Removing Cell Demultiplexing Performance Bottleneck in ATM Pseudo Wire Emulation over MPLS Networks

Puneet Konghot and Hao Che
{pxk7783@omega.uta.edu, hche@cse.uta.edu}
The Department of Computer Science and Engineering
University of Texas at Arlington

ABSTRACT

Cell multiplexing is highly desirable to achieve increased transport efficiency in Asynchronous Transfer Mode (ATM) pseudo wire emulation over a multiprotocol label switching (MPLS) network. Supporting cell multiplexing, however, can create performance bottleneck at the egress provider edge (PE) where fully programmable network processors are used to do the cell demultiplexing.

In this paper, a scheme is proposed to allow a large number of cells to be demultiplexed, without having to implement special ASIC-based hardware optimized for cell demultiplexing. The idea is to configure a sequence of concatenated data plane processing devices to share the cell demultiplexing task with the egress PE. Performance analysis shows that the use of just two processing devices is sufficient to achieve high transport efficiency. This may as well be achieved by the addition of one router without a central control card. Therefore, service providers can employ this scheme to enable cell multiplexing with limited add-on cost, while preserving the existing investment.

1 Introduction

The infrastructure of service providers providing multiple services consists of heterogeneous networks wherein each network implements a specific service like Frame Relay, ATM, etc. Providing interworking functions among heterogeneous networks is expensive in terms of capital, operational, management and planning costs. These are the main motivations for service providers to attempt to consolidate the delivery of multiple services through a single Packet Switched Network (PSN). This is achieved by emulating existing services or circuits by mapping the traffic of these services on to point-to-point tunnels across an MPLS PSN backbone. The mechanisms used to emulate the essential attributes of a service over a PSN are referred to as Pseudo Wire Emulation Edge-to-Edge (PWE3). The Internet Engineering Task Force (IETF) PWE3 Work Group [1] is working on standardizing PWE3 [2,3]. The "Martini Drafts" [4,5] (named after Luca Martini of Level 3 Communication), which describe transport of Layer 2 frames over MPLS PSN, are getting the most attention from the service providers and vendors.

In particular, four different encapsulation modes have been defined for transporting ATM services over an MPLS PSN [5], including ATM one-to-one cell mode, ATM n-to-one cell mode, ATM Adaptation Layer 5 (AAL5) CPCS-SDU mode, and AAL5 PDU frame mode. Unlike the last two AAL5-based modes, the first two cell-based modes do not require ATM Segmentation and Reassembly (SAR) at PE routers and they allow the support of all AAL types. To achieve increased transport efficiency, the two cell modes allow multiple cells to be concatenated and encapsulated in the same Pseudo Wire (PW) Protocol Data Units (PDUs).

As long as the limits of the Maximum Transfer Unit, and the cell transfer delay and cell delay variation are satisfied, transport efficiency can be maximized by allowing a large number of ATM cells to be multiplexed in a single PW PDU. However, the demultiplexing capabilities at egress PE can be a major bottleneck in this regard. An egress PE needs to break a PW PDU into a potentially large number of ATM cells and then transmit them one-by-one through CE bound interfaces.

In particular, a PE, using Fully Programmable Network Processors (FPNPs) to carry out the data path functions in its Network Interface Cards (NICs), may have this bottleneck issue. This is because most of the FPNP solutions are not optimized to perform instruction intensive fragmentation tasks. For example, the INTEL IPX1200 FPNP has 6 micro engines in it. A typical configuration is to employ a two-stage pipelined architecture with 4 micro engines in stage one for receiving and processing packets, and the other 2 in stage two for transmitting packets. Each micro engine runs up to 4 threads. Hence, up to 16 packets in stage one can be processed simultaneously and they share the rest of 8 threads in stage two for transmission. Clearly, if instruction intensive cell demultiplexing is to be supported, the second stage can become a potential bottleneck because many cells may simultaneously compete for the transmission threads. Similar issue arises for FPNPs employing a parallel architecture. For example, the AMCC

NP7120 FPNP has 2 micro engines working in parallel, with each running 8 threads. The execution of packet fragmentation and transmission is through a channel command FIFO. Hence, there can be up to 8 packets competing for the same command channel for packet fragmentation and transmission. Again, this can create a potential bottleneck if a large number of channel commands are issued per packet, which can actually be the case for cell demultiplexing.

In this paper, we focus on solving this bottleneck issue for PEs that use FPNPs to perform data path functions. A scheme is put forward which is scalable and allows a sufficiently large number of ATM cells to be demultiplexed at egress PEs. For service providers whose PE devices suffer from this bottleneck issue, the proposed scheme provides a viable solution for the immediate deployment of PW services over an MPLS PSN.

The rest of the paper is organized as follows. Section 2 briefly introduces the PWE3 over MPLS PSN architecture. Section 3 describes the proposed scheme, including its architecture and implementation considerations. Section 4 analyzes the performance. Finally, Section 5 concludes the paper.

2 Architecture of PWE3 over MPLS PSN

PWs function by encapsulating the service-specific PDUs arriving at an ingress port, carrying them across a tunnel, managing their timing and order, and providing any other operations required to emulate the behavior and characteristics of the service as faithfully as possible. From the perspective of a customer edge (CE) equipment, the PW is characterized as an unshared link or circuit of the chosen service.

2.1 Reference Model

The network reference model for PWs over MPLS PSN is shown in Fig 1. Layer 2 protocol data units are carried across the PSN by prepending an MPLS label stack to these PDUs. This label stack has at least two labels. The bottom label corresponds to the PW (a MPLS label switched path (LSP)) and is called the PW label. The label immediately above the PW Label corresponds to the PSN Tunnel LSP and it is called the PSN tunnel label.

As shown in Fig 1, the PSN tunnel is an LSP that merely takes packets from PE1 to PE2. The disposition of the packets after they reach PE2 is decided by the PW Label, e.g. the CE bound outgoing interface is indicated by the PW label. When PE1 receives a packet from CE1, it first pushes a PW label on its label stack and then (if PE1 is not adjacent to PE2) pushes on a PSN tunnel label. The PSN tunnel label takes the MPLS packet from PE1 to PE2; the PW label is not visible until the MPLS packet reaches PE2. In other words, the PW traffic is invisible to the core MPLS PSN, and the core MPLS PSN is transparent to the CEs.

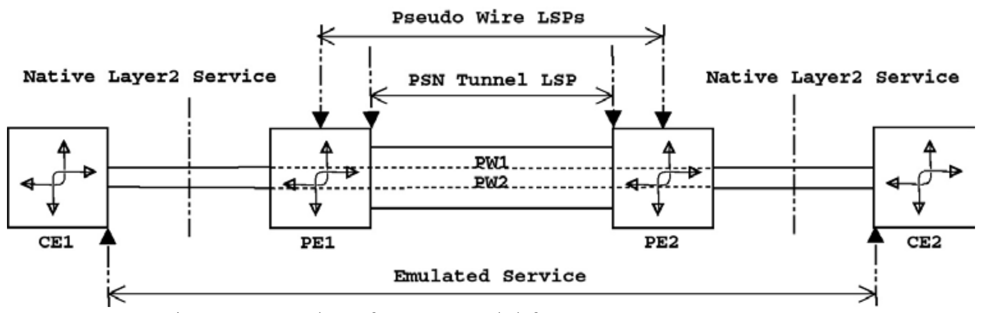

Fig 1 : Network Reference Model for PWE3 over MPLS PSN

An emulated service can be established only after a PW has been set up. This may be accomplished by static label assignment or by signaling. Moreover, to support the cell multiplexing, the ingress and egress PEs need to reach an agreement on the maximum number of cells to be multiplexed for any given PW. This agreement may be accomplished via a PW specific signaling mechanism or via static configuration.

2.2 ATM Cell Modes

[6] defines two ATM cell modes, i.e., *one-to-one cell mode* and *n-to-one cell mode*. In the one-to-one cell mode, one ATM virtual channel connection (VCC) or virtual path connection (VPC) is mapped to a single PW. In ATM

VCC cell transport service, one or more cells from one ATM VCC are encapsulated in a PW PDU. In this case the VPI/VCI is not included since the PW is known to carry cells from one particular VCC. At the egress PE, the PW label is used to decide the outgoing interface and VPI/VCI. The ATM control byte must be repeated for each cell and hence 49 bytes are needed to encapsulate each 53-byte ATM cell.

The ATM VPC cell transport service emulates a VP cross-connect across the PSN and all the VCCs belonging to the VPC are carried transparently by the VPC service. The VPI value is not included in the PW PDU but the VCI is carried unchanged. The egress PE decides the outgoing VPI solely based on the PW label. One or more cells from one ATM VPC are encapsulated in a PW PDU. The ATM control byte must be repeated for each cell and hence 51 bytes are needed to encapsulate each 53-byte ATM cell.

The ATM n-to-one mode is the only mandatory mode for ATM transport over PWE3 and it allows the mapping of multiple ATM VCC or VPC to a single PW/PSN Tunnel. One or more cells from multiple ATM VCC/VPC are encapsulated in a PW PDU. Both VPI & VCI fields are included and hence 52 bytes are required to transport each ATM cell.

In summary, there are three cell multiplexing formats, $f = 1, 2, 3$, two corresponding to the one-to-one cell mode, i.e., ATM VCC and VPC, and one corresponding to the n-to-one cell mode.

3 Proposed Scheme

In this section, we propose a scheme to remove the cell demultiplexing bottleneck at an egress PE for the support of all three cell multiplexing formats.

3.1 Architecture

The idea is to configure a series of concatenated devices with forwarding capabilities, including the PE device itself, to work as a *virtual* PE. The cascaded devices except the PE device are programmed to serve the purpose for partial PW PDU fragmentation. Each of these devices consists of a set of packet processing units. The central component of a processing unit is an FPNP. Therefore, we shall use the terms processing unit and FPNP interchangeably.

The cell demultiplexing capability of this virtual PE is exponentially proportional to the number of cascaded devices in use. For instance, with just two cascaded devices in addition to the PE device, which may well be accomplished by the addition of a router with 2M NICs, approximately n^3 number of cells can be demultiplexed, where n is roughly the number of segments into which a PW PDU can be broken up by a FPNP and transmitted. For example, for n = 4, this virtual PE can demultiplex 64-cells.

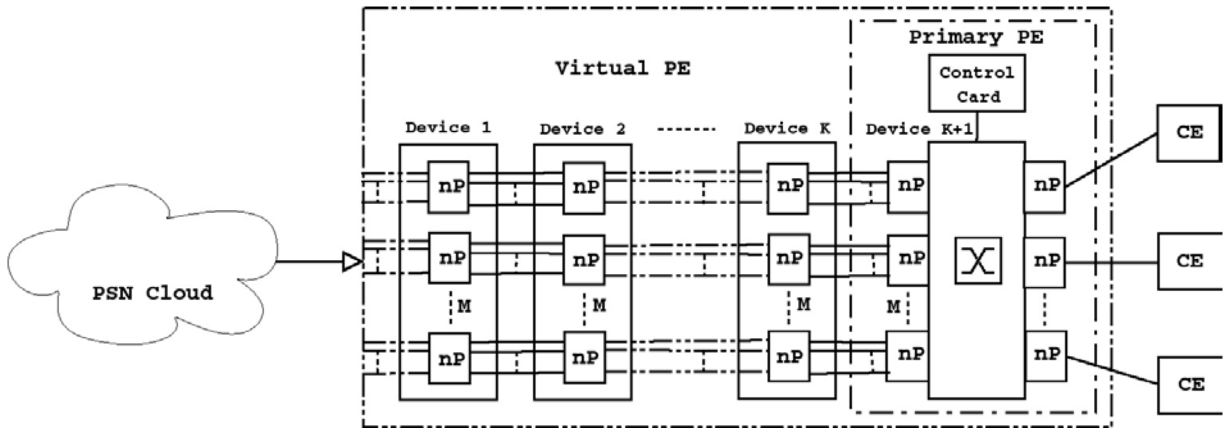

Fig 2. Virtual PE Architecture

Fig 2 shows the architecture of a virtual PE which comprises of $K+1$ devices in cascade. The $(K+1)$-th device closest to the CE side is a full-fledged PE device with fully operational control plane and data plane functions. We call it the *primary PE*. The other K devices are capable of performing the data plane functions but are not required to perform any control plane functions.

Suppose the primary PE has M NICs on the trunk side (i.e., PSN bound) supporting cell mode based PWs. Then each of the other K devices is required to have M comparable data plane processing units. The interfaces or ports associated with each processing unit are configured to be identical to the ones associated with the corresponding NIC in the primary PE. The devices are concatenated in such a fashion that all the corresponding ports are serially interconnected.

One possible implementation to create K concatenated devices is to simply use a router with M pairs of NICs interconnected through the switching fabric to generate a pair of back-to-back devices. The control card may be absent to reduce the cost. K even number of devices can then be created with K/2 identical routers in cascade. Another possible solution is to design one or multiple hardware boards to accommodate M arrays of K cascaded FPNPs.

The control card in the primary PE serves as the "brain" or the control unit for the entire virtual PE. The K devices simply pass all the traffic going through them, except for PWs in cell modes and some other control messages, as will be explained shortly. The FPNPs in these devices is programmed to handle partial fragmentation tasks for a PW PDU in a cell mode. The following describes how this can be done.

Suppose the j-th FPNP in the k-th device is capable of fragmenting a PW PDU of multiplexing format f into $N(f, l_s; j, k)$ segments of l_s cells each ($f = 1, 2, 3; j = 1, 2, \ldots, M; k = 1, 2, \ldots, K+1$), while keeping up with the line rate. Note that for $k = K+1$, *i.e.,* the primary PE, l_s must equal 1. In other words, an ingress NIC in the primary PE must be able to demultiplex an incoming PW PDU into individual cells and sends them one-by-one to egress NICs on the CE side. With this, we observe that *the maximum number of cells the primary PE can demultiplex is* $N(f, 1; j, K+1)$.

We further make use of the following property: *$N(f, l_s; j, k)$ is a non-decreasing function of l_s.* This property holds simply because the larger the segment size is, the more time a FPNP has at its disposal to break a PDU into fragments of that size. So, potentially, the FPNP can generate more segments as the segment size increases, without compromising the throughput performance.

With the above observation and property, we have the following iterative procedure to find the maximum number of multiplexed cells, $L(f)$, the entire virtual PE can support and the maximum segment size $m(f; j, k)$ the j-th FPNP in the k-th device can let through.

```
ALGORITHM-1 ($f;\ j, K$)
Initialization:  $L(f) = N(f, 1; j, K+1)$
                 $m(f; j, K+1) = L(f)$
LOOP:  FOR ($k = K$ to 1, step 1)
        $L(f) = N(f, L(f); j, k)\, L(f)$;
        $m(f; j, k) = L(f)$;
       END
```

Now the idea is first to get an estimate of $N(f, l_s; j, k)$ for a device. Then for desired $L(f)$ ($f=1, 2, 3$), iteratively run ALGORITM-1 to find (1) a minimum K value which can support all desired $L(f)$ ($f = 1, 2, 3$); (2) $m(f; j, k)$; (3) $N(f, m(f; j, k); j, k)$, for $f=1, 2, 3$, $j = 1, 2, \ldots, M$ and $k = 1, 2, \ldots, K+1$. The following gives the exact algorithm:

```
ALGORITHM-2
$K=0$
WHILE ()
  ALGORITHM-1($f; j, K$);
  IF ($L(f) >=$ desired $L(f)$) EXIT;
  ELSE $K = K + 1$;
END
```

The j-th FPNP in the k-th device is then programmed to break a PW PDU of framing format f into segments with sizes no larger than $m(f; j, k)$. With this setup, the control CPU in the primary PE will then be able to negotiate with the PEs on the other side of the PSN to decide the exact PW framing format f, and the maximum PW PDU size upper bounded by $L(f)$.

However, in general, $N(f, l_s; j, k)$ is a function of aggregated bandwidth for all the ports associated with the FPNP j in the k-th device as well as the traffic pattern. The higher the bandwidth a FPNP has to support, the smaller its instruction budget is for per packet processing, and thus smaller $N(f, l_s; j, k)$. Also the higher the percentage of the cell multiplexed PW traffic is, the smaller the $N(f, l_s; j, k)$ is. For this reason, the static setup described above may not always result in desired performance. Throughput bottlenecks may be created in some devices due to overly optimistic estimation of $N(f, l_s; j, k)$ or traffic pattern changes over time. Likewise, a too conservative estimation of $N(f, l_s; j, k)$ or traffic pattern changes over time may result in under utilization of the FPNP capacity while $L(f)$ falls short of its desired value. Therefore, in our architecture, a key requirement is to allow *in-service*, *dynamic adjustment* of the functional relationship $N = N(f, l_s; j, k)$ to ensure that neither throughput bottlenecks are created nor are resources under utilized.

Note that for in-service update, K cannot be changed, which involves adding or removing devices. The changes are made on $\{N(f, m(f; j, k); j, k), m(f; j, k)\}$. A simple approach is to reduce (increase) $N(f, l_s; j, k')$ by an integer number if the j-th FPNP in the k'-th NIC is overloaded (under loaded). Then run the "FOR" loop in ALGORITHM-1 from k' to 1 to update $L(f)$ and $\{N(f, m(f; j, k); j, k), m(f; j, k)\}$ values for $k = 1, 2, ..., k'$. Note that $\{N(f, m(f; j, k); j, k), m(f; j, k)\}$ for $k = k', ..., K+1$ stay unchanged.

3.2 Implementation Considerations

Our architecture allows the *same* piece of micro-code to be run in all the devices except the primary PE where other data path functions also need to be taken care of, in addition to the cell demultiplexing function. However, this micro-code can be used, with minor changes, to carry out the cell demultiplexing function in the primary PE. For this reason, in what follows, we focus on the implementation considerations for the devices other than the primary PE.

To allow in-service updates, the parameters $\{N(f, m(f; j, k); j, k), m(f; j, k)\}$ must not be hard coded. Instead, some global registers should be used to store these parameter pairs. For example, these parameter pairs may be stored in two 32-bit global registers in the FPNP. The lower 24 bits of each register can be used to store the values corresponding to the three different multiplexing formats, each taking 8 bits of the register memory. Each FPNP also maintains a *PW table* in its memory, which maps the PW MPLS label (the inner label in the MPLS label stack) and the input port number, from which the PW comes, to a multiplexing format f.

For the inbound traffic flowing from the PSN into the virtual PE, as soon as enough header information is received, the FPNP identifies whether the incoming frame is from an MPLS tunnel or not. If not, the frame is forwarded to the next device without modification. Otherwise, the PW table is looked up to find the f value. With this f value, the FPNP then reads the $m(f; j, k)$ value from the corresponding global register. Based on this value, the FPNP segments the frame into segments of length $m(f; j, k)$ cells each except the last one, which can be smaller than $m(f; j, k)$. The FPNP increments a counter by one and compares it with $N(f, m(f; j, k); j, k)$ in the other global register each time a segment is generated and sent. If the counter exceeds $N(f, m(f; j, k); j, k)$, the rest of the frame is dropped and an error message is sent to the primary PE.

For outbound traffic to PSN, in the case of static setup, a device other than the primary PE simply passes the frames, whether they are PW frames or not, to the next-hop device. However, in the case of dynamic setup as proposed in our architecture, the FPNPs must be able to receive control messages from the control card in the primary PE. More specifically, the FPNPs in all the devices in the virtual PE must be able to (1) collect the overall throughput and frame loss statistics; (2) send a message to the control card CPU in the primary PE, whenever it incurs low throughput due to either resource under run with no frame loss or resource overrun with large number of frame losses; (3) receive control frames containing an updated $\{N(f, m(f; j, k); j, k), m(f; j, k)\}$ pair from the control card in the primary PE; (4) update $\{N(f, m(f; j, k); j, k), m(f; j, k)\}$ values in the two global registers; (5) send an ACK back to the control card in the primary PE after each successful update.

FPNPs are generally capable of collecting statistics required in (1). An FPNP can be easily programmed to meet the requirement in (2) by comparing the throughput with a threshold upon each throughput statistic update. Since the communications between any FPNP and the control card CPU in the primary PE are purely internal to the virtual PE, the actual messaging formats used in (2), (3), and (5) can be implementation specific. For example, if the data passing between devices is encapsulated in the Ethernet II or IEEE802.3 frame format, unused ETHERTYPE codes can be selected to identify different types of internal message formats. FPNPs are generally designed with the ability to execute, delete, and then ACK a small piece of code, known as an applet, received from a control CPU. Hence, (3)-(5) can be achieved using applets.

4 Performance Analysis

Transport efficiency of a PW can be defined as the ratio of actual data bits transmitted to the total number of bits transmitted in PWE3 packet. Fig 3 presents a plot of transport efficiency against the number of ATM cells encapsulated in a PWE3 packet. It shows that the transport efficiency increases with the size of the PWE3 packet (in terms of number of encapsulated ATM cells) but begins to saturate for more than 15 cells per packet. This is because the 'per cell overhead' has to be carried even when multiple cells are concatenated into a single PWE3 packet.

As an example, consider a case where there are two devices apart from the primary PE. If the primary PE is assumed to have fragmentation capability of just 1 segment and the other two devices have maximum fragmentation capability of 3 segments each. This results in a total fragmentation ability of 9 segments for the entire virtual PE. Without cell concatenation a transport efficiency of 80% can be achieved for n-to-1 VCC/VPC mode whereas with the virtual PE setup mentioned above, the efficiency would be about 91%. This 11% improvement is significant for two reasons. Firstly, 11% represents a large amount of saving in transport capacity. For example, for a NIC supporting an aggregated OC-48 line rate, about 250 *Mpbs* transport capacity can be saved. Secondly, concatenation of a large number of ATM cells into PW PDUs implies that the core routers have to handle lesser number of packets. Also, since these packets are larger in size, the FPNPs in the core routers get more time to process the packets. This can help in improving performance of core routers, which have to forward packets at line rates.

Another important advantage of this scheme is that it reduces the load on the egress PE by distributing the fragmentation operations over the other devices in the virtual PE. This can be particularly useful when the egress PE is overloaded and does not have enough instructions available to handle fragmentation operations. The savings in instruction budget at the PE helps to deploy PWE3 without affecting the PEs performance for existing packet switched traffic. Moreover, in general, a virtual PE with 2 additional devices would improve the transport efficiency to almost the maximum possible value.

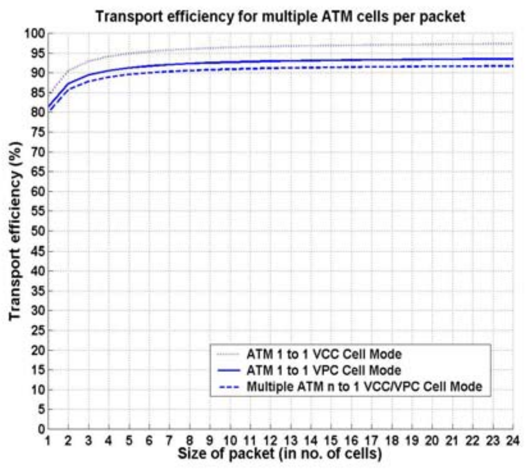

Fig 3. Transport efficiency Vs PWE3 packet size

5 Conclusions

This paper has proposed a method for service providers to provide PWE3 services over their existing MPLS PSNs, with minimum additions and modifications to their existing infrastructure. This scheme provides a cost effective way to improve the transport efficiency of the PWE3 service and at the same time reduces the load on the core and edge routers of the MPLS PSN.

Acknowledgement:
The authors would like to thank Chethan Kumar on the discussion of INTEL IXP1200 architecture.

6 References

[1] PWE3 IETF Workgroup web site, http://ietf.org/html.charters/pwe3-charter.html
[2] X. Xiao, D. McPherson, and P. Pate, "Requirements for Pseudo-Wire Emulation Edge-to-Edge (PWE3)", *IETF draft-ietf-pwe3-requirements-04.txt*, June 2003.
[3] S. Bryant and P. Pate, "PWE3 Architecture", *IETF draft-ietf-pwe3-arch-01.txt*, May 2003.
[4] L. Martini, N. El-Aawar, E. Rosen, T. Smith, and G. Heron,"Transport of Layer 2 Frames Over MPLS", *IETF draft-ietf-pwe3-control-protocol-01.txt*, May 2003.

[5] L. Martini, M. Bocci, G. Koleyni, J. Brayley and E. Rosen, “Encapsulation Methods for Transport of ATM Cells/Frame Over IP and MPLS Networks”, *IETF draft-ietf-pwe3-atm-encap-00.txt*, April 2003.