OpenFlow Timeouts Demystified

Adam Zarek
Department of Computer Science
University of Toronto

Supervised by:
Yashar Ganjali, David Lie

Abstract

This paper explores the impact of the timeout length on performance, measured through the miss rate, and table occupancy. It finds that as timeouts increase, the miss rate decays exponentially while the table size grows near-linearly. We observe there is an operating point where any further increases in the timeout lead to insignificant reductions in the miss rate while unnecessarily swelling expanding the table occupancy. In one dataset this timeout is at 5 seconds, while it is centered around 10 for three others.

Additionally, this paper introduces hybrid flow table management that combines timeouts with explicit controller eviction messages. It establishes a lower bound of 57% fewer table entries in one dataset without impacting the miss rate, while a practical TCP-based implementation reduces the table size by 42%, or almost 32,000 entries.

Finally, this work compares the performance of different flow table replacement policies. It identifies that while LRU outperforms the other policies, it cannot be implemented in current OpenFlow switches. A FIFO replacement policy performs [**AZ: number!**] worse than LRU but less than 0.1% better than Random replacements.

Based on our observations, the future of OpenFlow timeouts is to delegate the responsibility of assigning timeouts to a control-loop within the controller so the timeout can both be dynamically tuned to individual flows, as well as individual network requirements.

Contents

List of Figures

List of Tables

1 Introduction

One of the responsibilities of the OpenFlow controller is to manage the contents of the flow tables. The primary aspect of table management is deciding how to assign flow rules, but no less important is deciding when and how to remove rules from the flow table. The number of spaces in the flow table is limited and there is no benefit in assigning a table entry to a flow for which no more packets will arrive.

OpenFlow uses per-flow timeouts to manage the lifetime of each flow rule. A flow table entry is removed if no packet matches the rule within a certain *inactivity timeout*. OpenFlow also allows the controller to proactively remove entries with explicit control messages. However, to make use of this functionality the controller may need fine-grained view of packet arrivals, as well as the ability to query stats from the switch, both of which may not be feasible [6]. Under these constraints, timeouts are both more effective and scalable than relying entirely on control messages to remove flow entries.

There is a tradeoff between shorter timeouts, which may prematurely evict a flow rule before all packets matching the flow have arrived, and longer timeouts, which may lead to larger flow tables than necessary. Premature evictions result in unnecessary flow table misses that cause *(i)* an extra round-trip to the controller when the next packet arrives, adding approximately 10ms to the packet's latency [6,17][1]; and *(ii)* an extra *packet_in* event generated and sent to the controller. A large number of premature evictions, therefore, can add a significant load on the controller. We note that previous work on controller performance and scalability assumes that every *packet_in* was from a new flow [10, 17, 18].

Longer timeouts reduce premature evictions, but this comes at a cost. Lengthier timeouts lead to a longer rule lifetime, which in turn leads to a

[1]Assuming that packets are spaced far apart that only one packet will miss and go to the controller. If multiple packets arrive before the flow rule is re-instated, then more packets will bear the cost.

larger table occupancy. Flow tables are built out of TCAMs that support wildcard entries and parallel lookups. TCAMs have high power consumption and a large silicon footprint. As a result, today's switches usually support a very limited number of entries. The HP5406zl OpenFlow-enabled switch supports approximately 1500 OpenFlow rules due to the TCAM limitations and the NEC PF5820 is capped to only 750 entries [6, 13].

To the best of our knowledge, there has been no formal exploration on the effect of OpenFlow timeouts on both the number of table entries and miss rates. Previous work has used timeout values ranging from 5 seconds, in the NOX and Stanford deployments [7, 16], to 10 and 60 seconds, in DevoFlow [6]. While DevoFlow does mention anecdotally that the number of entries decreased sixfold when they reduced the timeout from 60 to 10 seconds, there is no mention of the corresponding impact on performance.

This paper makes three main contributions. First, the paper provides an in-depth exploration of the effect of flow timeouts on both the miss rate and the number of flow entries. We show the benefit of longer timeouts, as measured through the miss rate, decays exponentially, while the number of entries increases almost linearly with longer timeouts. Furthermore, we observe there is an operating point where any further lengthening of the timeout results in insignificant reductions in the miss rate and unnecessarily increasing the table size. We call this value the *break-even point* and find that it is 5 seconds for a data center packet trace, 9 seconds for an enterprise dataset, and 11 seconds for a core network and coincidentally, very close to the timeout values implemented in other work.

Second, this work advances beyond relying solely on timeouts to remove flow entries, but introduces hybrid flow management that combines inactivity timeouts with explicit controller messages to remove table entries. Using a clairvoyant algorithm, we obtain a lower bound of 8.5% and 58% fewer table entries in in two datasets without any increase in the miss rate. We follow up with a more practical algorithm that infers flow state from TCP headers and

eagerly evicts the rule before the timeout expires. Our results show that in one dataset we can reduce the table size by almost 32,000 flow table entries, or 42% of the size, compared to relying strictly on timeouts.

Third, this paper compares the performance of different flow table replacement policies. We show that while LRU has the best performance, it is neither supported in current versions of OpenFlow nor can it be easily implemented in the SDN control-plane. We further see that a FIFO replacement policy outperforms Random replacement by less than 0.1%, but requiring significantly more overhead.

Our main takeaway from this work is that the relationship between timeouts, miss rate and table occupancy is quite complex. For example, we noticed improvements in the miss rate by assigning timeout values at $10, 15, 20, 30, \ldots$ seconds that is not inline with the exponential decay observed with all other timeout values. We also observed that the relationship is very sensitive to temporal traffic patterns and that the results from one network have limited portability to another network. In light of our results, we believe the future of OpenFlow timeouts is for the controller to autonomously assign different timeouts to individual flows tailored to individual network constraints such as miss rate, table size, and power consumption.

2 SDN and OpenFlow Background

Software-defined-networking (SDN) is a rapidly growing field that decouples packet-forwarding from network control. Packet-processing is done in hardware and at line-rate. The network is managed by a controller that can be programmed by the network operator. The controller architecture, whether it is a single device or a cluster [10,19], is not relevant to this work, but rather it is how the controller manages the network resources that is the subject of this paper.

The controller interfaces with the switch over the OpenFlow protocol [12]. OpenFlow is rapidly becoming the dominant protocol for SDN in both academia and industry and while the ideas here can be generalized to most SDN protocols, we only focus here on OpenFlow. The OpenFlow architecture operates on flow-level granularity, where a flow is defined by an n-tuple composed of packet header fields[2].

One of the controller's responsibilities is to decide how a switch should process each flow. The controller accomplishes this by installing *flow rules* in each switch. A flow rule, or a flow entry, consists of a set of matching packet headers, a set of actions, and counters for gathering statistics. The switch looks up every arriving packet in its *flow table* and if a match is found, the switch processes the packet as specified by the action set. If a match is not found, then the switch forwards the packet to the controller to be resolved.

Complementary to the controller's task of installing new rules, it is also responsible for removing flow rules, which is the main subject of this work. The OpenFlow protocol supports two methods of removing entries from a flow table. In the first method the controller sends an explicit FLOW_DEL message to the switch specifying which entry or entries should be removed. The flow, or flows, to be removed are specified by the set of packet headers; all entries that match the specified headers will be deleted from the flow

[2]the value of n depends on different versions of OpenFlow [1, 2]

table.

In the second method, the controller assigns a timeout to each flow entry. When the timeout expires, the switch evicts the rule from the flow table and, optionally, notifies the controller the flow was removed. OpenFlow supports two types of timeouts, an inactivity timeout and a hard timeout. We are not aware of any work using hard timeouts. Different values for inactivity timeouts have been used in recent work, ranging from 5 seconds, in the NOX and Stanford deployments [7, 16], to 10 and 60 seconds, in DevoFlow [6]. To the best of our knowledge, this is the first work that explores different strategies of effectively removing rules from the flow table.

3 Related Work

This paper extends recent work on network caching and table sizing within OpenFlow networks.

3.1 Table Sizing

The question of accurately sizing flow tables in SDN and OpenFlow is not a new research area. Ethane explored the table sizing requirements in a university-sized deployment [4]. It concludes that while switches in different parts of the network will have different requirements, a flow table of between 8K-16K entries should be sufficient. It goes further and determines that using 64B-sized entries it requires about 1MB of storage, which is significantly less than the typical memory requirements in a traditional switch. However, OpenFlow rules are now larger than 64B and support wildcarding on any of the matching fields which will impact the memory requirements.

There is recent work focused on deploying OpenFlow in data center networks [6, 18]. Data centers have very different traffic composition compared to the campus deployments discussed in Ethane. Benson *et al* measured the number of active flows in four data centers and showed the traffic patterns can be classified in two groups [3]. In the first group, the number of active flows ranges between 100 and 500 flows 90% of the time. The second group ranges between 1,000 and 5,000 flows approximately 90% of the time. The order of magnitude difference between the two classes suggests the importance of network instrumentation *before* deploying an OpenFlow network to accurately provision the table size. In addition, the canonical definition of a flow used for this measurement (5-tuple of src/dst IP, src/dst port, protocol) may be different in an OpenFlow network so the size requirements will depend on the network policies.

There has been other work that amidst evaluating specific OpenFlow applications discuss their required flow table size. These works target specific

applications, such as L2 forwarding [5] or data center forwarding [18], and while they provide a rough estimate of the flow table requirements in a similar network, they do not explore how the timeouts they used will affect the table size; which is where this paper fits.

Complementary to sizing flow tables is the challenge of quickly querying them for flow statistics. DevoFlow identified that querying flow statistics from a large flow table takes too long for flow-based scheduling [6]. Curtis *et al* mention that reducing the timeout does have a large impact of reducing table size, before proposing their solution that instead reduces the table size by extensive use of wildcard rules at the cost of limiting the controller's ability to only respond to significant flows.

3.2 Network Caching

Applying cache replacement policies to network routing has been previously studied in [9]. That work analyzed packet traces from a tier-1 ISP's backbone and defined a flow on source and destination /24 prefixes. Most OpenFlow deployments and research activity is currently focused on intra-Autonomous System networks and it is not immediately clear that similar working set properties exist within an AS, so there remains a gap in our knowledge of cache replacement policies using different matching schemes.

Furthermore, due to the nature of the dataset used in [9], Kim *et al* measured the working set by using timeouts ranging from 1 to 60 *minutes*. To the best of our knowledge, the largest timeout value used in any OpenFlow work is 60 seconds [6] so we are further interested to see if how the results may differ with shorter timeout lengths in the order of seconds.

4 The Impact of Timeout Length

This section explores three questions about timeouts in OpenFlow networks: (1) How does the timeout impact the number of flow table entries? (2) How is the performance affected by timeouts (*i.e.* the impact of premature evictions)? And (3), is there a timeout value that balances the tradeoffs between table size and performance?

We answer these questions through a set of experiments that measure the effect of timeouts in an OpenFlow network. We implemented an OpenFlow simulator that replayed a packet-level trace. The simulator assumes an infinite size table where flows are only evicted due to timeouts. We sampled the table occupancy every second and calculated the miss rate at the end of the dataset. The first miss of every flow is a *cold miss* since it is unavoidable in any *reactive* control scheme (where we do not proactively create default rules for flows).

We ran our experiments on 4 network traces summarized in Table 1. The first dataset, UNIV, comes from a university data center [3]. The second dataset, LBNL, is a trace from Lawrence Berkeley National Laboratory's internal network [15]. The published dataset exhibits long inactive periods so we selected a 4 hour slice of active traffic so we could study the timeout effect during peak utilization. CAIDA/32 is a trace from an OC-192 link [8]. We post-processed the traffic by applying a /24 mask and denote it as a new trace, CAIDA/24. All flow rules match on source and destination IP address pairs, except for CAIDA/24 which matches on source and destination prefixes. We note that the anonymization of CAIDA/32 preserves prefixes, and that is why we can apply other prefix masks to this dataset.

4.1 Table Size

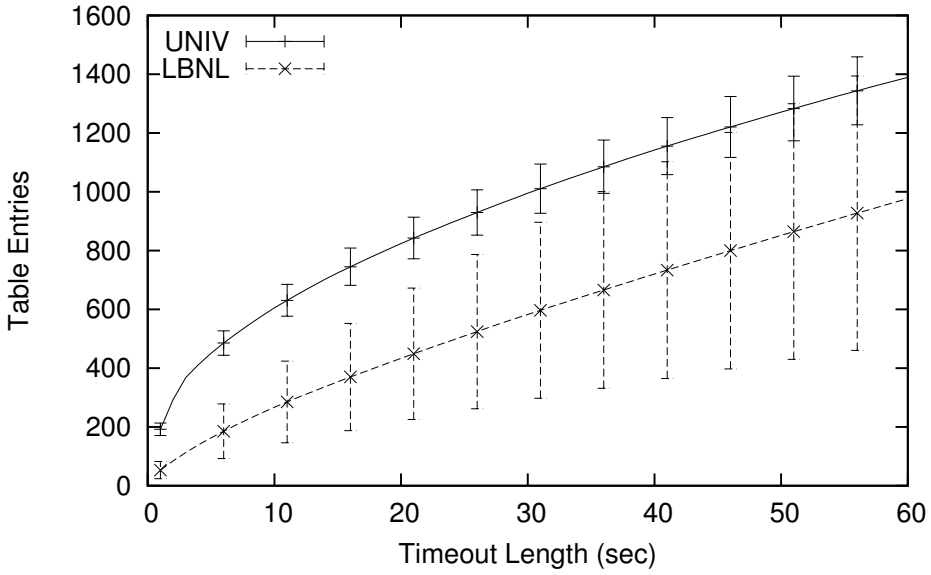
Intuitively, longer timeouts mean each rule will be retained for longer in the flow table, so the table size (i.e. peak occupancy) should grow as timeouts

Table 1: Summary of the 4 datasets we used for our simulations.

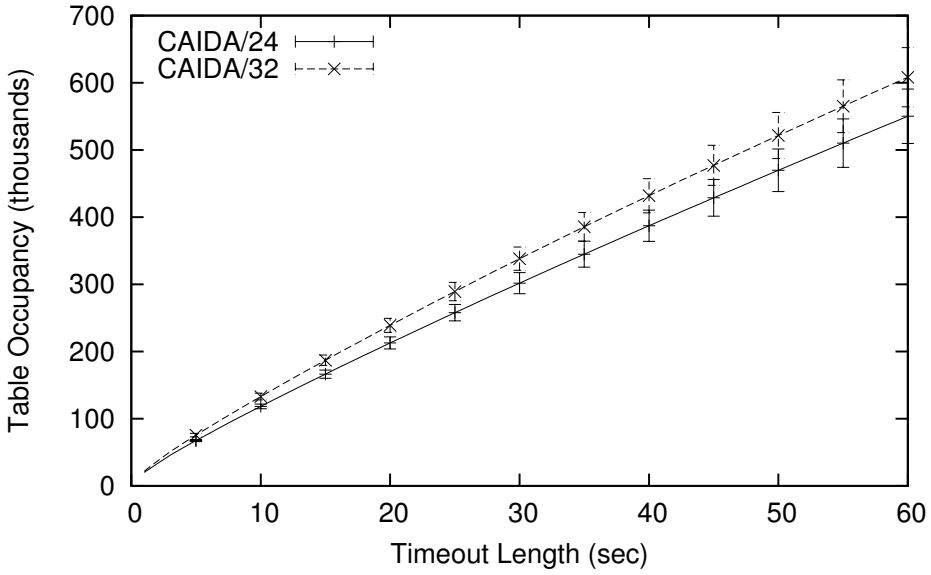
Dataset	LBNL	UNIV	CAIDA/24	CAIDA/32
Packets	6.9M	16M	377M	377M
Flows	73K	19K	19M	21M
Length	3.8 hr	65 min	55 min	55 min
Date	1/7/05	-	1/21/10	1/21/10
Source	[15]	[3]	[8]	[8]

increase. Figure 1 shows the nature of this growth; the table size grows near-linearly with respect to the timeout value. This result is in inline with the observations reported in [6] where they decreased the timeout from 60 to 10 seconds, and the number of entries shrunk by a factor of 6. Since the table size is linearly dependent on the timeout, large adjustments to the timeout, such as in [6], will cause corresponding large changes to the number of flows the switch needs to support.

In contrast to the other datasets, the number of entries in LBNL exhibit a much higher variance. To further elucidate its behaviour, Figure 2 shows the table occupancy over time with timeouts of 5 and 60 seconds. With the 5 second timeout the table size is much more stable and quickly evicts the extra entries after an interval of bursty traffic. It is difficult to say if the 60 second timeout ever stabilizes. This large variance in the number of entries complicates table sizing because with a large table - and higher power costs - there will be long intervals where the table is under-utilized, but a small table means that there will be intervals where the table will be oversubscribed, leading to poor performance.



(a) UNIV and LBNL



(b) CAIDA/24 and CAIDA/32

Figure 1: Mean number of flow entries

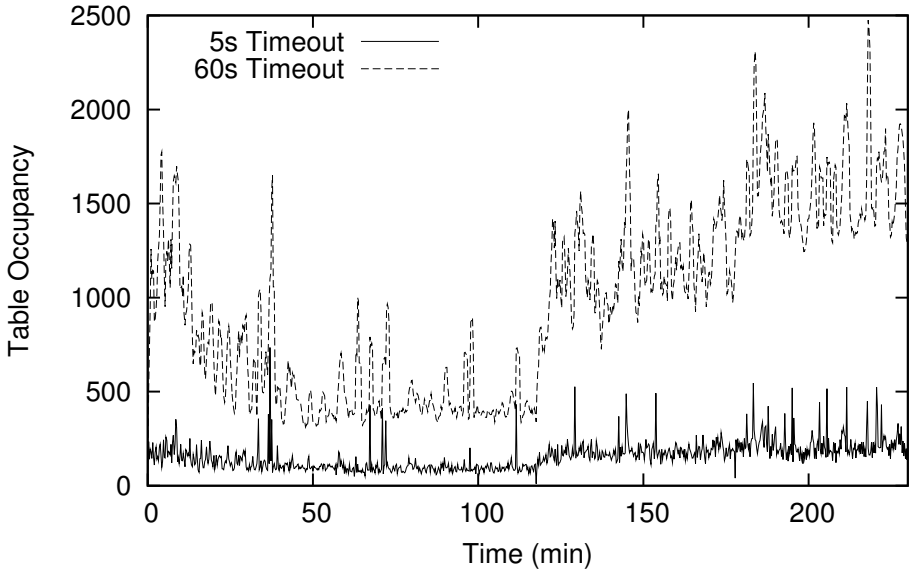


Figure 2: LBNL table occupancy time-series

4.2 Miss Rate

Misses happen when the packet inter-arrival times in a flow is greater than the timeout length. In other words, the miss rate will be related to the CDF of the packet inter-arrivals within each flow.

Figure 3 shows the effect of the timeout length on the miss rate. Our first observation is the large difference in the miss rate between UNIV and the other networks. While they all decay exponentially, UNIV is markedly lower and after 3 seconds the miss rate drops below 1%. On the other hand, LBNL, CAIDA/24, and CAIDA/32 require timeouts of 60, 520, and 620 seconds, respectively, for the miss rate to drop below 1%.

Interestingly, in both the LBNL and UNIV datasets there are drops in the miss rate that are uncharacteristic of exponential decay. Figure 4 shows these changes by plotting the difference in the miss rate as we increase the timeout by one second. Both networks have local maxima centered around multiples of 10 seconds. We suspect these arise from keepalive and control

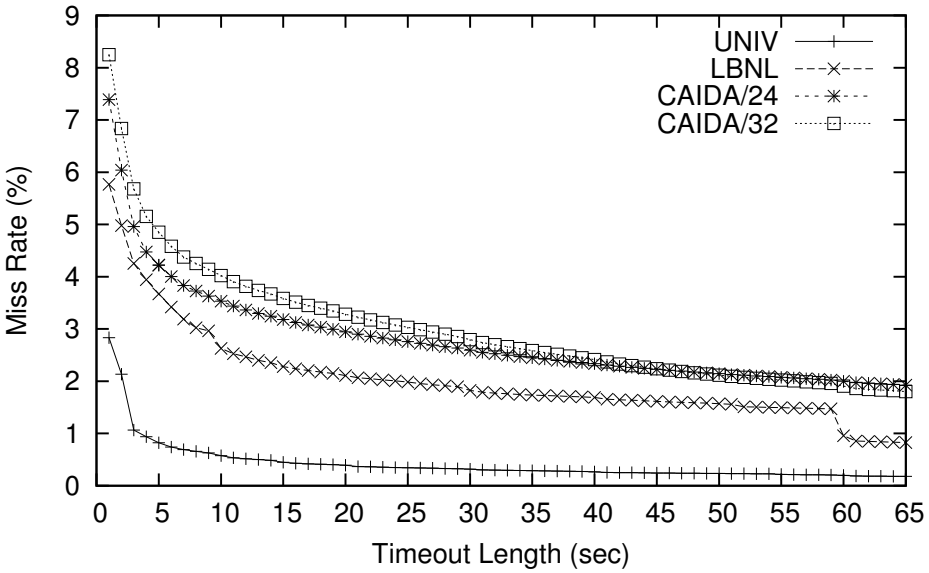


Figure 3: The cold miss rate of each network trace as a function of the timeout. The first miss is ignored since it is not impacted by the timeout.

messages that are transmitted at periodic intervals. For example, we noticed there was one flow in UNIV that had a 98.8% miss rate with a 5 second timeout, but the miss rate dropped to 0% when the timeout was increased by just 1 second. There is further evidence of this effect in LBNL where incrementing the timeout from 59 to 60 seconds used an extra 10 entries, but lowered the miss rate by 0.5%.

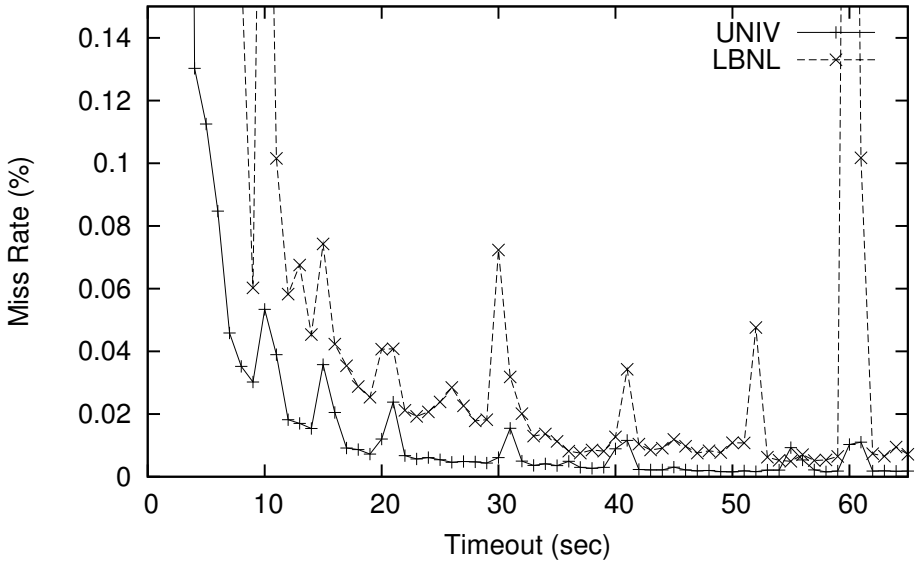


Figure 4: Marginal change to the miss rate if we adjust the timeout by 1 second. Note the peaks at multiples of 10 seconds.

4.3 Break-even Point

As shown above, increasing the timeout grows the table occupancy and reduces the miss rate so there is a definite tradeoff between these two conflicting choices. The exact timeout that optimally straddles these two requirements is entirely subjective and network-dependent, so any attempt to categorically define an optimal timeout is futile.

What we do instead is rely on our previous results to define the network-specific timeout called the *break-even point* where the tradeoff is simplified. We defined the break-even point as the timeout value whereby increasing the timeout by one second causes the miss rate to decrease by less than 0.1%. We select 0.1% because any further levels of precision are likely to be deemed insignificant. Further increasing the timeout beyond the break-even point yields minimal improvement in miss-rate due to exponential decay, but a near-linear increase in flow table entries. The evaluation of the break-even

Table 2: Break-even point timeout for each of the four datasets with their corresponding miss rate and number of entries. This analysis assumes there are no imposed constraints, such as minimal miss rate or maximal table size.

Dataset	Timeout	Miss Rate	Table Size
UNIV	5s	0.82%	451
LBNL	11s	2.53%	285
CAIDA/24	11s	3.91%	144,001
CAIDA/32	9s	3.63%	108,411

point is a local decision and does not consider large drops in the miss rate further in the future, perhaps due to periodic transmissions discussed above.

Table 2 shows the timeouts at the break-even points and the resulting miss rate and number of entries for each dataset. Most surprising to us is how closely these timeouts mirror commonly used timeouts in NOX, the Stanford OpenFlow deployment, and DevoFlow, which use 5, 5, and 10 second timeouts, respectively [6, 7, 16].

However, upon further analysis it appears the similarities with previous timeouts are likely coincidental. DevoFlow uses a 10 second timeout and was designed for high-performance data center networks. Yet, UNIV, a data center network, has its break-even point timeout at 5 seconds, instead of DevoFlow's 10 seconds. Increasing UNIV's timeout from 5 to 10 seconds would decrease the miss rate by 0.24%, but at the added expense of increasing the table size by 34%.

Recall that due to TCAM limitations, some OpenFlow switches support a very small number of flow entries. Using the break-even point timeout, both UNIV and LBNL could comfortably fit within an HP5406zl and the even more constrained NEC PF5820 which have 1500 and 750 entries, respectively [6,13]. If both of these networks were using the HP switch, they could increase the timeout to 13 seconds for UNIV and 62 seconds for LBNL, with miss rates of 0.5% and 0.85%.

5 Hybrid Management Algorithms

This section introduces hybrid table management that uses both timeouts and controller messages to control the flow table size. This section looks at algorithms that infer flow state while the subsequent section explores algorithms that model the flow table as a cached system.

5.1 Clairvoyant Algorithm

Hybrid approaches are a new technique to manage flow table contents with, as of yet, unexplored potential. A hybrid algorithm adds extra overhead to both the controller and the switch. To evaluate the potential benefit of integrating the controller as an active participant in flow removals, we want to look at the best-case scenario and determine if the benefits make this worth pursuing.

If the objective is to minimize the overall miss rate, then the provably optimal algorithm is to leave each flow entry in the table until it completes. The first packet of each flow will miss, but then all subsequent packets will find a matching entry. While this is optimal with respect to miss rate, it results in unreasonably large table sizes – on the UNIV dataset it increased the table size by 16 times compared to using 5 second timeout. Thus, we want an algorithm that has a miss-rate similar to inactivity timeouts, but a better table size.

To find an algorithm that optimizes for table size without sacrificing miss rate, we devise a clairvoyant algorithm. The algorithm,shown in Algorithm 1, combines both OpenFlow timeouts and proactive controller evictions. If a packet does not have a matching OpenFlow entry in the switch, the controller installs the rule with the designated timeout, as before. In addition, the switch forwards every packet to the controller, where it looks up if the subsequent packet from the flow will arrive within the timeout interval. If the interarrival time is greater than the timeout, then the controller removes

Input: Timeout
foreach *PacketArrival p* **do**
  if *nextArrival(p) > Timeout* **then**
    $flow$=extractFlow(p) removeFlowFromTable($flow$)
  end
end

Algorithm 1: Flow table management algorithm given clairvoyant capabilities.

the corresponding entry[3].

It is straightforward to compare our clairvoyant algorithm with inactivity timeouts method because they both have the same miss rate. In both cases the packet will miss if the if the per-flow interarrival time is greater than the timeout. By eliminating the time the spent waiting for the timeout to expire, we can eagerly evict OpenFlow rules and reduce the overall table occupancy.

There are two flow-level events that will lead us to reductions in the length of time the flow is in the table, when the interarrival time is greater than the timeout and when the flow is complete. We leverage this observation in the following section when we discuss a practical implementation of this algorithm.

5.2 TCP-Based Heuristic

For obvious reasons our clairvoyant algorithm is not a realistic approach that can be implemented in a real network. In this section we introduce our preliminary work that mimics its behaviour without having to rely on perfect knowledge of future network events.

Previously we evicted flow entries when they were either complete or had an interarrival gap longer than the timeout length. Our approach here is to use protocol-level features to detect these events. We rely on TCP flags to

[3]Since we never wait for the timeouts to expire, the timeouts do not need to be set and flow entries could instead be assigned infinite timeouts.

detect imminent flow completions so we can evict the rule before the timeout expires. We do not yet have a heuristic of detecting long interarrivals, though we are aware of other work that integrates sampling into OpenFlow that may help us detect these events.

There are several limitations to using TCP flags to detect flow completion. Using TCP flags forces us to rely on transport-layer flows so flow aggregation using wildcards can result in false positives that cause premature eviction. One method to mitigate this issue is to have the controller maintain a counter that tracks the number of TCP flows that match each rule and to evict the rule when all flows have completed. However, this does require us to store extra flow-level state at the controller.

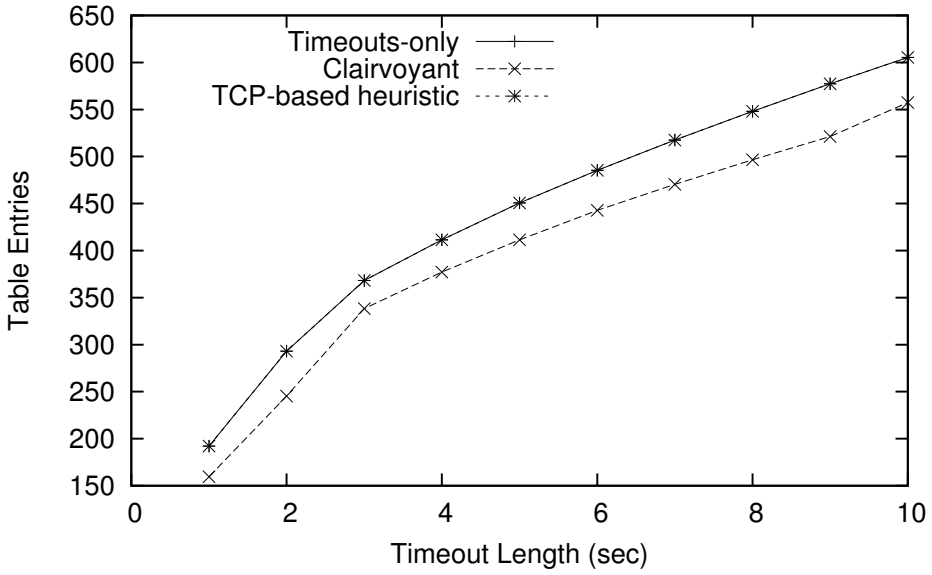
OpenFlow does not directly support matching on TCP header flags, but it can be extended using extensible matching supported in OpenFlow 1.2 [14]. The controller can then install a rule that matches any packets on the SYN, RST, or FIN flags and forwards it to the controller.

Finally, using TCP header flags to detect flow completion limits this approach to TCP flows. However, recent network measurements show that 95% of traffic is TCP so targeting TCP can still be effective [11].

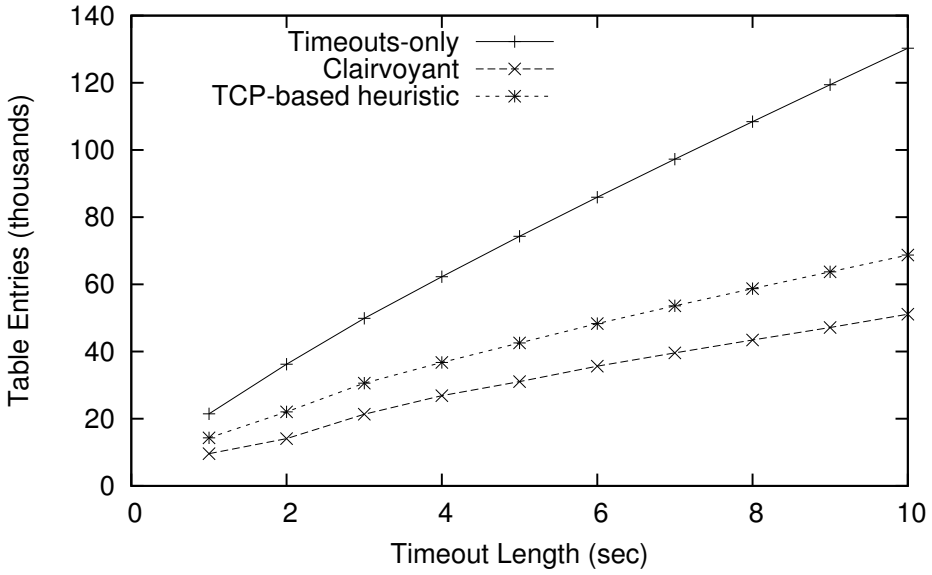
5.3 Evaluation

We ran both hybrid algorithms on the UNIV dataset and the first 10 minutes of CAIDA/32 with timeouts ranging from 1 to 10 seconds. Figure 5 compares the average number of entries in the hybrid flow management approaches to the conventional case of only relying on inactivity timeouts.

Our first observation is that in both datasets, the hybrid approach performs better as the timeout interval increases because each eager eviction has a larger effect. The CAIDA/32 dataset experiences large savings in table size under both the clairvoyant and TCP-based schemes. Using a 5 second timeout, the clairvoyant algorithm uses 58% fewer entries and the heuristic comes within 74% of the clairvoyant hybrid mechanism. This shows that if



(a) UNIV



(b) CAIDA/32

Figure 5: Mean table occupancy under three different timeout mechanisms.

the controller has the capability to infer flow state, there is potential for large savings by having the controller actively remove complete or inactive flows before the timeouts expire.

These savings however are not universal. In the UNIV dataset, the clairvoyant algorithm uses only 8.5% fewer entries than inactivity timeouts, while the TCP-based algorithm barely outperforms timeouts. Compared to the CAIDA/32 dataset, the UNIV dataset has fewer misses to begin with, providing fewer opportunities for the TCP-based algorithm to save an entry through proactive eviction. The clairvoyant algorithm does better because it is able to proactively evict entries due to a long inter-arrival gap while the TCP-based algorithm is not.

6 Cache-Based Policies

Up to this point our main focus has been on predominantly relying on timeouts to regulate flow table size where entries are eagerly evicted once the timeout expires independently of the actual table size. This section explores a cache-based model that uses lazy evictions and only evicts entries when the table is full, an issue that has not been addressed previously.

6.1 Cache Replacement Policies

Previous work has shown the effectiveness of LRU in route caching [9], but for architectural reasons LRU is impossible to practically implement in current OpenFlow switches. In conventional cached-systems where the control-plane is co-resident with the data-plane, the control-plane has complete visibility into the data-plane accesses and can use these access patterns to implement an algorithm like LRU that relies on access ordering. However, in SDN the data-plane exposes a much more narrow interface to the control-plane. Instead of having access to the exact ordering of cache accesses, the controller only has access to coarse-grained counters, and even accessing the counters is not available in real-time [6]. The result is any cache policy that relies on the order of accesses cannot be implemented by the controller.

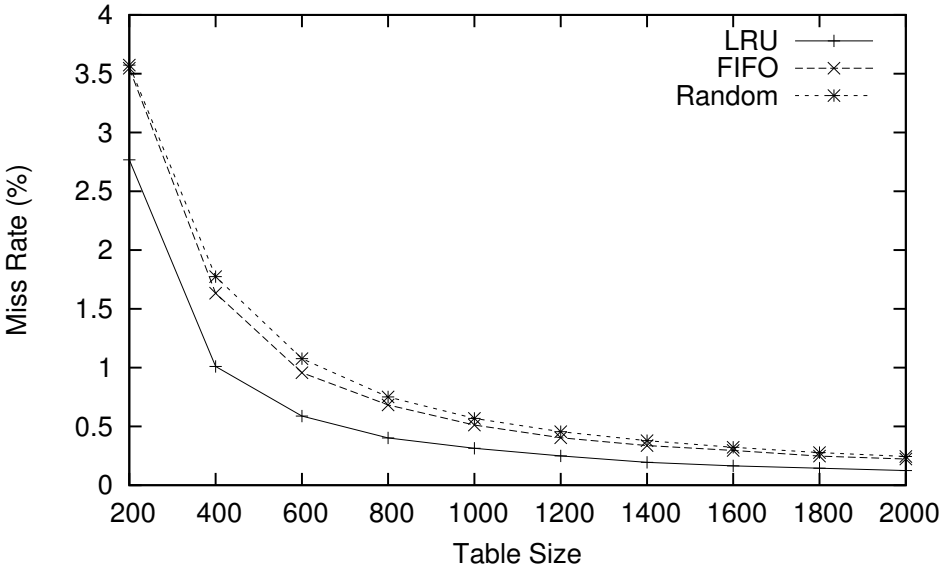
To further complicate matters, algorithms that rely on knowing the flow table state may also be prohibitly expensive. Instead of inherently knowing the cache contents by the nature of co-residency, an OpenFlow controller would need to track the table contents with a local "shadow flow table". Shadow flow table inserts are simple, but tracking flow removals is slightly more complicated. The controller must instruct the switch to send a FLOW_REMOVED message to the controller on every flow timeout. In a network with a large number of switches and each switch have potentially tens of thousand of entries, it is unclear how this will impact both switch and controller scalability.

Notwithstanding these difficulties, we explore the performance of cache-based management schemes for two reasons. First, we are aware of complementary SDN research that runs local network applications within the same hypervisor as virtual switches. This new architecture could reduce the overhead of a shadow flow table, in addition to finer-grained access of access ordering. Second, while algorithms that rely on access ordering may be be difficult to implement, there are other cache replacement policies that could be more easily applied to OpenFlow networks.

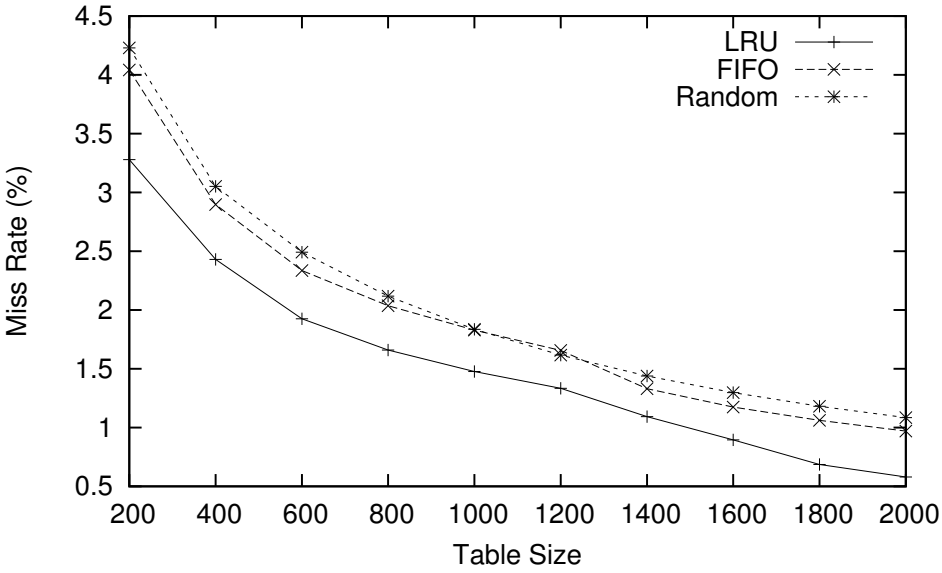
Finally, the question arises of the performance and table sizing effects of combining timeouts and a replacement policy. As already shown in Section 4, timeouts can significantly reduce the number of table entries. However, we were unsure of the effects once the physical table is reduced so the flow table is at capacity.

6.2 Evaluation

We compare the performance of three cache replacement policies, LRU, FIFO, and Random, on the UNIV and LBNL datasets. Figure 6 shows that while LRU outperforms the other replacement policies, as expected, the difference is not as significant as the differences between LRU and LFU reported in [9]. Furthermore, the differences between the FIFO and Random replacement policies are relatively small. Random replacement adds negligible overhead to the controller so it is a better choice than FIFO in these datasets.



(a) UNIV



(b) LBNL

Figure 6: Cold miss rates under LRU, FIFO, and Random replacement policies.

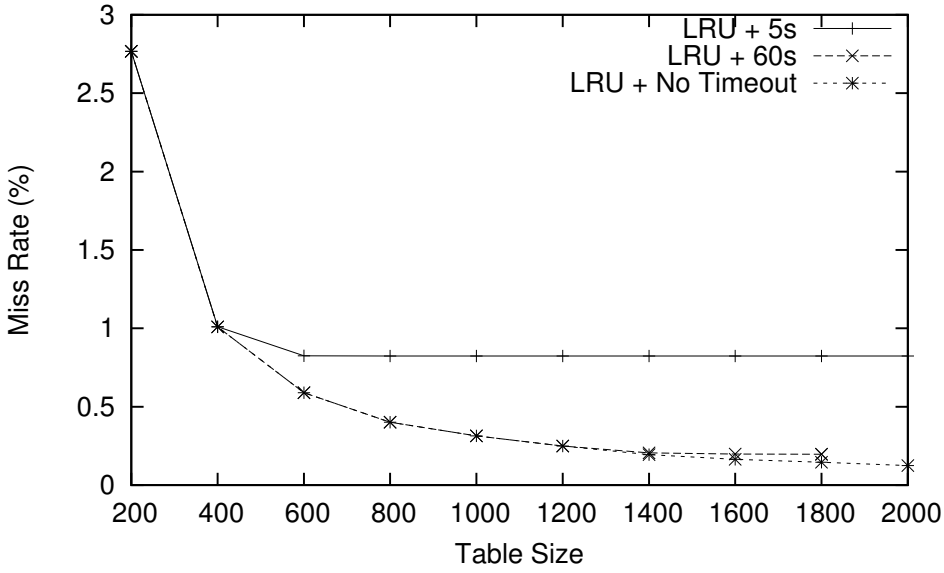
Next, we focus on the performance of combining timeouts with LRU replacement[4]. Similar to [9], we refer to the *working set* as the number of active flows accessed in a given unit of time; in this case, the time unit is the timeout interval.

Combining Figure 6 and Figure 7, we identify three operating regions showing the relationship between timeout value and replacement policy on the miss rate. The first region arises when the table size is smaller than the number of active flows for the *smallest* timeout. In this region the timeout value does not have a large impact because the number of active flows is larger than the table size. Most entries will be evicted due to the replacement policy and not from the timeout expiring. Any performance improvements within this area are dependent on a well-implemented replacement policy.

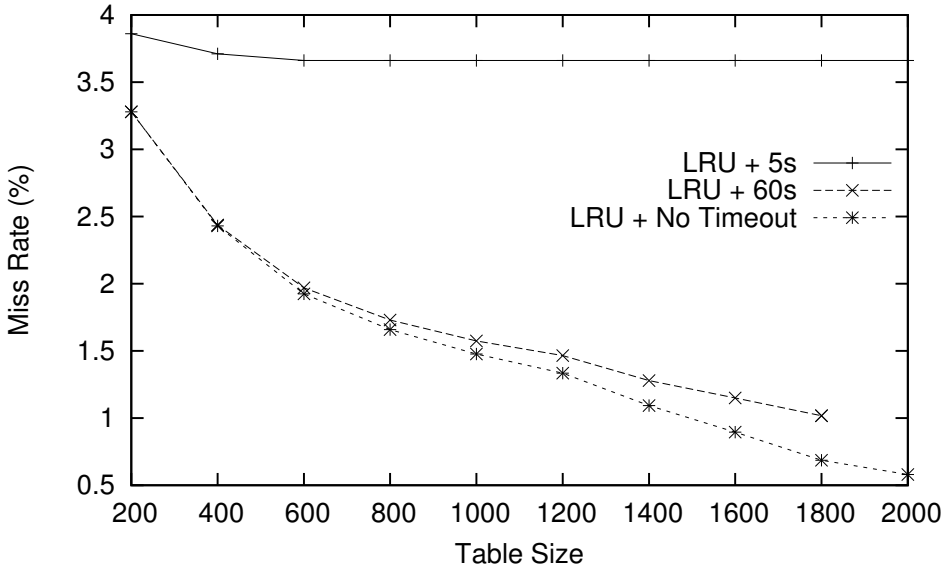
As the table size increases, we begin to see miss rates diverge for different timeout values. Within this second operating region, the working set is still larger than the table size and the table size, timeout length, and replacement policy will all impact the resulting miss rate.

The third operating region begins once the table size is larger than the working set. The miss rate stabilizes within this area and similar to the break-even point, increasing the table size any further is unnecessary and does not lead to any performance improvement.

[4]Even though LRU is not supported by current OpenFlow switches, we select LRU for this comparison because it provides a near-optimal lower-bound on the performance, based on [9].



(a) UNIV



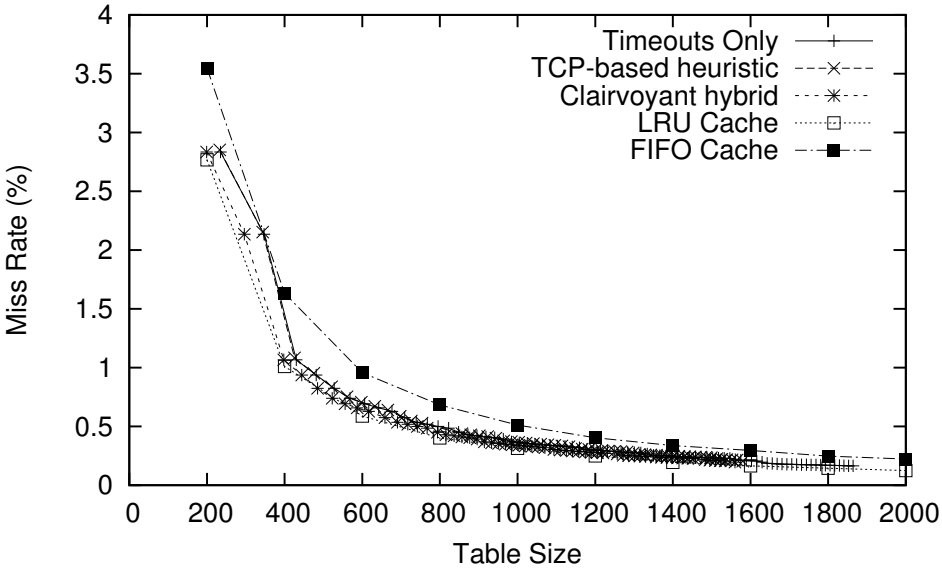
(b) LBNL

Figure 7: Cold miss rate of UNIV and LBNL when combining timeouts with LRU replacement.

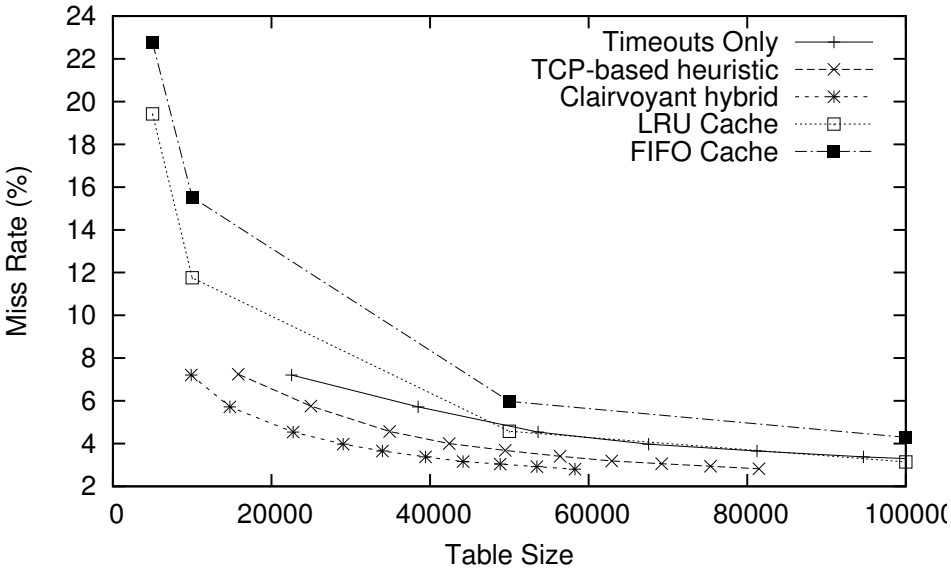
Finally, we compare the performance of a complete cache-based model to our hybrid scheme proposed in the previous section. It is difficult to capture the table requirements for both the timeout-only and hybrid algorithms because we did our experiments assuming an infinitely-sized table and no replacement. We estimate the required table size by adding twice the standard deviation to the expected number of table entries. We have no method for estimating the new miss rate as a result of constraining the table size and we acknowledge the comparison is not perfect so our conclusions are limited.

Figure 8 shows the miss rate and table requirements for different management models in the UNIV and the first 10 minutes of CAIDA/32 datasets. Except for the FIFO replacement, all of the methods closely approximate each other in the UNIV dataset. Due to the nature of the dataset, there is no large benefit to choosing one management scheme over another.

With CAIDA/32, the extremely large miss rate with small tables, upwards of 12%, combined with difficulty projecting an infinite-sized flow table to a finite-sized table makes the comparison very challenging. However, the hybrid algorithms do not exhibit exponential-like growth in the miss rate with small tables, which bolsters our initial decision to pursue hybrid algorithms as a mechanism to manage the flow table contents.



(a) UNIV



(b) CAIDA/32

Figure 8: Table size and miss rate of different flow table management schemes.

7 The Future of Timeouts

After running our experiments, there were three observations that suggest the future of timeouts in OpenFlow networks. First, traffic sent at regular intervals, shown clearest in LBNL, poses a conundrum if the controller assigns the same timeout value to all flow rules. From a performance perspective, it is better to lengthen the timeout so periodic control and keepalive packets with their predictable arrival times do not repeatedly miss. However, it may not be worth extending the timeout for all flows on account of these regular arrivals.

Second, selecting an effective timeout that meets any network constraint requires first capturing traffic, analyzing the trace and then reconfiguring the controller. The ability to adapt the timeout to network changes is limited by how frequently the controller is reconfigured and the overhead of capturing and analyzing the trace.

Third, we saw that applying the same timeout to different networks gives vastly different results. There are no guarantees that a timeout in one network will have similar results in a second network. Each individual operator will need to independently tune the timeout value.

The solution to all three of these challenges is to treat the timeout value as a tunable parameter that can be fully managed by the controller in real-time. Instead of specifying the timeout as an input, the controller should dynamically adapt the timeout to achieve miss rate, table size, and/or power constraints, which should be treated as the more meaningful inputs. Furthermore, the controller should forgo the simplicity of assigning the same timeout to all flows and instead the admittedly more complex, yet more powerful, paradigm of assigning appropriate timeouts to each individual flow. In this way the operator no longer needs to straddle the performance tradeoff of performance and table size, but can instead specify his objectives and let the controller manage the underlying flows.

8 Conclusion

OpenFlow networks are dependent on the efficient insertion and removals of rules in the flow table. While there has been much focus on different approaches to inserting rules, as of yet there has been no work on how to efficiently expunge rules.

This paper presented the first evaluation of how inactivity timeouts affect both the miss rate and flow table sizing, demonstrating the significant impact inactivity timeouts can have on switch resources. It establishes that the miss rate decays exponentially and the flow table size grows almost linearly as the timeouts increases. We observed there is an timeout value whereby any additional lengthening of the timeout has an insignificant improvement to the miss rate at the expense of increasing the table size.

We proposed a hybrid flow table management scheme that uses both timeouts and controller messages to remove flow rules. A clairvoyant algorithm reduces the number of flow entries by 57% in a core network without affecting the miss rate. Preliminary work on an heuristic that leverages TCP protocol state to infer flow events approximates the clairvoyant algorithm by 75%.

Finally, we foresee the future of OpenFlow timeouts as one where the controller dynamically tunes the timeout value based on traffic and network conditions to achieve a predefined miss rate or table size. Even though we have assumed all flows are assigned the same timeout values, such an adaptive scheme might lead to significant gains in terms of table size and/or miss rate by assigning different timeout values to different flows and/or classes of flows.

References

[1] OpenFlow switch specification, v1.0.0. http://bit.ly/openflow10.

[2] OpenFlow switch specification, v1.1.0. http://bit.ly/openflow11.

[3] Benson, T., Akella, A., and Maltz, D. A. Network traffic characteristics of data centers in the wild. In *IMC 2010* (2010).

[4] Casado, M., Freedman, M. J., Pettit, J., Luo, J., Mckeown, N., and Shenker, S. Ethane: Taking control of the enterprise. In *Proceedings of SIGCOMM 2007* (2007).

[5] Casado, M., Koponen, T., Moon, D., and Shenker, S. Rethinking packet forwarding hardware. In *Proc. Seventh ACM SIGCOMM HotNets Workshop* (2008).

[6] Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. Devoflow: scaling flow management for high-performance networks. SIGCOMM '11.

[7] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review 38*, 3 (2008).

[8] kc claffy, Andersen, D., and Hick, P. The CAIDA anonymized 2010 internet traces - Jan 21, 2010. http://www.caida.org/data/passive/passive_2010_dataset.xml.

[9] Kim, C., Caesar, M., Gerber, A., and Rexford, J. Revisiting route caching: The world should be flat. In *PAM 2009*.

[10] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., et al. Onix: A distributed control platform for large-scale production networks. In *Proc. OSDI* (2010).

[11] Lee, D., Carpenter, B., and Brownlee, N. Observations of UDP to TCP ratio and port numbers. In *ICIMP* (2010).

[12] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (2008), 6974.

[13] NEC. NEC ProgrammableFlow Networking. http://www.necam.com/PFlow/, Mar. 2012.

[14] Open Networking Foundation. OpenFlow switch specification, v1.2. http://bit.ly/HsWqI2, Dec. 2011.

[15] Pang, R., Allman, M., Bennett, M., Lee, J., Paxson, V., and Tierney, B. A first look at modern enterprise traffic. In *IMC 2005*.

[16] Sherwood, R. Personal email, Nov 2011.

[17] Sherwood, R., Gibb, G., Yap, K. K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G. Can the production network be the testbed. In *USENIX OSDI 2010*.

[18] Tavakoli, A., Casado, M., Koponen, T., and Shenker, S. Applying NOX to the datacenter. In *HotNets-VIII* (2009).

[19] Tootoonchian, A., and Ganjali, Y. HyperFlow: a distributed control plane for OpenFlow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking* (2010), p. 3.