, each consisting of an ATCA chassis and a general purpose blade server. In each pair, the ATCA chassis and blade server are connected by a pair of 10 Gb/s links, while each pair of ATCA chasses is connected by six 10 Gb/s links. This gives each chassis 120 Gb/s of inter-chassis bandwidth. While this is considerably less than the intra-chassis bandwidth (each ATCA chassis has an internal switching capacity of 240 Gb/s), it is ample if all PEs used by a single metarouter are clustered within the same chassis. In this case, inter-chassis bandwidth is only used to gain access to LCs that terminate physical links in other chasses. If each chassis has only three LCs, each terminating a 10 Gb/s link, 30 Gb/s of inter-chassis bandwidth is sufficient to handle the worst-case in this configuration. Constraining PEs to a single ATCA chassis does mean that no single metarouter can scale up to use more than nine PEs, but since we expect the vast majority to use no more than one or two PEs (indeed many will use a fraction of a PE), this appears to be an acceptable limitation. While the inter-chassis bandwidth can support metarouters that have PEs in multiple chasses, it may be challenging to manage their configuration in a way that ensures nonblocking performance and effective traffic isolation.

The direct connection approach can be used for systems with 2, 3, 4 or 7 chasses. In systems with 4 or 7 chasses, some inter-chassis traffic may require two hops, but the inter-chassis bandwidth is sufficient to accommodate this.

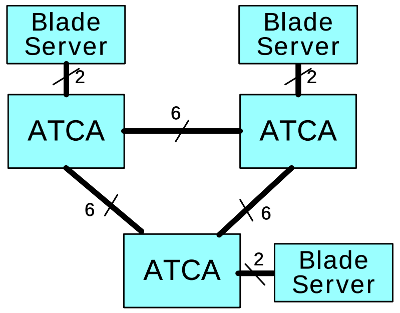

Figure 8. Multi-chassis Configuration with Direct Connections

Note that a 7 chassis system has 84 slots in its ATCA chasses that can be used for LCs, NPEs or FPEs and 98 slots for GPEs.

6.2. Larger Multi-Chassis Configurations

The direct connection approach while conceptually simple offers limited scalability. Larger systems can be assembled using 10 GE switches, such as the FM2224-EP from Fulcrum Microsystems, which is a single board 10 GE switch with 24 ports and full VLAN support. A system using 12 such switches can be used to interconnect 24 ATCA chasses, through the switch uplink ports. Such a configuration would provide 288 slots for LCs, NPEs and FPEs plus 336 for GPEs. If each ATCA chassis had three LCs, the system as a whole, would terminate 72 10 Gb/s links, providing 720 Gb/s of system IO capacity.

6.3. Smaller Configurations

While high performance systems will be needed for the GENI backbone, the GENI testbed will also require access routers at university sites, to act as gateways that feed traffic into the backbone. Smaller scale configurations of the GBP, perhaps with a different mix of PEs, can be useful in this context. A single ATCA chassis with one or two 10 Gb/s LCs, plus a mix of GPEs and NPEs could be suitable for this application. Smaller ATCA chasses (8 slot and 5 slot) with a single switch blade, rather than a redundant pair can also be used in such settings.

7. Closing Remarks

The reference design described in this paper represents just one of a number of possible system architectures for the GBP. The purpose in putting it forward is to provide one example of a design that is sufficiently specific and detailed that it can serve as a reference point for consideration of alternative approaches. It is quite likely that the proposed design will not serve the needs of all researchers who would like to use GENI. We hope that by putting a reference design on the table, the broader research community will help identify possible shortcomings and recommend ways in which they can be eliminated. The more feedback that researchers provide to those interested in designing and implementing the GBP, the more likely it is that the resulting system will meet the needs of the largest number of prospective users.

References

[AN05] Anderson, Tom, Larry Peterson Scott Shenker and Jonathan Turner. “Overcoming the Internet Impasse through Virtualization,” *IEEE Computer Magazine*, 4/05.

[CH02] Choi, Sumi, John Dehart, Ralph Keller, Fred Kuhns, John Lockwood, Prashanth Pappu, Jyoti Parwatikar W. David Richard, Ed Spitznagel, David Taylor, Jonathan Turner and Ken Wong. “Design of a High Performance Dynamically Extensible Router,” *Proceedings of the DARPA Active Networks Conference and Exposition*, 5/02.

[CH03] Chun, Brent, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. “PlanetLab: An Overlay Testbed for Broad-Coverage Services,” *ACM Computer Communications Review*, vol. 33, no. 3, 7/03.

[FLCM] Fulcrum Microsystems 24-Port 10-Gigabit Ethernet Switch Chip. `http://www. fulcrummicro .com/fm2224.htm`.

[GE06] Global Environment for Network Innovations. http://www.geni.net/, 2006.

[IXP] Intel IXP 2xxx Product Line of Network Processors. `http://www.intel.com/design/network/ products/npfamily/ixp2xxx.htm`.

[NLR] National Lambda Rail. `www.nlr.net`.

[NW05] Report of NSF Workshop on Overcoming Barriers to Disruptive Innovation in Networking. `www.arl .wustl.edu/netv/noBarriers_final_repo rt.pdf`, 1/05.

[PCMG] PCI Industrial Computer Manufacturers Group. “AdvancedTCA Specifications for Next Generation Telecommunications Equipment,” available at `http: //www.picmg.org/newinitiative.stm`.

[PE02] Peterson, Larry, Tom Anderson, David Culler and Timothy Roscoe. “A Blueprint for Introducing Disruptive Technology into the Internet,” *Proceedings of ACM HotNets-I Workshop*, 10/02.

[RA05] Radisys Corporation. “Promentum™ ATCA-7010 Data Sheet,” product brief, available at `http://www.radisys.com/files/ATCA-7010_07-1283-01_0505_datasheet.pdf`.

[VRTX] Virtex 5 Multiplatform FPGA. `http://www. xilinx.com/products/silicon_solutions /fpgas/virtex/virtex5/index.`