# Packet Prediction for Speculative Cut-Through Switching

Paul Congdon
University of California, Davis
1 Shields Avenue
Davis, CA 95616
+1 916 765 4056
ptcongdon@ucdavis.edu

Matthew Farrens
University of California, Davis
1 Shields Avenue
Davis, CA 95616
+1 530 752 9678
farrens@cs.ucdavis.edu

Prasant Mohapatra
University of California, Davis
1 Shields Avenue
Davis, CA 95616
+1 530 754 8016
prasant@cs.ucdavis.edu

## ABSTRACT
The amount of intelligent packet processing in an Ethernet switch continues to grow, in order to support of embedded applications such as network security, load balancing and quality of service assurance. This increased packet processing is contributing to greater per-packet latency through the switch.

In addition, there is a growing interest in using Ethernet switches in low latency environments such as high-performance clusters, storage area networks and real-time media distribution. In this paper we propose Packet Prediction for Speculative Cut-through Switching (PPSCS), a novel approach to reducing the latency of modern Ethernet switches without sacrificing feature rich policy-based forwarding enabled by deep packet inspection.

PPSCS exploits the temporal nature of network communications to predict the flow classification of incoming packets and begin the speculative forwarding of packets before complex lookup operations are complete.

Simulation studies using actual network traces indicate that correct prediction rates of up to 97% are achievable using only a small amount of prediction circuitry per port. These studies also indicate that PPSCS can reduce the latency in traditional store-and-forward switches by nearly a factor of 8, and reduce the latency of cut-through switches by a factor of 3.

## Categories and Subject Descriptors
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – *packet switching networks, store and forward networks*

## General Terms
Performance, Design

## Keywords
Ethernet Switching, Cut-Through, Speculation, Packet Prediction

## 1. INTRODUCTION
Ethernet and IP communications have become the most popular means of computer communications, in part due to the simplicity and scalability of connectionless packet oriented communications over a statistically multiplexed network. As Ethernet moves to 10 Gbps speeds and beyond, there is a strong desire to use this commodity technology in specialized high performance parallel computing environments where traditionally specialized interconnect fabrics have been deployed (e.g. Infiniband, Myrinet and Quadrics[1]). Interconnect fabrics designed for Message Passing Interface (MPI) applications, for example, are focused on connection oriented, low latency and high-bandwidth communications, but they are often complex and expensive [2, 3]. Using commodity Ethernet packet switches instead of specialized interconnection fabrics can lower both cost and complexity, but current packet switches suffer from excessive switch latency.

There are also many advantages to extracting information from various fields in each packet and performing policy based forwarding decisions based on that data. Applications such as firewalling, intrusion detection/prevention, connection rate metering and load balancing all rely upon deep packet inspection and rapid flow classification of each packet. In addition, there is a trend towards multiplexing a variety of different traffic types (e.g., voice, video and data), each potentially with different service requirements, onto the same network fabric. In all of these situations, low latency, yet policy rich forwarding based upon flow classification is strongly desired.

Providing these rich forwarding features without negatively impacting the switch latency puts extreme pressure on the classification process of a network switch. The problem of packet classification has been well studied [4, 5] and is known to be a compute intensive step in the switch forwarding process. Specific hardware support is often used to improve performance, but at considerable expense and without entirely eliminating the classification bottleneck. Space efficient searching schemes, such as Bloom filters, have also been shown to reduce the amount of resources required to match packets against a set of rules [6, 7].

Cut-through switches have been designed to provide the lowest possible latency by allowing a packet to begin transmission on an egress port before the packet has been completely received at the ingress port [19, 20]. However, this approach seriously impacts the amount of deep packet inspection that can be performed. There is simply no time in a traditional cut-through switch to inspect transport or application level fields to perform firewalling, intrusion detection and load balancing before the packet is switched. In addition, traditional cut-through switches operate on a packet by packet basis and take no advantage of the temporal locality of network communications.


*ANCS'08*, November 6–7, 2008, San Jose, CA, USA.

This paper presents Packet Prediction for Speculative Cut-through Switching (PPSCS), a novel approach to reducing switch latency while maintaining the rich policy based forwarding features of modern packet based routing switches. The basic idea is to apply the architectural techniques of value prediction and speculative execution to the problem of packet switching. In PPSCS, a packet predictor takes advantage of the temporal locality of network communications to speculate on the flow membership of the next received packet. Knowing the flow membership of a packet allows the forwarding engine of the switch to apply a rich set of forwarding policies to the packet while it is still being received. Predicting the packet's flow membership allows this rich policy based forwarding to begin with the lowest possible latency.

The observation that network communications has strong locality and that this may be used to optimize resource utilization is not new [15]. There are differences of opinion as to whether the temporal locality of Internet traffic is sufficient to enable optimized forwarding using caching [21, 22, 23]. Since traditional Ethernet LAN traffic is an aggregation of a much smaller number of flows than within the core Internet, it is expected to have a greater degree of locality. A preliminary study of the temporal locality in various network traces is described in Section 4.3 of this paper. Ethernet LAN traffic is bursty over varying timeframes and known to be self-similar [16]. A popular model for LAN traffic is the on/off model [17] that shows how the self-similar nature is the result of packet trains. All packets in a packet train are members of the same flow and require the same forwarding treatment by an Ethernet switch.

We believe PPSCS-based switches will be attractive to high performance computing environments and will benefit network communications in general. There is strong evidence that packet prediction is a plausible approach to optimizing switch forwarding, especially for switches that are closer in the network topology to the end-host and its applications [18].

The remainder of this paper is organized as follows. Section 2 describes the switch model used and the measurement of switch latency, while Section 3 describes PPSCS in detail. Section 4 discusses the simulation environment used to evaluate the improvement in latency and Section 5 describes the results of those simulations. Section 6 explores related work, Section 7 discusses limitations and areas for future study and Section 8 provides a conclusion.

## 2. SWITCH ARCHITECTURE

Figure 1 shows a diagram of a scalable switch architecture used when developing the models of a store-and-forward and a cut-through switch in this paper. The model assumes line cards with physical media ports are connected to a passive switch backplane fabric, and the line cards are equipped with separate input and output memory, lookup logic and backplane fabric interfaces. Packets are received at line rate from the physical media ports and put into the input memory. This constitutes the store stage of the store-and-forward switch. The larger the received packet, of course, the longer it takes to store the packet in the input memory. The packet is not known to be error free until it is completely received.

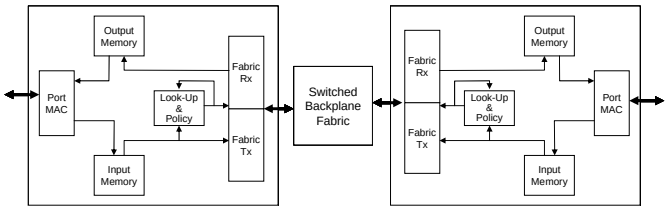


**Figure 1. Basic Packet Switch Architecture**

Once received error free, a variety of lookup steps based on the contents of the packet are performed in order to return the results needed to support policy operations, packet modifications and ultimately the forwarding, redirecting or filtering of the packet. [1] The lookup steps effectively return a set of logical instructions that will be used to process the packet according to the switch configuration and policy. The duration of the lookup steps depends upon the complexity of the switching policy. Functions such as access control lists (ACLs), application rate meters or content aware filtering may require multiple passes through the lookup step.

The fundamental structure used to perform the lookup is a flow key. A flow key is a summarization of critical fields from the packet that uniquely identify the packet as being part of a flow. It can be generalized as an n-tuple that is defined by a set $H = \{H_1, H_2, \ldots, H_n\}$ of fields from the packet. All packets that are part of a flow are subject to the same policy and treatment by the switch. For a typical routing switch that performs layer-2 bridging, layer-3 routing and transport level filtering, a flow key can be represented by a 9-tuple that includes the following fields: VLAN ID, destination MAC address, source MAC address, ethertype, IP protocol number, source IP address, destination IP address, TCP/UDP source port number and TCP/UDP destination port number.

A flow table is a large database of flow keys that is searched by the lookup process. This structure may be implemented in software using SRAM and a fast network processor, or more often implemented in hardware by ternary content addressable memories (TCAMs). TCAMs are an expensive, high performance resource for the switch. The TCAMs may be shared by multiple input ports on the same line card and consequentially may be subject to contention and further arbitration delays. The process of classifying the packet and searching the flow table has been well studied [4, 5], and is known to be a time consuming and critical stage in the switch pipeline with complexity O(log N).

The results of the lookup steps tell the switch where to forward the packet across the switch fabric, and optionally, what modifications to the packet may be required. In this particular model, the receiving line card handles making the necessary modifications to the packet and initiates a transfer of the packet across the fabric to the output memory on another line card. The speed of the backplane fabric interface is usually faster than the speed of the input port and in this model the backplane does not

[1] In some store-and-forward architectures it is possible to begin the lookup process before the entire packet has been received and stored. The unique distinction of a store-and-forward switch is that it does not begin transmitting the packet until it has been fully received.

represent a bottleneck. Once the packet is received in the output memory it may immediately begin transmission on the egress port. The switch model assumes all ports are operating at the same line rate and that there is no contention for the egress port.

## 2.1 Switch Latency

The process of switching a packet can be pipelined in order to increase switch throughput. While the lookup process is working on a packet, the next packet can be copied from the ingress port to the input memory, and the previous packet can be modified and transferred to the output memory of the egress port. Figure 2 shows the pipeline diagram for the store-and-forward switch.

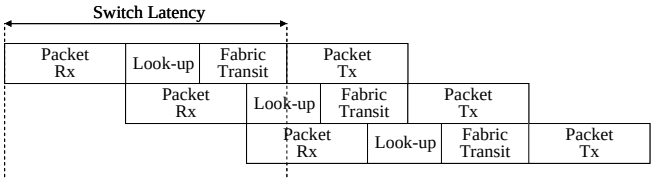


**Figure 2. Store-and-Forward Switch Pipeline**

In order for the switch to maintain line rate forwarding, no stage of the pipeline can exceed the time it takes to receive a packet from the wire. On a 10 Gbps Ethernet port there are potentially 14.88 million minimum size packets arriving per second; therefore, no stage can exceed 67.2 ns. A generalized way to look at the minimum required pipeline stage length is to normalize the stage to received bit times. A minimum size Ethernet packet is 64 bytes and is therefore received in 512 bit times, as determined by the speed of the ingress link.

The duration of the packet Rx and packet Tx stages of the pipeline are directly tied to the physical media line rate. The fabric transit and packet modification stage is faster than the physical media line rate. Therefore, to forward at line rate, the lookup stage and the fabric transit stage must be no longer than the time it takes to receive a minimum sized packet. To simplify the calculation of switch latency the model assumes the lookup stage time will be a constant and equal to the amount of time it takes to receive a minimum size packet.

Let $K_{sf}$ be the number of bits in a minimum size Ethernet packet (which is the constant 512). Let $R_p$ be the received port line rate in bps and let L be the length of the packet in bits. Let $R_f$ be the fabric interface transfer rate in bps and assume that $R_f > R_p$. Switch latency is the amount of delay a packet experiences inside the switch, and will be measured as the amount of time between when the first bit of a packet is received on the ingress port and the time the first bit is transmitted on the egress port. The formula for a store-and-forward switch latency is then:

$$\text{Store and Forward Latency} = (L / R_p) + (K_{sf} / R_p) + (L / R_f) \quad (1)$$

This formula represents the time taken to receive the packet plus the time to perform the lookup stage plus the time to make any modifications and transfer the packet across the fabric. (We assume that packet transmission on the egress port starts immediately once the packet is in the output memory.)

To improve store-and-forward switch latency, two things must change. First, the lookup process and packet modification with transfer across the backplane must begin before the current packet has been completely received. Second, the transmission of the packet on the egress port must also be allowed to begin before the current packet has been completely received (and certainly before it has completed being transferred across the backplane.)

In the cut-through model of a switch without any prediction the lookup process can begin no sooner than after the last bit of the packet needed to construct a flow key has been received. Let $D = \{D_1(H_1), D_2(H_2), \ldots, D_n(H_n)\}$ be the set of functions in the classification process that return the starting bit displacement for the flow key fields in H. Then $D_n(H_n)$ is the starting bit offset for the last field necessary to create the n-tuple needed for the lookup. If we define $K_{ct}$ as the number of bits that must be received to construct the flow key for the lookup stage to begin we have:

$$K_{ct} = D_n(H_n) + |H_n| \quad (2)$$

Assuming that packet modification is part of the fabric transfer stage and the transmission of the received cut-through packet may begin as soon as the first bit has arrived in the output memory, we have the following formula for cut-through switch latency:

$$\text{Cut-Through Latency} = (K_{ct} / R_p) + (K_{sf} / R_p) + 1/R_f \quad (3)$$

This formula represents the time taken to receive enough of the packet to construct the flow key plus the time to perform the lookup stage in order to maintain line rate plus the time to optionally modify the packet and transfer the first bit of the packet across the fabric. We assume that packet transmission on the egress port starts immediately once the first bit of the packet is in the output memory. [2] Figure 3 shows the pipeline diagram for a cut-through switch.

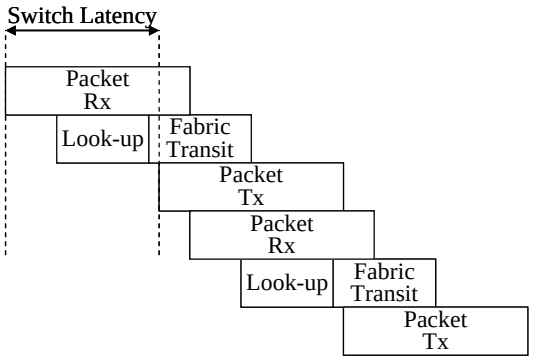


**Figure 3. Cut-Through Switch Pipeline**

# 3. Packet Prediction and Speculation

The switch latency of both store-and-forward and cut-through switches can be reduced by exploiting the architectural techniques of value prediction and speculative execution. Value prediction attempts to remove the limits on parallelism imposed by true data dependencies, while speculation seeks to reduce the latency of obtaining computed results. In an Ethernet switch, the lookup stage is a data dependent operation that requires the reception of enough packet data to construct a flow key, and the operations applied to the packet once the lookup completes must endure the lookup latency before execution can begin. Packet prediction and speculative switching remove this barrier by exploiting the

[2] Real implementation will typically transfer the packet in blocks across the fabric and may wait for the entire frame to be transferred to avoid the complications of under-run management when egress and ingress port speeds differ.

temporal locality of network traffic to predict the data being received, allowing speculative packet operations for the flow to begin before the lookup has completed.

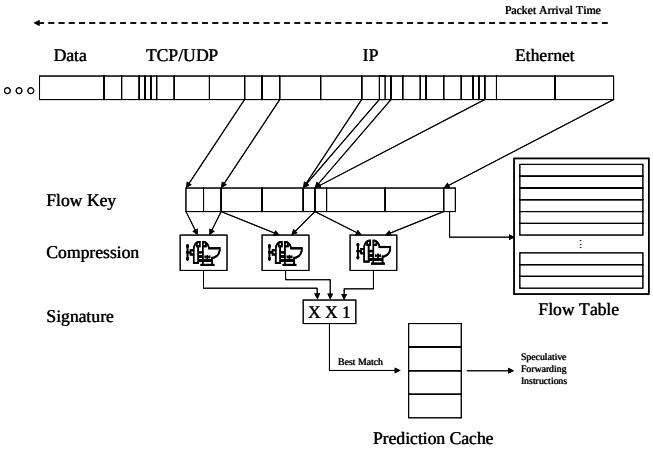


**Figure 4. Packet Prediction Signature Creation**

In a switch with packet prediction a compressed signature of the packet is computed as the packet arrives at the input memory. This signature is searched in a local per-port prediction cache that contains the signatures of previously received packets. If a match is found, the packet is assumed to be part of the same flow as the previous packet from the matching signature and the same operations can be speculatively applied. The prediction step and speculative operations occur in parallel with the traditional lookup stage. Figure 4 shows the process of signature creation relative to packet reception and flow key lookup.

Figure 5 shows a diagram of switch architecture that includes packet prediction logic. Since the goal is to begin the fabric transfer as soon as possible with the highest probability that the packet is actually transmitted through the correct egress port, the predictor snoops on the input memory bus and generates the packet signature as the packet is streamed into memory. This logic is required on each input port of the switch, so it is worthwhile to find the smallest and most efficient implementation possible. There are trade-offs between the size of the signature, the size of the cache and the method and amount of time taken to generate the signature.

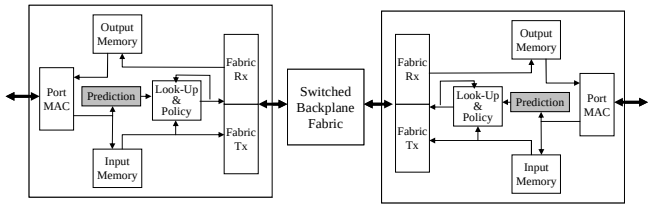


**Figure 5. Switch Architecture with Packet Prediction**

Switch latency for a packet predicting speculative switch is limited by the time it takes to generate enough of the packet signature to confirm a match in the prediction cache. Let $S = \{S_1, S_2, \ldots S_m\}$ be a packet signature of length m that consists of a set of bits that have been derived from the fields in H. There are a set of functions F that derive bits $S_i$ through $S_j$ of S from the fields in H as they arrive from the link. Let $K_p$ be the number of bits received to form enough of S to find a match in the prediction cache. Then the latency for a packet predicting speculative switch is as follows:

Packet Prediction with Speculation Latency = $(K_p / R_p) + 1/R_f$ (4)

This formula represents the time taken to receive enough of the packet to construct enough of the packet signature to find a match in the prediction cache, plus the time to transfer the first bit of the packet across the fabric. This results in the pipeline diagram shown in Figure 6.

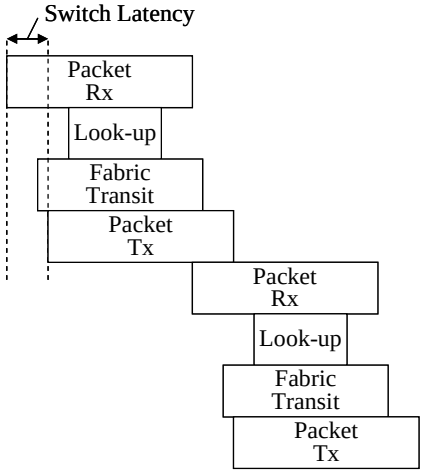


**Figure 6. Packet Prediction Switch Pipeline**

As with any pipeline employing speculation, there are several complications and clean-up steps that may be required. The results of the lookup stage confirm or deny the correctness of the speculation. Since it is possible for the first bit of the packet to be transmitted on the egress port before the results of the lookup are complete, the current egress transmission may be incorrect. Such an incorrect speculation requires that the packet transmission be aborted in some cases. To accomplish this, the packet must be corrupted by the transmitter before the last bit is sent. (However, this may not always be necessary - in the case of a layer 2 bridged network, a packet that is forwarded through the wrong port would simply appear as an extraneous flood and typically not cause an error.)

If the lookup process takes longer than the time allotted to receive a minimum sized packet, it is possible that the transmission of the packet on the egress port completes before the results of the lookup are determined. Since this model assumes a fixed time for the lookup stage and that some number of bits must be received to construct a signature, this condition cannot occur. In practical implementations, however, it is possible that such delays could exist and as a consequence packet transmission must be carefully scheduled.

In the cases where prediction and speculation are correct, the latency of the packet is significantly reduced. Results presented in section 5 show that reductions of over 85% are achievable, depending on the type of network traffic, the cache replacement algorithm, and the hardware configuration used.

# 4. SIMULATION ENVIRONMENT

In order to evaluate the effectiveness of the PPSCS approach, a C program was written that consumes actual traces of network traffic and simulates the behavior of the proposed architecture. The traces may be either live traffic, mirrored from a live switch

port, or they may be recorded trace files in libpcap format [8]. The simulator classifies the input network traffic as flow entries, which are stored in the flow table of the routing switch modeled above. Prediction algorithms are applied to the received traffic to create a packet signature of size 8, 16, 24 or 32 bits. This signature is compared to previously received packet signatures stored in a prediction cache, which may vary in size from 2 to 64 entries. If the signature is found in the cache, the incoming packet is assumed to be part of the same flow as the matching signature, and is forwarded with the same operations applied to it that had been applied to the previous packets in that flow. Forwarding begins at the time the signature match is found.

If the signature is not found, a new entry is put into the cache, replacing the least frequently used entry, and the packet is handled in the normal non-speculative manner (i.e. it is forwarded once the lookup is complete.)

## 4.1 Prediction Methods

There are numerous ways to construct the packet signature. The current simulator supports 6 different methods of generating and matching packet signatures. These methods are:

### *4.1.1 Fixed*

The Fixed method extracts bits from pre-defined locations in the packet as it is arriving. The offset locations have been chosen based on experience and an understanding of the important packet fields in an untagged Ethernet frame carrying a UDP or TCP message. Since the bit offsets are predetermined and fixed, there is no logic that parses the packet and adjusts the offset according to the frame encapsulation. As a consequence, bit offsets that would normally align with the TCP port fields will be unaligned if the packet is VLAN tagged, and may point to user data if the packet is an IP fragment. For a practical implementation of this method a different set of offsets should be considered based upon the port configuration.

The bits that are chosen are ones that are expected to vary the most between distinct flows. This includes the group address bit in the destination MAC address, low order address bits in both the MAC and IP headers, bits from the IP protocol field, and the TCP/UDP port numbers. The set of bit offsets selected for the signatures (where the first bit of the packet is noted as offset 0) are listed in Table 1.

**Table 1. Fixed Bit Offsets for Packet Signatures**

| Signature | Bit Offset |
|---|---|
| 8-bit | 7, 47, 94, 95, 238, 239, 270, 271 |
| 16-bit | 7, 46, 47, 94, 95, 100, 187, 190, 238, 239, 270, 271, 286, 287, 302, 303 |
| 24-bit | 7, 46, 47, 92, 93, 94, 95, 96, 100, 109, 110, 187, 189, 190, 237, 238, 239, 269, 270, 271, 286, 287, 302, 303 |
| 32-bit | 7, 45, 46, 47, 92, 93, 94, 95, 96, 100, 109, 110, 187, 189, 190, 191, 236, 237, 238, 239, 268, 269, 270, 271, 284, 285, 286, 287, 300, 301, 302, 303 |

The Fixed method must wait for the last bit offset to arrive before constructing the packet signature. Once the signature is assembled, it is compared to the signatures in the prediction cache. The 8-bit signature has the advantage of not having to wait as long as the other signatures, but has the obvious disadvantage of attempting to represent a packet with very few distinct bits.

### *4.1.2 Eager*

The Eager method uses the exact same bit offsets as the Fixed method to construct the signature, but builds partial signatures as the bits arrive. The partial signatures are presented as a key to the fully associative prediction cache, where missing bits are marked as don't care conditions for the match. If no matching entries are found, there are clearly no previous elements from this flow in the cache and the packet must wait for the flow lookup to complete and be forwarded normally. If there is precisely one entry found, then there is a chance that this entry is an exact match and the speculative forwarding of the packet may start immediately. This method forwards the packet as soon as possible, but experiences a higher misprediction rate. Receiving more bits for the signature can reduce the chance of a false positive match, but the probability of a misprediction can not be completely eliminated.

The cache replacement algorithm needs special consideration for the Eager method since there is a greater chance that a false positive will occur from a partial signature match. If a false positive occurs, the cache must be queried again with the full signature in order to replace the incorrect entry.

### *4.1.3 Hash*

The Hash method waits for the first 304 bits of the packet to be received and then constructs a 29-byte flow buffer from the offsets into the 9 fields of the packet that constitutes a flow. This method does not interpret the bits of the packet but rather extracts the predetermined offsets for these fields from what is presumed to be an untagged Ethernet frame encapsulating a TCP or UDP message. As with the Fixed method, if the packet is not a TCP/IP packet, if it is VLAN tagged or if it is an IP fragment, the offsets will not align with the desired fields. The flow buffer is constructed with whatever bits are located at the predetermined offsets.

A simple hash function (one of many developed by Professor Daniel J. Bernstein of the University of Illinois) is then applied to the 29-byte flow buffer to create the signature of the desired size. (A more extensive list of hash functions is available at http://www.partow.net/programming/hashfunctions/index.html.) The prediction cache is then searched using this signature. Similar to the Fixed method, the packet may not be forwarded until at least the first 304 bits have been received.

### *4.1.4 Smart Hash*

The Smart Hash method is similar to the Hash method, except that logic is applied to parse the packet and properly create the 29 byte flow buffer. The logic is capable of decoding the exact Ethernet header used and whether the frame is a TCP/UDP message, IP fragment or some other type of layer 2 protocol. Fields of the 29 byte flow buffer that are not present in the packet are filled with zeros. If the packet is an IP fragment, then the IP fragment ID field is used instead of the TCP/UDP port numbers.

The goal of this method is to trade off more logic in the packet prediction implementation for a more accurate packet signature to reduce the number of false positive matches. Similar to the Fixed and Hash methods, the packet may not be forwarded until the first 304 bits have been received.

### *4.1.5 Eager Hash*

The Eager Hash method is also similar to the Hash method, with the difference being that the signature is assembled from separate hashes of distinct portions of the 29 byte flow buffer. This method waits for the offsets of distinct chunks of the packet to arrive, such as the Ethernet header, IP addresses or TCP port numbers, and calculates a hash based only on those chunks to perform partial construction of the signature for eager matching in the cache. Once a partial signature has been created from the hashes, it is presented to the fully associative prediction cache with missing portions of the signature marked as don't cares. As with the Eager method, if there is zero or exactly one match, the search is terminated. The goal of this method is to forward the packet as soon as possible, but also reduce the number of false positives that might exist in the Eager method. The cache replacement algorithm used is the same as in the Eager method.

### *4.1.6 Smart Eager Hash*

The Smart Eager Hash method combines the informed construction of the flow buffer used in the Smart Hash method with the early speculative forwarding of the Eager Hash method. The same Eager method replacement algorithm is used.

This paper merely scratches the surface of the many prediction methods possible. Small amounts of additional logic could enable application specific prediction algorithms, for example, or history traces of protocol activity could enhance the accuracy and further exploit the temporal locality in network traffic. We intend to more fully explore this design space in the future.

## 4.2 Actual Trace Datasets

In order to evaluate the various prediction methods, simulations were run using a fixed set of traces. These traces represent different network environments and different parts of the network topology. The temporal locality of the received traffic differs with the network environment and location within the topology as will be shown in section 4.3. Network ports that are closer to individual stations have lower numbers of multiplexed flows, for instance, and network ports that are in the core of the network or at the Internet edge are likely to have greater numbers of multiplexed flows. We anticipate PPSCS will be most effective in the data center, near clusters of message passing servers, where the total number of flows is expected to be relatively small and low latency cut-through switching will be most beneficial.

The following 3 network trace datasets were used in the simulations:

### *4.2.1 LBL*

Lawrence Berkeley National Laboratory (LBNL) maintains 11 GB of anonymized packet header traces from October 2004 through January 2005, which are available for download from http://www.icir.org/enterprise-tracing/download.html. The traces are of LBNL enterprise campus LAN traffic from subnet links connected directly to the site router. A thorough analysis of these traces is available in [9]. We used a single 148 MB file from the LBNL datasets with 2.2M packets and 15K flows. The average packet size in the trace is 344 bytes and contains 98% IP traffic, of which 96% is TCP. (Other files from the dataset were also simulated for consistency checking, but their results are not reported here.)

### *4.2.2 R3L*

The R3L trace file was captured from the LAN backbone of a network engineering department in May 2008. The trace contains only inbound traffic to a core switch with a backbone 10GbE port connecting the engineering development servers. Therefore the servers are the source addresses of the packets. The outbound traffic is not included in the trace, which more accurately represents the type of traffic the prediction logic would experience in a real implementation.

The trace file has 490K packets from 32K flows with an average size of 198 bytes. IP traffic represents 99% of the trace, of which 55% is UDP and 44% is TCP. The remaining 1% of traffic is divided between various layer-2 protocols and ARP.

### *4.2.3 Edge*

The Edge trace file was captured from a link to a workgroup switch in the same network engineering department as R3L in May 2008. The trace captures only the inbound activity of a small number of engineering users, and therefore client stations are predominantly the source addresses of the packets.

The trace file has 250K packets from 5K flows with an average size of 151 bytes. 18% are ARP packets, 2% other layer 2 packets and 80% IP packets (of which 40% are UDP, 38% are TCP and the remaining 2% are ICMP and IGMP).

## 4.3 Temporal Locality of Network Traces

The effectiveness of the packet prediction scheme depends upon the temporal locality of network traffic. Figure 7 shows that the network traffic in the trace datasets exhibit substantial locality - more than 50% of the packets from all traces arrive with a spacing of 4 packets or less from a previous packet in the same flow. The figure shows the distribution of packet spacing up to a gap of 10 packets, which covers approximately 75% of all packets in the traces. The remaining ~25% of the packets lie in the long tail of the distribution. The measured distribution of the packet flow gap in the trace datasets closely matches the results observed in [15].

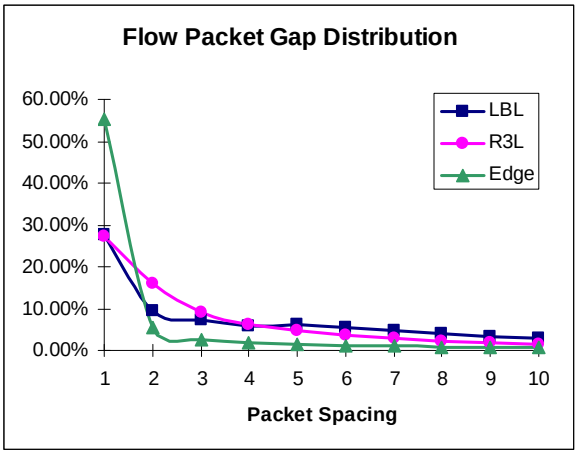


**Figure 7. Temporal Locality of Trace Datasets**

# 5. SIMULATION RESULTS

A simulation run was made for each combination of cache size (2, 4, 8, 16, 32 and 64 entries), packet signature size (8, 16, 24 and 32 bits) and trace dataset (LBL, R3L and Edge). The formulas for

calculating switch latency, equations (1), (3), and (4), were used to compute the various latencies.

Intuitively, one would expect that the largest signature size and largest cache size would be the most effective. However, the objective of packet prediction is to begin forwarding the packet as soon as possible with the highest probability that the speculation is correct. This would imply that eager methods with small but accurate signatures would fare the best. Figure 8 shows the reduction in switch latency on the R3L trace for different packet prediction schemes, compared to a conventional store-and-forward and cut-through switch. The signature size is set to 32 bits.

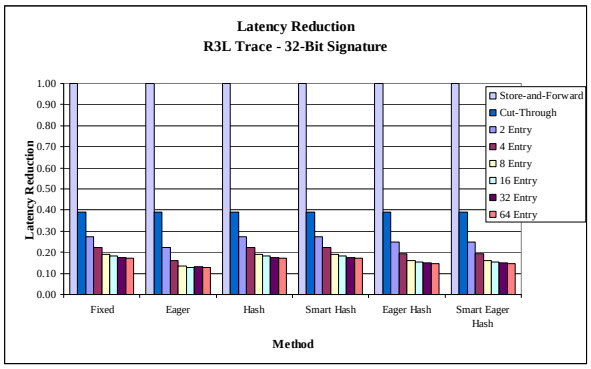


**Figure 8. Latency Reduction for R3L with 32-Bit Signature**

The figure shows the Eager method performs the best with a 64-entry cache - the switch latency for this configuration is 0.13 times the latency of a store-and-forward switch and 0.33 times the latency of a cut-through switch. This corresponds to nearly a factor of 8 and a factor of 3 reduction in latency, respectively.

Figure 9 shows the Eager method latency reduction for each of the trace datasets. It is interesting to note that as the cache size increases, the performance of the Eager method degrades for the LBL trace. This phenomenon occurs because a larger number of stale entries exist in the prediction cache, which only serve to further delay the exact match of partial signatures.

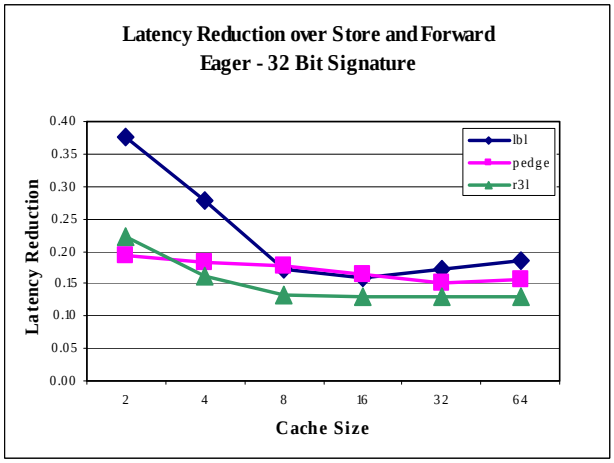


**Figure 9. Eager Method Latency Reduction for All Traces**

The apparent optimal cache size for the LBL trace is 16. Figure 10 shows the performance of the Eager method on the LBL trace for all cache sizes and signature sizes. As expected, 32-bit signatures provide the highest performance, but the difference is not as significant at larger cache sizes.

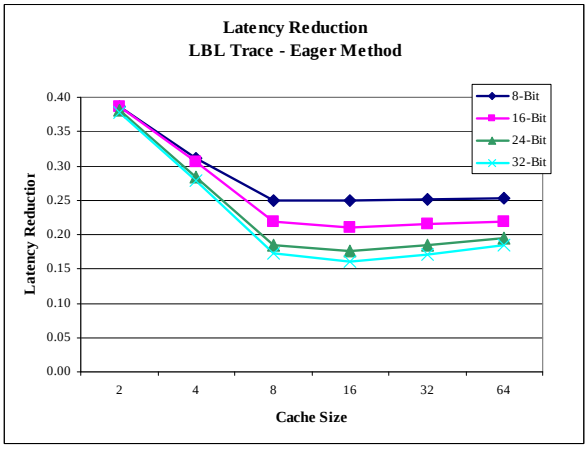


**Figure 10. Eager Method Latency Reduction on LBL Trace**

Figure 11 reveals that the non-eager methods all exhibit a continuously decreasing latency as the cache size grows, which is not the case for the eager methods. Non-eager methods always wait for a complete signature to be constructed before searching the cache, so their performance is essentially bounded. There will either be a matching signature, or there will not be. Eager methods, on the other hand, begin to search the cache as soon as enough bits are available to assemble a partial signature. In practice, when the same bits are used to generate a signature, any time a non-eager method mispredicts[3] an eager method will as well. However, an eager method will also generate a misprediction instead of a cache miss if it finds exactly 1 entry in the cache that matches its partial signature (a false match that would disappear over time as more bits get added to the partial signature). It is important to note that an eager method will never pick the wrong signature if the correct one is in the cache – the only mispredictions that will occur are when the actual signature is not in the cache, but a signature with a matching prefix is.

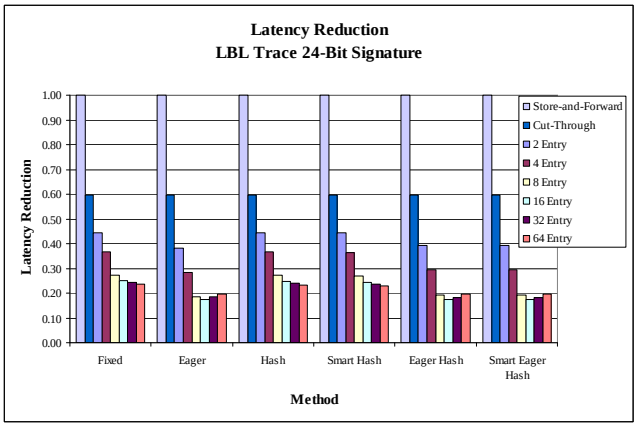


**Figure 11. Latency Reduction of LBL Trace**

[3] Remember, it is possible for two different flows to have the same signature. Thus, a misprediction occurs when a signature matches one in the cache, but further analysis shows that the actual flows are different.

Since we do not add any extra penalty for mispredictions, a non-eager method can never outperform an eager method that uses the same bits of the packet to generate a signature, even if the eager method mispredicts more frequently. In the end the two methods will both experience the same set of cache misses, and pay the full lookup latency penalty on those misses.

It is possible for an eager method with a lower correct prediction rate to outperform a non-eager method with a slightly better correct prediction rate, since the eager method begins the speculative forwarding of the packet sooner and the gain from more aggressive speculation outweighs the cost of the incorrect prediction. Figure 12 shows the correct and incorrect prediction rates for the different methods using a 24-bit signature on the LBL trace dataset.

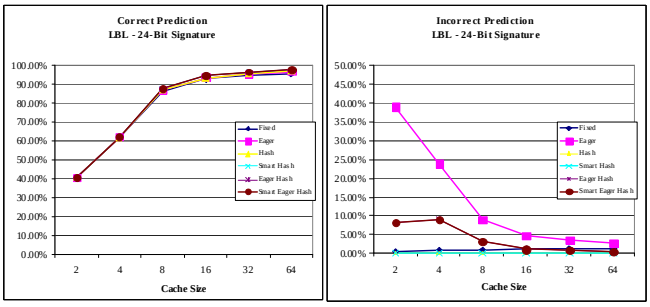


**Figure 12. Prediction Rates on LBL Trace**

The figure clearly shows that correct prediction rates nearing 97% are possible. It also shows that all eager methods have a high incorrect prediction rate when the cache size is small. This is understandable, since an eager method stops searching the prediction cache under two conditions - when there is exactly 1 entry that matches the partial signature, or when there are no entries that match. When the cache size is small it is more likely that a small partial signature will match with exactly 1 entry in the cache because there is little diversity. When the cache is large there is more diversity among the entries and more bits of the partial signature are required to narrow the search. This reduces the chance of a false positive match and thus the number of incorrect predictions.

The hash based methods rarely miss-predict, since they take into account a greater number of bits when creating a signature. The Fixed method, on the other hand, experiences a slight increase in the incorrect prediction rate as the cache size grows (indicating that choice of bits used to create the signature is not significant enough to uniquely identify the flows). The difference between these two types of methods is best seen in Figure 13.

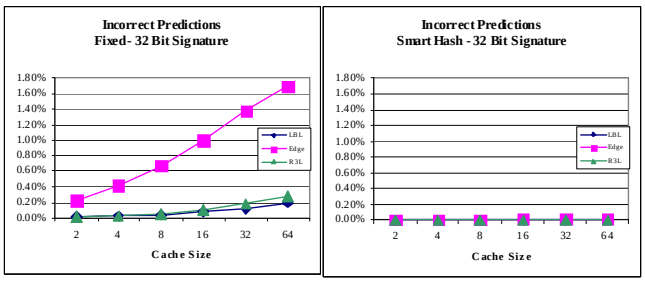


**Figure 13. Fixed verses Smart Hash Incorrect Predictions**

The Smart Hash method is much more effective at eliminating incorrect predictions on the Edge trace than the Fixed method. Recall that the Edge trace has the highest mix of non TCP/UDP traffic - the Edge trace dataset has 18% ARP packets and 2% other layer 2 frames, while the R3L and LBL trace files have 99% and 98% IP traffic, respectively. Since the Fixed method simply extracts bits from predetermined offsets, and those offsets are optimized for TCP/UDP traffic, it is no surprise that the Edge traffic has the highest number of false positive matches between the two. The figure shows the effectiveness of hashing over selecting pre-defined bits for all signature and cache sizes used with the Edge trace dataset.

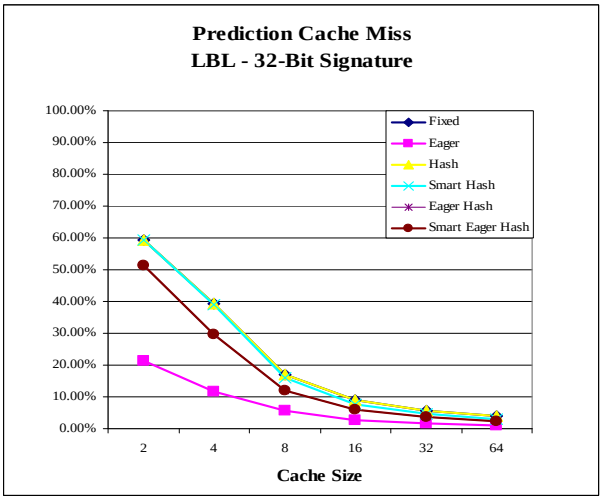


**Figure 14. Prediction Cache Miss Rates**

Prediction cache misses occur when a new flow is established or the signature for an existing flow has been removed from the cache. Figure 14 shows that the Eager method has a lower prediction cache miss rate with small caches (~20%) than the Hash method (~60%). This is because the Eager method has a significantly higher incorrect prediction rate with lower cache sizes. Incorrect predictions are not counted as cache misses - whether there is an incorrect prediction or a prediction cache miss, the same switch latency penalty is paid, so the more speculative approach tends to benefit in the overall latency calculations. The downside to the more speculative eager approaches is that they potentially waste backplane resources and power, which in practice is not free. A detailed evaluation of the impact of incorrect predictions is left for future study.

In summary, the results in this section show that correct flow prediction rates approaching 97% and a reduction in latency by a factor of nearly 8 are achievable. The hash methods have a slight performance advantage over fixed offset methods, at the expense of more logic on the input port. Additionally, the hash methods reduce the incorrect prediction rate, which saves power and backplane bandwidth. Eager methods have the lowest latency and are most effective with a medium sized cache. However, eager methods also put more pressure on backplane resources and potentially consume more power because of higher incorrect prediction rates.

# 6. RELATED WORK

Using prediction techniques to reduce latency and improve communication performance has not been extensively studied. No work that we are aware of has used compressed packet content

to facilitate prediction in parallel with the flow classification process on a statistically multiplexed packet switch.

The goal of reducing the number of pipeline stages in a routing switch by using predictive switching was proposed in [10]. The proposed technique looks at the forwarding history of an ingress port to predict the egress port, irrespective of the contents of the packet. The objective is to allocate connection resources across the backplane before the packet arrives to avoid the overhead of virtual channel establishment in a connection oriented switch. Suffix pattern matching is used on a history array, containing a list of the previous egress ports, to predict the next egress port. The forwarding of the packet does not occur until the results of the route calculation (i.e. lookup) are complete. The 2-D torus network for which this prediction scheme was developed is a connection oriented switch for specialized high performance computing applications. The authors achieved 77% prediction accuracy with the NAS parallel benchmarks.

Speculative techniques are proposed to reduce the latency of setting up paths across a connection oriented crossbar fabric in [11]. In this work, fabric bandwidth is arbitrated under the speculation that fabric virtual channel allocation will typically succeed. All of this occurs after the decode and routing stages (i.e. packet Rx and lookup stages). However, as the richness of the forwarding policy increases, the complexity and latency of decode and routing stages will begin to dominate.

Another connection oriented switch for high performance multiprocessor clusters uses prediction to improve the performance of TDM circuit allocation by freeing the oldest connections in a working set [12]. This proposal reduces latency by priming the working set of connections and taking hints from compiler directives. The approach differs significantly from a connectionless packet oriented switch that operates under statistical multiplexing methodology, and is targeted at the interconnect fabric of a tightly coupled multi-processor system.

A host oriented approach uses traffic speculation to improve the performance of IP fragment reassembly in [13]. The proposal modifies the driver interface to the message passing API of high performance computers to allow applications to send large memory pages over Ethernet. These large messages must be fragmented to transit an Ethernet network, so the proposal speculatively creates receive DMA chains that enable support for zero copy message passing.

The comparison of speculative switching ideas with concepts from advanced computer architecture is best described in [14]. Surendra et al show how the temporal locality of network traffic can be exploited to improve instruction reuse and reduce latency in network processors. Their proposal is to have a separate instruction reuse buffer for each active flow, and to switch the processor context when a packet is received from a different flow. The approach is conceptually similar to PPSCS in that the prediction cache holds a set of forwarding instructions for packets from a particular flow. The same temporal locality is used to select a context and speculatively operate on a packet. However, they assume the packet classification completes with 100% accuracy and there is no speculation. They are focused on speeding up the instructions that operate on the packet after the classification is done, where PPSCS uses a cache in parallel to speculate the results of the classification.

# 7. LIMITATIONS AND FUTURE WORK

The reduction in latency from PPSCS for a given configuration of cache size and signature size is directly related to the number of concurrent flows and amount of temporal locality in those flows. The scheme will require more resources if applied to aggregated links in the core of the network; however, these resources are in proportion to the amount of resources required for the general packet classification process.

Additional logic and fabric signaling is required to handle the clean-up from incorrect speculation. In practical situations this additional logic may be quite complex. For example, to avoid improper transmission of a packet, the egress scheduling must be synchronized with the completion of the lookup process. In the simple switch model used to introduce PPSCS, the time for the lookup process is deterministic. In many low cost implementations that deploy hashing or tree searching, the delay from the lookup process may be variable. It may be most desirable to simply delay the transmission of the frame until the lookup completes, which diminishes the latency gains from the speculative transfer.

There are many areas of future study for PPSCS. For example, performing real-time analysis of significant bits from which to form signatures could be highly beneficial. The Fixed method of signature creation is very simple and has acceptable performance in general cases, but is susceptible to variations in packet encapsulation and non-IP traffic mix. The performance of the Fixed method could be improved in real-time by a monitoring process that measures the entropy of significant bits for the current environment and downloads a new set of offsets to generate signatures.

There are also various methods of cache design and replacement algorithms that could be explored. Examples include measuring the effectiveness of adding a victim cache, or whether n-way associative caches with larger key sizes are more cost effective.

The current approach uses a minimum of packet contents to identify a flow. Given the temporal nature of network traffic, a history buffer could support true prediction before any bits have been received. This history could take into consideration the state of various control bits in previous packets, such as TCP FIN or SYN settings, to predict the status of an individual flow. The history record could also be used to build confidence in the current prediction using partial signatures. If the confidence is not high enough, the predictor could wait for additional bits of the signature to arrive.

A major area of consideration for future work involves the design of the clean-up phase after incorrect prediction, and an analysis of the potential performance impact in a multi-port configuration. It is possible that the actual timing and communication mechanisms necessary to support clean-up may be quite complex, although it is also possible that existing mechanisms can be leveraged to significantly lower cost.

A further area of consideration is an analysis of the security threats associated with incorrect predictions. An attacker that is aware of the prediction schemes may attempt to inject small messages through the prediction logic before time consuming security lookup operations complete.

## 8. CONCLUSION

This paper proposes a predictive technique to accelerate the forwarding operations of a modern Ethernet switch in order to reduce latency. All applications benefit from lower end-to-end latency, but many specialized applications in the high-performance computing arena depend upon very low latency. The need for a low latency, high bandwidth interconnect fabric has driven an industry of specialized connection oriented switches for many years. With the addition of packet prediction, a connectionless and statistically multiplexed Ethernet packet switch can help address the needs of high performance computing well beyond that of a traditional Ethernet switch. The results of simulations using real network data have shown that packet prediction can reduce the latency of a traditional store-and-forward switch by nearly a factor of 8 and reduce the already low latency of a cut-through switch by a factor of 3. Correct prediction rates approaching 97% are achievable with a moderate amount of per-port circuitry.

## 9. REFERENCES


[1] Liu, J., Chandrasekaran, B., Wu, J., Jiang, W., Kini, S., Yu, W., Buntinas, D., Wyckoff, P., and Panda, D. K. 2003. Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics. In Proceedings of the 2003 ACM/IEEE Conference on Supercomputing (Nov. 15 - 21, 2003). Conference on High Performance Networking and Computing. IEEE Computer Society, Washington, DC, 58.

[2] Hamid, N. and Coddington, P. 2007. Averages, distributions and scalability of MPI communication times for Ethernet and Myrinet networks. Proceedings of the 25th IASTED International Multi-Conference: parallel and distributed computing and networks (Innsbruck, Austria, 2007).

[3] Sokolowski, P. J. and Grosu, D. 2004. Performance Considerations for Network Switch Fabrics on Linux Clusters. Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing Systems (MIT Cambridge, USA, November 2004).

[4] Gupta, P. and McKeown, N. 2001. Algorithms for packet classification. Network, IEEE, 15, 2, (March 2001), 24-32.

[5] Taylor, D., E. 2005. Survey and taxonomy of packet classification techniques. ACM Comput. Surv., 37, 3 (Sept. 2005), 238-275.

[6] Broder, A. and Mitzenmacher, M. 2004. Network applications of Bloom filters: a survey. Internet Math., 1, 4, (May 2004), 485-509.

[7] Dharmapurikar, S., Krishnamurthy, P., and Taylor, D. E. 2003. Longest prefix matching using bloom filters. In Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications (Karlsruhe, Germany, August 25 - 29, 2003). SIGCOMM '03. ACM, New York, NY, 201-212. DOI= http://doi.acm.org/10.1145/863955.863979.

[8] TCPDUMP/LIBPCAP public repository. http://www.tcpdump.org/

[9] Pang, R., Allman, M., Bennett, M., Lee, J., Paxson, V. and Tierney, B. 2005. A first look at modern enterprise traffic. Proceedings of the Internet Measurement Conference 2005 on Internet Measurement Conference (Berkeley, CA, 2005). USENIX Association.

[10] Yoshinaga, T., Kamakura, S. and Koibuchi, M. 2006. Predictive Switching in 2-D Torus Routers. Proceedings of the International Workshop on Innovative Architecture for Future Generation High Performance Processors and Systems (2006). IEEE Computer Society.

[11] Peh, L. and Dally, W. J. 2001. A Delay Model and Speculative Architecture for Pipelined Routers. In Proceedings of the 7th international Symposium on High-Performance Computer Architecture (January 20 - 24, 2001). HPCA. IEEE Computer Society, Washington, DC, 255.

[12] Ding, Z., Hoare, R., Jones, A., Li, D., Shao, S., Tung, S., Zheng, J. and Melhem, R. 2005. Switch Design to Enable Predictive Multiplexed Switching in Multiprocessor Networks. Proceedings of 19th IEEE International Symposium on Parallel and Distributed Processing (2005).

[13] Kurmann, C., ller, M. M., Rauch, F. and Stricker, T. M. 2000. Speculative Defragmentation - A Technique to Improve the Communication Software Efficiency for Gigabit Ethernet. Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (2000). IEEE Computer Society.

[14] Surendra, G., Banerjee, S., and Nandy, S. K. 2003. On the effectiveness of flow aggregation in improving instruction reuse in network processing applications. Int. J. Parallel Program. 31, 6 (Dec. 2003), 469-487.

[15] Jain, R. and Routhier, S. 1986. Packet Trains--Measurements and a New Model for Computer Network Traffic. IEEE J. Sel. Area Comm., 4, 6 (1986), 986-995.

[16] Leland, W. E., Taqqu, M. S., Willinger, W., and Wilson, D. V. 1994. On the self-similar nature of Ethernet traffic (extended version). IEEE/ACM Trans. Netw. 2, 1 (Feb. 1994), 1-15. DOI= http://dx.doi.org/10.1109/90.282603.

[17] Willinger, W., Paxson, V., and Taqqu, M. S. 1998. Self-similarity and heavy tails: structural modeling of network traffic. In A Practical Guide To Heavy Tails: Statistical Techniques and Applications, R. J. Adler, R. E. Feldman, and M. S. Taqqu, Eds. Birkhauser Boston, Cambridge, MA, (1998), 27-53.

[18] Mogul, J. C. 1992. Network locality at the scale of processes. ACM Trans. Comput. Syst. 10, 2 (May. 1992), 81-109

[19] Kermani, P. and Kleinrock L. 1979. Virtual Cut-Through: A New Computer Communication Switching Technique. Computer Networks 3, (Sept. 1979), 267-286.

[20] Abo-Taleb, A. and Mouftah, H. 1987, Delay Analysis Under a General Cut-Through Switching Technique in Computer Networks, IEEE T. Commun., 35, 3, (Mar. 1987), 356-359

[21] Feldmeier, D.C. 1988. Improving gateway performance with a routing-table cache. In Proc. IEEE INFOCOM '88. (New Orleans, March, 1988), 298-307

[22] Newman, P., Minshall, G., Lyon, T. and Huston, L. 1997. IP switching and gigabit routers. IEEE Commun. Mag., 35, 1 (Jan 1997), 64-69.

[23] Partridge, C. 1996. Locality and route caches. NSF Workshop on Internet Statistics Measurement and Analysis (http://www.caida.org/outreach/isma/9602/positions/partridge.html), 1996.