In-network Video Prioritization via iBox Classification Predicates

George Manning Porter
Randy H. Katz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2005-1
http://www.eecs.berkeley.edu/Pubs/TechRpts/2005/EECS-2005-1.html

September 29, 2005

In-network Video Prioritization via iBox Classification Predicates

George Porter Randy H. Katz*

29 Sep 2005

Abstract

We propose a novel datapath mechanism for tracking and acting on headers in a variety of layer-7 protocols called Classification Predicates, or "cPredicates". We apply this mechanism to the emerging field of in-network storage (Storage Area Networks, or SANs), and consider a multimedia streaming service with video stored in a converged SAN that also contains non-video content. We show that cPredicates have a low, amortized overhead because they only have to examine a small subset of the packet stream in depth. In our experimental environment, only 5% or fewer packets are examined in depth, leading to less than a 10% amortized latency increase. We built a content-based prioritization system for an iSCSI-based SAN and show that it can provide better than best-effort service for video files in a converged SAN containing both video and non-video content.

1 Introduction

The increasing demands on edge networks (including corporate, campus, and access networks) have driven a demand for new functionality from the network itself. A proliferation of so called "network appliances" have sprung up to meet this demand, including Checkpoint's VPN endpoint[3], Packeteer's PacketShaper[9], Nortel's Alteon HTTP load balancer[11], and many others. While the growth of in-network packet processing has enabled new network services, it has complicated the task of network management. Given a myriad of devices from separate vendors, network administrators must often cope with different configuration interfaces, unexpected device interactions, and increased power and rack space demands. This ad-hoc drive towards specialization and "stovepipe" solutions naturally leads to a desire for a more general-purpose platform for deploying edge services. We call this emerging class of devices *Inspection-and-Action boxes*, or iBoxes[5].

iBoxes, like network appliances, will exist in the edges of the network, near servers and clients, rather than in the core. iBoxes are enabled by the increasing availability of powerful new network processors, RISC cores, and raw computational power that can be put into switches and other network devices at minimal cost. An iBox resides in the data path, and for anticipated functions it might have hardware dedicated to TCP header processing (for applications such as NAT and firewalling). Likewise it has support for a set of actions such as **drop_packet**. Because the iBox is programmable, it is possible to extend its classifications and actions to some extent. It is clear that the on-board hardware has limits as to its reprogrammability, and so a "backdoor" to a more general-purpose processor will be available. This path enables rich classification and action processing on the packet, but its overhead will likely be expensive relative to the standard classification path. For this reason it is important to process as few packets as possible in depth. This observation drives the design of our cPredicates mechanism.

In this paper we address storage in the context of our envisioned iBox design, specifically for the support of video content delivery. Our goal is to support a variety of storage functions in the network itself to improve the performance and ease of management of

*Both authors: EECS Department, Univ of California, Berkeley, CA.

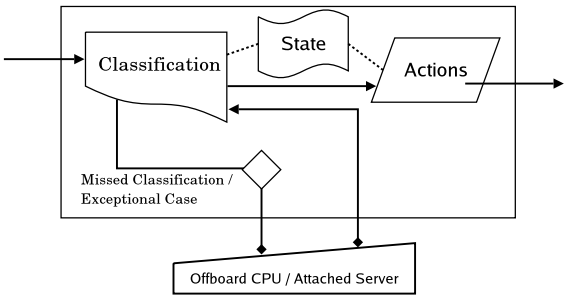

Figure 1: An iBox consists of some classification engine, state respository, and a set of actions. Extending this functionality further is enabled by a general processor that can handle exceptional cases and events that occur infrequently.

storage subsystems of multimedia services. Central to that is the ability to process storage commands efficiently in the iBox. Our design integrates into what we envision will be the structure of these devices. We consider as an application service the prioritization of video content in a converged SAN. By processing storage traffic at the application layer, we get a visibility into the semantic meaning of storage commands. Based on these commands, we can give those clients accessing video files higher priority than clients doing backups, or accessing mailboxes or web pages. This will make it easier to incorporate video and other files with strict access requirements with typical SAN content, rather than keeping a seperate SAN for multimedia content.

Section 2 motivates some requirements for cPredicates as well as the need to introduce storage functions for supporting multimedia storage systems. Section 3 presents our mechanism for doing this state management called cPredicates, as well as the design of our prioritization framework. Section 4 describes an evaluation of our design in a deployed multimedia service and compares it to a non-iBox based alternative. Section 5 discusses related work.

2 Video Delivery and Networked Storage

Because of its strict real-time requirements, we consider a multimedia delivery service as the target of our iBox application. Multimedia streaming environments are characterized by the movement of large amounts of data from storage to the video servers with large numbers of concurrent users accessing the video content.

2.1 TCP Dynamics, Stream Processing, and Protocol Layering

The iSCSI protocol exists entirely above the TCP layer, and consists of a sequence of data units, each with a 48-byte long header and variable length data. These data units can be very large, in some cases hundreds of kilobytes (certainly larger than an Ethernet frame), forcing these large iSCSI data units to span multiple TCP/IP packets. Also, given that the minimum length of an iSCSI data unit is 48 bytes, several iSCSI commands can fit into one 1500-byte Ethernet frame. These headers do not necessarily exist at any particular offset within the packet, and their location generally can only be determined using length and offset information from the previous data unit's header.

Observing and modifying an iSCSI stream in the network presents some unique challenges compared to other in-network packet processing tasks such as IP routing and network address translation. Unlike IP and TCP headers, which reside in each packet at an easily calculated offset, iSCSI headers might exist at an arbitrary offset into the packet. Alternatively, a packet might contain multiple headers, or no headers at all. Each of these cases has been observed in our testbed using commercially available iSCSI initiators and targets.

One way to handle this complexity would be to have the iBox become a TCP endpoint. In this situation, the iBox would be acting as a proxy. Stream processing in this case would be simple, because the TCP layer in the iBox would ensure that iSCSI data units are delivered in order and without loss. Unfor-

tunately, this option would entail examining the data payloads of all packets transiting the iBox, which would grealy limit its scalability. We assume that general processing of data portions of packets is only practical for a small percent of the packets transiting the iBox. This is especially true in iBox environments which contain programmable computational units (RISC cores, etc.) that can process some–but not all–of the data stream. Our solution is to take advantage of iSCSI's large data unit size to reduce the number of packets that we must process to keep track of the state of the iSCSI session. We observe that for streams consisting of large data units, we must examine only a small percent of the packets to track each of the iSCSI headers. Consider a sequence of 128-Kilobyte data units made up of 1500-byte packets. The first packet in the stream contains the iSCSI header of the first data unit, while the next 85 packets contain data alone (with the possible exception of the last packet in the stream, which might contain the beginning of the next iSCSI data unit). In this stream, only about 1.1% of the packets need to be examined to process the stream. This amortization is especially true of multimedia workloads, which tend to consist of large data transfers. For this reason our mechanism is especially suited towards the transfer of video and audio files. The relation between file size and number of packets examined in depth (i.e., of the data payload of TCP segments) is given in Figure 2.

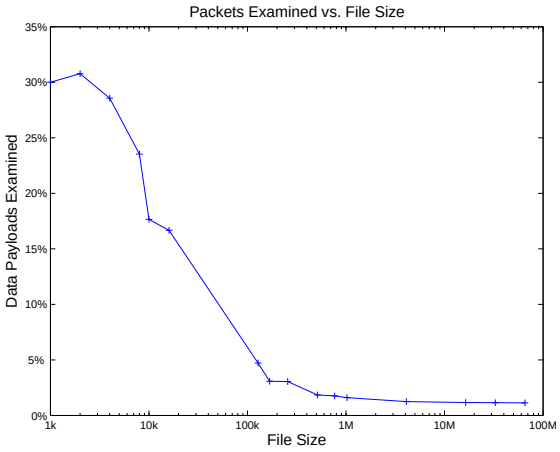

Figure 2: Data payloads examined vs. total packets in an iSCSI transfer of various files of different sizes. The files resided on the Linux *ext2* filessystem and the Intel iSCSI drivers were used on both the initiator and target.

2.2 Multimedia Delivery Service

Consider a multimedia service consisting of a tier of N video servers (see Figure 2). These servers source their video content (i.e., $MPEG4$ files [15]) from a storage area network (SAN) through a network switch. This design is desirable because each of the video servers has easy access to the devices that comprise the SAN, and so any client can utilize any of the servers to access content. Furthermore, offline processes and users can access the SAN directly and periodically add new video content without disrupting the video servers.

Consider such a video-based SAN providing content to a large news website such as CNN.com. For the highest performance one might design such a sys-

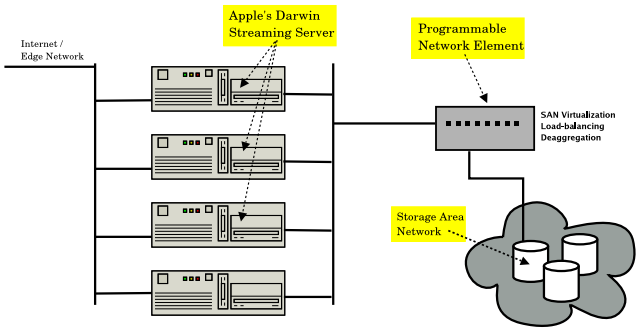

Figure 3: A multimedia service consisting of a tier of video servers sourcing content from an attached SAN through a switch. Our deployment replaces this switch with a Click-based cPredicate element

tem with a dedicated SAN for video content, and another SAN for HTML, images, databases, and other non-realtime content. For economic as well as ease-of-management reasons, however, it makes more sense to converge these two SANs into a single storage system. However, such a system will need to ensure that despite low-priority crosstraffic the high-priority video content is able to reach the video servers before their deadline expires so that end clients will get good performance. We consider this application as a target of our design.

In our multimedia service, we deploy our iBox in place of the switch connecting the video server network and the storage network. There, the iBox will be able to observe, intercept, and modify SAN transactions to and from the video servers. This is desirable since in an iBox the cPredicates module is able to see the crosstraffic patterns, and is able to take those into account when assigning priorities to packets (as well as higher-level units such as iSCSI commands and responses). The alternative–implementing this prioritization at the edge–is undesirable since edge devices are not able to observe crosstraffic patterns, and thus are not able to lower their sending rates when crosstraffic levels increase. We examine this process analytically in section 5.

2.3 Multimedia Testbed Deployment

To evaluate our iBox design and cPredicates mechanism, we have built a deployment in the Berkeley Oasis testbed. This testbed is a collection of twenty Pentium-III Linux server blades connected together through a configurable high-speed interconnect (Nortel Passport 8000 series routers). Through this interconnect we can configure many topologies and network conditions using the NISTnet network emulation software. We have implemented the cPredicates software in the Click Modular Router[8]. The testbed is configured into a video LAN connected to a SAN through our Click-based cPredicates element.

We chose to use Apple's Darwin Streaming Server[12] due to its open-source nature and its ability to stream *MPEG-4* and *Quicktime* files. The Darwin server sources video content from Intel's iSCSI driver suite[7]. Our overhead measurements are generated using the workload presented by the Darwin streaming service running on Linux, while prioritization throughput is provided by accessing ranges of blocks from a custom-written C application. Some of those blocks are defined to be high-prioirty, while the others are considered low priority.

2.4 Server/iBox integration

One intriguing environment to utilize iBoxes is in so-called "Blade Servers" such as IBM's BladeCenter[4], Sun's SunFire[10], and HP's Proliant BL[16]. These platforms tightly integrate multiple servers with a fast switching interconnect to simplify server management and application deployment. With the increased ability to add computational power to the switches that reside in Blade Server platforms, more intelligent server load balancing, storage functions, and stream processing capabilities can be deployed in tight integration with application programs residing on the blade server. In this case, server blades in the blade server would host the multimedia streaming software and the content itself would reside in network attached storage accessible through the fast interconnect provided by the blade server. In this case, the cPredicates mechanism could easily be extended to support the processing of both storage traffic as well as HTTP-based web traffic.

3 Enabling Video Storage with iBoxes

3.1 cPredicates

The heart of our stream processing engine is the cPredicates mechanism. This multi-protocol mechanism tracks the boundaries of application-layer application data units (ADUs) and forwards packets containing application-layer headers (such as iSCSI and HTTP) to a processing element that performs the per-application function. By examining only those packets that contain application-layer headers, and sending all other packets along a low-latency fast path, it will be able to run at high rates in iBoxes

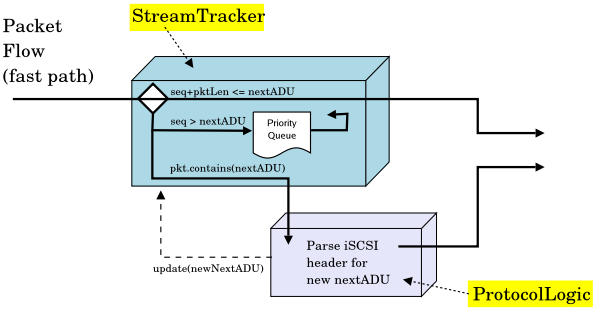

Figure 4: cPredicates elements utilizing fixed-header protocol logic block. Only packets that contain the header of the next ADU are sent to the **ProtocolLogic** block. All others are sent along the fast path directly through the cPredicates module.

containing programmable computational engines because those processing engines would only have to process a small fraction of packets transiting the iBox. This mechanism is designed to process application protocols that exist on top of a reliable transport layer such as TCP. Its efficiency is obtained from the fact that only those packets with application-layer headers in them are examined in depth. All other packets can transit a fast path. cPredicates are not suitable for application protocols in which a majority of the packets must be examined in depth, and so another mechanism–likely an application proxy–must be used to process these types of protocols.

3.1.1 Mechanism

Figure 4 shows the cPredicates element in the datapath of an iBox. We envision that each line card in the router or switch could contain this element as part of it's packet classification step. The cPredicates element is responsible for keeping TCP (Layer-4) state for each connection. Attached to the cPredicates is a protcol-specific header processing logic element called ProtocolLogic. This element is responsible for understanding the structure of a particular application-layer protocol. Different application-layer protocols can be supported in the iBox by connecting different ProtocolLogic elements to the cPredicate. By

Figure 5: cPredicates and ProtocolLogic Interfaces

```
class cPredicates {
   notifyNextHeader(FID flowId,
                    TcpSeq nxtSeq);
}

class ProtocolLogic {
   ProtocolLogic();
   process(Packet p,
           int offsetIntoPacket);
}

class FixedLogic extends ProtocolLogic {
   FixedLogic(int headerLength);
   setDataLengthField(int offset,
                      int length);
}

class iSCSILogic extends FixedLogic {
   iSCSILogic() {
      // The iSCSI header is
      // 48 bytes long
      super(48);

      // The data length field
      // starts at the 5th
      // byte in the header
      // and is 3 bytes long
      setDataLengthField(5, 3);
   }
}
```

Figure 6: Per-Flow State in cPredicates

```
struct flowState {
   uint32t expSeqNumber;

   // q_ is only nonnull during
   // loss and reorder events
   PriorityQueue<Packet *> * q_;
}
```

separating the packet processing and TCP header management (cPredicate) from the application-level protocol procesing (ProtocolLogic) by a clean interface, we enable more configurable, multi-protocol stream processing applications. The interface between these two elements is given in Table 5. In our deployment, the two elements communicate via inter-element Handler connections. In a network-processor implementation, this communication could use shared memory or busses.

When the first packet from a flow arrives in the cPredicate, it is sent to the ProtocolLogic element (in our case, the iSCSILogic element) for header processing. The iSCSILogic element extracts the data length field from the packet and signals that value to the cPredicate. cPredicates then update the next expected header location by adding the length of the ADU to the sequence number of its header. If the priority queue is nonempty, then any packets belonging to this flow are removed and sent on their way out of the cPredicates element. Subsequent packets that arrive while the header is being processed by the iSCSILogic element are stored in the priority queue located in the cPredicates module. Once the header is processed, any buffered packets belonging to that ADU are released to the switching fabric of the iBox.

Once the iSCSILogic element determines the length of the next iSCSI command or data transfer, it notifies the cPredicate element of that length. The cPredicate element is able to use that updated length to calculate the expected sequence number of the next application-layer header. Packets arriving to the cPredicate element can easily and quickly be checked to see if they belong to the current application data unit by comparing the sum of their length and sequence number to the expected sequence number. This process is outlined in Figure 4. Additionally, other elements in the router can register with the ProtocolLogic element to receive information about the layer-7 header. In the next section, this will be used to assign priorities to flows based on which data blocks they access. The state requirement of cPredicate module is given in Table 6. The ProtocolLogic element is stateless, and once it processes the header of an ADU, it sends it directly out to the rest of the switching fabric.

The cPredicates's priority queue is used when TCP segments containing the next application-layer header are lost. In such a case, received TCP segments are queued until the header is retransmitted. Note that no buffering is done when TCP segments known to contain data alone are lost. For example, assume that the sequence number of the header of the next iSCSI ADU is S_{nxt}. If TCP segments whose sequence space is less than S_{nxt} are lost, reordered, or corrupted, no special action is taken. They are delivered to the endpoint unmodified. Only in the case where a packet has a starting sequence number greater than or equal to S_{nxt} is it queued. This is because cPredicates have no way of knowing to which ADU it belongs, since it must process the header located at sequence number S_{nxt} before it can determine the status of any segments located later in the sequence space. For this reason, in the steady state, we expect that the $q\_$ pointer will be null. Only in cases of reorder or loss does this queue become used.

3.2 Video Prioritization

To support the prioritization of video content in a converged SAN, we embed the cPredicates mechanism into the configuration shown in Figure 7. As before, the iSCSI Protocol Logic block notifies the cPredicate element of the location of the next header in the protocol stream. In our application though, the Protocol Logic block also notifies a Priority Manager element. This element contains a mapping between disk block ranges and priority levels. The details of how this element is configured is given in Section 3.2.1. In general the Priority Manager has a number

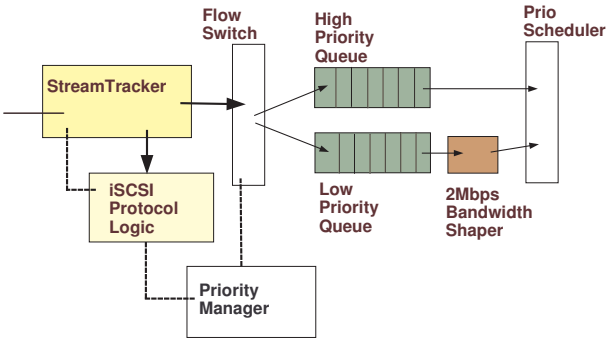

Figure 7: The iSCSI Protocol Logic block notifies the priority manager of block requests. This priority manager in turn directs the requesting flow into either a high priority queue or a bandwidth-shaped low priority queue.

of actions that it can perform on the flows transiting the iBox given that it has visibility into the semantics of the protocol (as furnished from the Protocol Logic block). For example, it could set Differentiated Services[1] bits in the packet, which would allow for policy enforcement elsewhere in the network. In our case, we choose the simple in-iBox enforcement scheme of bandwidth shaping the low-priority traffic.

When the Priority Manager is notified of an access to a high priority disk block, it configures the Flow Switch with the layer-4 information about that flow (Source IP address, Destination IP address, Source TCP port, and Destination TCP port). The flow switch sends packets from that layer-4 flow to a high priority queue. The Priority Scheduler looks for packets to send to the routing fabric from the high-priority queue first. If that queue is empty, it will then look in the low-priority queue. When the Priority Manager is notified that a flow is accessing a data block that is not considered high priority, it configures the flow switch to direct that flow to the low-priority queue. In our application, this queue is bandwidth-shaped to a rate of 2 MBits/sec.

3.2.1 Configuration and Management

In our testbed deployment, we statically configure the Priority Manager with a set of disk blocks representing video files. All other disk blocks (which represent data files, email, etc.) are considered low-priority. In a real deployment, the Priority Manager would have to be configured with the locations of video content in the SAN. This could be done by a tool that searches the SAN looking for known video formats. The tool would install entries in the Priority Manager containing the locations of video files that it finds. If the Priority Manager were to fall out of synchronization with the SAN, then blocks would still be served correctly, however some blocks might receive either a higher priority, or a lower priority, then they should.

Consider a SAN storing HTML, images, video, and other content required for a large news portal such as CNN.com. Since a tier of front-end servers need to serve HTML content and static images, as well as full-motion video, the prioritization framework presented above could be used to ensure that video traffic reaches video servers with high priority. Alternatively, there could be two high-priority queues: one for paying customers' video, and one for nonpaying customers's video. The system is flexible both in what characteristics of the layer-7 protocol stream can be used to determine priority, as well as mechanism of enforcment.

3.3 Additional Considerations

An immediate concern exposed by introducing processing into the network is the interaction between network appliances (both fixed-function and programmable) and data encryption. The presence of IPsec[6] and SSL make packet inspection and modification impossible. We explicitly assume that data connections transiting the iBox are unencrypted. This could be a policy of the edge network, or it could be the case that SSL and IPsec tunnels are terminated before they reach the iBox. For example, in a BladeServer environment, SSL offloading might be utilized to remove the data encryption as the connection reaches the BladeServer. Among the blades, data would remain unencrypted, which would

additionally allow in-network functions such as server load balancing and XML offloading. When responses are sent back to the client, they are reencrypted in a manner transparent to the client. An example of this type of environment is the Alteon HTTP load balancer[11].

4 Evaluation

To evaluate our design, we have deployed cPredicates in a Click[8] software-based router in our testbed. This router intercepts traffic between the SAN network and a video server running the Apple Darwin Streaming Server. Our SAN consists of a 100-Mbyte ramdisk exported to our video server with Intel's iSCSI driver suite. Traffic to the SAN is generated from the three workloads described in Table *I*:

Clip Name	Bitrate	Length	File Size
Mpeg1	357 Kbits/sec	125 s	5.5 Mb
Mpeg2	803 Kbits/sec	185 s	18 Mb
Mpeg3	644 Kbits/sec	347 s	27 Mb

Table 1: Workloads utilized in our evaluation. All clips are encoded at 24 frames/sec and at a resoluation of 360x240.

4.1 High-vs-low Priority SAN Content

Figure 8 shows the result of our prioritization framework. Specifically, it shows the instantaneous throughput as seen by a single iSCSI client accessing a set of files from the SAN. Some of those files are marked as high-priority, and the rest are low-priority. As the figure shows, the throughput experienced accessing the high-priority files is approximately 7 Mbits/sec., which is the maximum throughput that our experimental testbed can support. For those files not marked as high-priority, the throughput seen is approximately 2 Mbits/sec. The throughput seen by this client is determined by the set of blocks that it accesses. If the client were to access only video files, its throughput would be close to

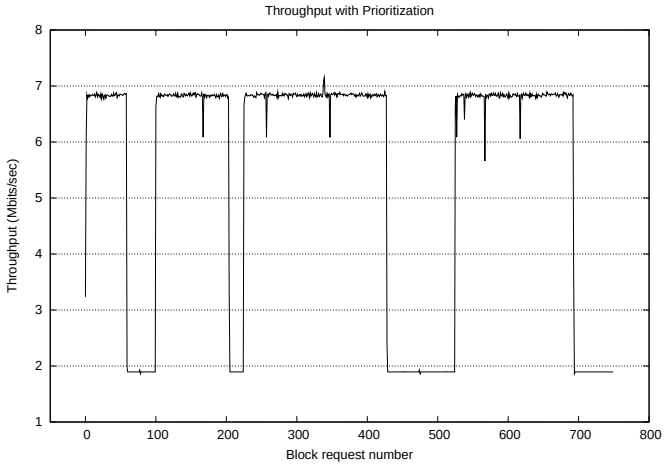

Figure 8: Instantaneous throughput seen by a single client accessing a series of high- and low-priority files. The low points of the square wave represent bandwidth-shaped low-priority file accesses, while the high points high-prioirty block accesses.

7 Mbits/sec. for the duration of connection. Likewise, if it accessed only email or other documents, its throughput would remain at 2 Mbits/sec.

We now consider the overhead in tracking an application-layer protocol using the cPredicates mechanism.

4.2 Overhead of the cPredicates Mechanism

As Table 4.2 shows, the results of our experiment are encouraging. In the three multimedia workloads considered, approximately 95% of the packets transited the iBox through the fast path. For these packets, the iBox had to perform a hash lookup to determine the sequence number of the next ADU. It then had to perform an addition (of the packet's sequence number and its payload length) followed by a comparison. The packet is then sent through the normal switch forwarding path. The other 5% of the packets had to have their data payloads examined and their iSCSI headers extracted and processed.

In our Click implementation, processing a non-

	MPEG1		MPEG2		MPEG3	
	Avg. Latency	Overhead	Avg. Latency	Overhead	Avg. Latency	Overhead
Stock Click	36.32 μs	0.00 %	37.67 μs	0.00 %	38.02 μs	0.00 %
cPredicates (Data)	37.67 μs	3.72 %	38.99 μs	3.50 %	39.28 μs	3.31 %
cPredicates (Header)	73.23 μs	201.62 %	75.10 μs	199.36 %	75.42 μs	198.37 %
Percent Headers	5.23 %		4.56 %		4.58 %	
Amortized Cost	39.44 μs	8.59 %	40.65 μs	7.91 %	40.94 μs	7.68 %

Table 2: Performance of stock Click vs. cPredicates-enabled Click (as measured in CPU Cycles/sec) as well as number of data payloads examined (to perform header processing).

header packet takes approximately 3-4% longer (or about 1.5 μs) than standard Click packet forwarding. As expected, header processing takes much longer–in some cases over 200% longer. This high overhead is offset by the low percentage of headers in the observed protocol stream. As shown in Table 4.2, this leads to an amortized overhead of approximately 8%. In the case of iSCSI, this overhead is unlikely to lead to a significant reduction in throughput since iSCSI requests and responses can be pipelined, leading to a less interactive protocol, and thus less effected by latency.

4.3 Network-vs-edge Prioritization

An alternative to implementing a prioritization framework in the network would be to perform that function in the endpoints (specifically the clients accessing the SAN). To do that, the clients could be configured to bandwidth-shape their request traffic to 2 Mbits/sec when accessing databases, mailboxes, and other low-priority content. For high-priority content, they would not restrict the rate at which data is accessed in the SAN.

While this scheme would work well in the absense of crosstraffic, once other clients begin to join the SAN and access low-priority content, the performance of a high-priority flow would begin to suffer. This is because the endhosts do not have visibility into the conditions in the SAN–specifically the status of crosstraffic and congestion. An iBox configured with cPredicates could put all low-priority traffic into a single bandwidth-shaped queue, which would in effect cause all clients accessing low-priority traffic to share a "virtual" 2MBits/sec link. As the number of low-priority clients join the system, they would contend for their portion of that restricted bottleneck bandwidth, while the high-priority traffic would be able to use the rest of the bandwidth without being affected. In an endhost-based scheme, clients cannot tell which other clients are accessing high-priority traffic and which are accessing low-priority traffic, and so while the number of low-priority clients increases, each continues to send at 2Mbits/sec. This quickly causes the high-priority flow to share the link with the low-priority flows, and as Figure 9 shows, this leads to decreased performance as the number of clients increases.

In addition to prioritization, there are a variety of storage applications that benefit from existing in the network fabric itself, such as storage virtualization, redirection, and mirroring. In each of these cases, by placing the functionality in the network system administrators are able to centrally manage a storage subsytem. This leads to better efficiency and utilization as compared to endhost-based control of the SAN content. Although not convered in the work presented thus far, cPredicates could be extended to support the storage functions just mentioned.

5 Related Work

Several network-appliances process application-level protocol flows ([11],[9],[3]). One proposed mechanism for accelerating TCP proxies (encompassing HTTP

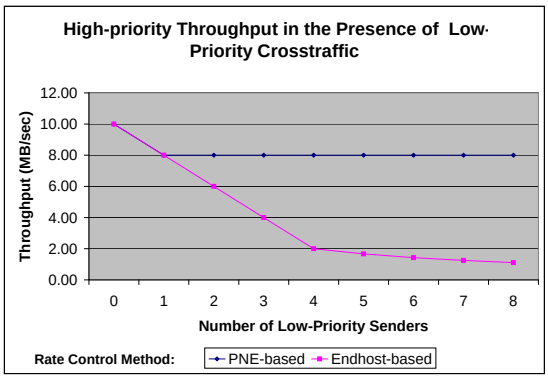

Figure 9: Analytical projection of the throughput of a single high-priority data transfer in the presense of increasing numbers of 2 Mbits/sec crosstraffic flows. In this example, we assume MAX-MIN fairness with the high-priority flow requesting infinite bandwidth. The network consists of 10Mbits/sec links with 10ms delay per segment.

load balancers, Telnet proxies, etc.) is called TCP Splicing[13, 14]. Our proposal extends this work by providing programmers with a well-defined interface between cPredicates and the application-specific ProtocolLogic elements. In this way, we have separated packet processing at the IP and TCP levels from application-layer protocol processing. This allows for multiple, protocol-specific modules to be introduced to extend the functionality of the iBox at runtime.

There have been several proposed software-based routers[8, 2]. Their focus has primarily been to enable rapid deployment of new router features. Our iBox model differs in that it provides a more restricted set of primitives designed to support a variety of edge services through a specification language encompassing commonly used packet classification and action steps.

6 Conclusion

We have proposed a mechanism for supporting application-layer protocol streams in programmable iBoxes. We envision that iBoxes will consolodate several network-based functionalities currently deployed in numerous domain-specific network appliances into one box. To support these applications, we require a mechanism to track state for protocol flows in a flexible and efficient manner. This mechanism is called cPredicates. cPredicates are defined by application-specific ProtocolLogic elements that separate protocol logic from TCP/IP packet processing with support from a CPU-based "backdoor" path. Taking network-based storage as an example application, we built a multimedia streaming service that supports content-based prioritization of video content based on application-level access patterns. We found that iSCSI protocol processing could be done on a command-by-command basis with little overhead (less than 10% amortized latency increase) as compared to stock IP forwarding. This is made possible because approximately 95% of packets transiting the iBox do so through a lightweight fast path. The other 5% are examined by the ProtocolLogic element that encapsulates more general purpose processing. This enables efficient processing of multimedia work-

load between video servers, web servers, and other content severs and producers. We envision that this mechanism can be extended to other application domains as well.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475: An architecture for differentiated services, December 1998. Status: PROPOSED STANDARD.

[2] Dan Decasper, Zubin Dittia, Guru Parulkar, and Bernhard Plattner. Router plugins: a software architecture for next-generation routers. *IEEE/ACM Trans. Netw.*, 8(1):2–15, 2000.

[3] Checkpoint VPN-1 Edge. http://www.checkpoint.com/products/enterprise/index.html.

[4] IBM eServer BladeCenter. http://www-1.ibm-.com/servers/eserver/bladecenter/.

[5] In Submission. Router-transparent annotations.

[6] IETF IP Security Protocol (ipsec) Charter. http://www.ietf.org/html.charters/ipsec-charter.html.

[7] Intel iSCSI Reference Implementation. http:/-/sourceforge.net/projects/intel-iscsi.

[8] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[9] Packeteer PacketShaper. http://www.packeteer.com/prod-sol/products/packetshaper.cfm.

[10] SunFire B1600 Blade Platform. http:/-/www.sun.com/servers/entry/blade/.

[11] Nortel Alteon Portfolio. http://www.nortelnetworks.com/products/01/alteon/.

[12] Apple Darwin Streaming Server. http:-//developer.apple.com/darwin/projects-/streaming/.

[13] Oliver Spatscheck, Jørgen S. Hansen, John H. Hartman, and Larry L. Peterson. Optimizing tcp forwarder performance. *IEEE/ACM Trans. Netw.*, 8(2):146–157, 2000.

[14] TCP splicing for application layer proxy performance. ftp://ftp.cs.cmu.edu/user/dmaltz/doc/splice-perf-tr.ps.

[15] ISO Mpeg standards. http://www.iso.ch/iso/en-/prods-services/popstds/mpeg.html.

[16] HP ProLiant BL Systems. http:/-/h18004.www1.hp.com/products/servers-/platforms/index-bl.html.