Adaptive Analysis of High-Speed Router Performance in Packet-Switched Networks

N.M. Mosharaf Kabir Chowdhury

David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
nmmkchow@cs.uwaterloo.ca

Abstract. Routers perform expensive *lookup* operations in their *router table*s to find appropriate output interfaces to forward packets toward their final destinations. Over the years several works have been done to optimize the lookup operation. We consider that it is also equally important to decrease the number of lookup operations altogether. In this paper we promote the idea of *niceness* of an incoming packet sequence to show that some sequences are more likely to result in fewer lookup operations than others. In addition, we propose five new measures of difficulty to identify and measure the niceness of a sequence.

1 Introduction

The Internet is a global, publicly accessible, complex network of interconnected computer networks that transmit data by packet switching using the standard Internet Protocol(IP). Each message is divided into smaller parts or packets at the source, which are then routed separately using the address in their headers through nodes over data links shared with other traffic. Once all the packets forming a message arrive at the destination, they are merged together to form the original message.

Traditional IP routing is relatively simple in the sense that it uses *next-hop* routing where the router only needs to consider the next destination of the packet, and does not need to consider the subsequent path of the packet on the remaining hops toward its actual destination. Whenever a packet arrives at a router, that router consults its *router table* to determine the output interface through which to forward the packet toward its ultimate destination. Even though it is a seemingly simple *lookup* operation, it becomes significant when the arrival rate of packets is larger than the speed of a lookup operation. This is exactly the case in the Internet backbone where high-speed routers are employed to handle such scenario. Performance of such router is largely dominated by its ability to optimize the lookup operation. A lot of research have been done over the years to speedup this process both in software and hardware [12, 6, 7, 15, 17, 16, 13].

Apart from the speed of the lookup operation, performance of a router is also dependent on the total number of lookup operations it performs for a given sequence of packets. Depending on the order of the packet sequence, total number of lookup operations in a given timespan can vary significantly. For example, let us assume that in a given timespan, n packets of two different destinations, say $dest_A$ and $dest_B$ arrive at the router and the router can remember only one entry from its router table. On the one hand, if the packets interleave perfectly the router must perform $\Omega(n)$ lookup operations. On the other hand, if all the packets with $dest_A(dest_B)$ follow the packets with $dest_B(dest_A)$ only $O(1)$ operations are necessary.

This simple observation leads to the idea of an *adaptive* algorithm [9, 14]. Such an algorithm determines the type of an instance as it proceeds without making any a priori assumptions. The objective of this paper is to find ways to identify and measure the *niceness* of an input packet sequence to a router. We say a sequence is *nicer* than another sequence if the former one results in fewer lookup operations than the later one. It is more challenging in this particular context because of the unavailability of the future knowledge which requires *online* analysis [4].

Paper Organization. The remainder of the paper is organized as follows. In Section 2 we define the models and notations that are used throughout the paper. In Section 3 we propose two completely new measures of difficulty, $MaxDis$ and $SumDis$, for this problem and three other difficulty measures, $Runs$, Exc and Rem, modified from the sorting problem to fit in this problem domain. In Section Algorithms we discuss issues regarding optimal algorithms for the proposed measures.

2 Model and Notation

In this section we formalize the model and the notation. We consider three main buffer models: the FIFO non-preemptive model, the FIFO preemptive model and the bounded delay model. First, we summarize the assumptions and notations that are used throughout the paper.

We define *input queue* as a link through which packets arrive at a router and *buffer* as a collection of internal spaces in the router where packets are stored. We use packet *type* to denote packets with different destinations. All the packets that have same destination are considered to be of same type. And *cache* is defined as very high speed small-amount of memory to hold a portion of the routing table.

We assume that time is discrete. At each time step t, there is a set of *packets* $Q(t)$ stored at the *buffer* of size B (initially $B = 0$). At time t, a set of packets $A(t)$ arrives through the *input queues*. $T(t)$ and $D(t)$ denote the set of packets that are *transmitted* and *dropped* respectively, at time t. We only consider the packets that are dropped inside the router, not those that are dropped because input queues were full. There is a cost associated with each of the $D(t)$ dropped

packets. The set of packets that remain in the buffer at time $t+1$ is $Q(t+1) = Q(A) \cup A(t) \setminus (T(t) \cup D(t))$. $C(t)$ denotes the entries from the routing table the cache is holding at the beginning of time t and C denotes the maximum possible size of this cache.

To move a packet toward its destination, a router has to find the next router in the path to forward the packet. At any moment, the router first consults the cache, $C(t)$ and if the mapping is found in the cache we call it a *hit*; otherwise we call the event a *miss* and for each miss the router has to perform a *lookup* operation which is very expensive in case of a high speed router. Total number of hits and misses at time t are denoted by $H(t)$ and $M(t)$ respectively. In this notation the less the sum $\sum_t M(t)$ is, the better the performance of a router.

There is an integer W called the *link bandwidth* which determines the maximum number of packets that can be delivered by the router at a time i.e. $|T(t)| \leq W$ for all t.

The sequence of packets transmitted by an algorithm obeys specific properties of a model it is working in. Now we specify three different buffer models that are commonly used in this context.

FIFO Non-preemptive Model [10]. This is the basic FIFO model with two constraints. First, the sequence of transmitted packets is a subsequence of the arriving packets. Second, at all time $|Q(t)| \leq B$. Once a packet is admitted it cannot be dropped from the queue so the decision regarding rejection must be taken at admission.

FIFO Preemptive Model [2, 11]. It is same as the previous model except that this model permits algorithms to preempt (drop) already accepted packets. To maintain the FIFO order, packets are always added at the end of the queue, and transmitted from the beginning of the queue. In addition, a packet can be dropped from the middle and the packets behind it shift forwards.

Bounded Delay Model [10]. In this model each packet p has an additional attribute called deadline, $dl(p)$. Any packet $p \in A(t)$ must be delivered or dropped within $t + dl(p)$. Also, packets can be rearranged in this model [10]. When all the packets have same deadline δ, we call this model δ-*uniform bounded delay model* and when all the packets have deadline bounded by δ it is called δ-*variable bounded delay model*. We will mainly concern ourselves with uniform bounded delay model.

3 Measures of Difficulty

The convention in traditional algorithmic analysis to measure the difficulty of a problem is to find out its worst-case complexity. But there are cases where some problem instances can be easily solved than others. Traditional difficulty measures cannot differentiate between such cases. This observation was first

made by Burge [5] for sorting problems in 1958 when he discovered the idea of *presortedness* in a sequence. After that, many difficulty measures for the sorting problem have been proposed and relationships between these measures have been identified and critically examined over the years [9, 14].

In this section we examine the idea of niceness for any incoming packet sequence and propose difficulty measures to identify as well as quantify this property. Many of these difficulty measures are based on their counterparts in the sorting problem domain with necessary modifications to fit in this problem domain.

Niceness. As mentioned before, we evaluate instance easiness in this problem by a measure of niceness. A sequence is *nicer* than another sequence if the first one requires less lookup operations than the other one.

Example 3.1. Let us assume five different types of packets $\{A, B, C, D, E\}$ and at time t, $Q_1(t) = \{A, B, E, D, C, A, E, C, C, A, A, B, A, D, D\}$ where packets are stored in left-to-right order in the buffer. Assuming $C = 1$, there will be 12 misses in this sequence i.e. $M_1(t) = 12$. If the sequence of arrival was a bit different, say $Q_2(t) = \{A, B, E, D, A, E, C, C, C, A, A, A, B, D, D\}$ then $M_2(t) = 10$ which is better than the first sequence and hence *nicer*. And an optimal sequence is $Q_{OPT}(t) = \{A, A, A, A, A, B, B, C, C, C, D, D, D, E, E\}$ with $M_{OPT}(t) = 5$.

In fact, considering only this instance at time t there are $5! = 120$ possible sequences that will result in exactly same number of misses i.e. they are equally nice. This is also true for offline case where the future is known beforehand. But observing a little more carefully we notice that even though they are all optimal at t, all of them are not necessarily equal when considered over a longer timespan if the future is unknown i.e. in online case. As the content of the cache at the end of time t will be used at time $t + 1$, the last element to be in the cache at time t is very important. So in this example, with $C = 1$, $Q_{OPT}(t)$ can now have only $4! = 24$ possible optimal sequences and this number will decrease with the increase of C, the size of the cache up to a certain limit. So the notion of niceness is dependent upon the knowledge of the future as well as the size of the buffer and the cache.

Measures. We propose the following difficulty measures to quantify the niceness of a sequence of packets in a router. It should be noted that constraints imposed by different models (Section 2) may make a measure less useful in one model than another.

3.1 *Runs*

Since consecutive occurrences of same type of packets do not result in any subsequent lookup operations, number of runs is a natural measure of difficulty for this problem. Unlike its counterpart in the sorting problem, it has no condition

on the runs to be ascending or descending. We define *Runs* as the number of boundaries between runs. We call this boundaries **flip**, at which there is a change of packet type.

Example 3.2. If there are two sequences $X_1 = \{E/C, C, C/A/B/A, A/B/D, D\}$ and $X_2 = \{E/C, C, C/A, A, A/B, B/D, D\}$, then they have $Runs(X_1) = 6$ and $Runs(X_2) = 4$ respectively.

In order to make the function zero for a sequence with minimum number of flips, we redefine *Runs* for a sequence X as,

$$Runs(X) = (\text{Number of flips} - |\mathcal{P}| + 1)$$

where $\mathcal{P}$ is the set of all different types of packets. It should be noted that *Runs* suffers from the problem described earlier in this section i.e. it does not differentiate among the $|\mathcal{P}|!$ possible sequences with same *Runs* value.

3.2 *SumDis* and *MaxDis*

The bigger weakness of *Runs* is its inability to take into account the presence of cache rather than its failure to know the future.

Example 3.3. Consider packet sequences $X_3 = \{A, A, A/B, B, B/C, C, C/D\}$, $X_4 = \{A/C/B/A/B/C/B/A/C/D\}$ and $X_5 = \{B/A/C/B/A/D/C/B/A/C\}$ with $Runs(X_3) = 0$ and $Runs(X_4) = Runs(X_5) = 6$. At the presence of a cache of size $C = 3$, both X_3 and X_4 need same number of lookup operations whereas X_5, even with this enlarged cache, results in more lookup operations.

To capture this information we need a finer measure of difficulty. *SumDis* is defined as the sum of the **TypeDistance** for all different types of packets, where TypeDistance is defined as the maximum number of different types of packets in between two consecutive packets of same type less the cache size or zero when it is negative. If $\mathcal{P} = \{p_1, p_2, \ldots, p_{|\mathcal{P}|}\}$, l_i denotes the total number of packets of type p_i and $(p_{i,k} \leftrightarrow p_{i,l})$ denotes the number of packets of type $p_j (j \neq i)$ located between two packets of type p_i then,

$$TypeDistance(p_i) = \max\left(\max_{0 \leq j < l_i} (p_{i,j} \leftrightarrow p_{i,j+1}) - C + 1, 0\right)$$

and for a sequence X,

$$SumDis(X) = \sum_{i=1}^{|\mathcal{P}|} TypeDistance(p_i)$$

To illustrate this, let us consider the sequences X_3, X_4 and X_5 again. In case of X_3 and X_4, $TypeDistance(A) = TypeDistance(B) = TypeDistance(C) = TypeDistance(D) = 0$ and consequently $SumDis(X_3) = SumDis(X_4) = 0$. But for X_5, $TypeDistance(A) = TypeDistance(B) = TypeDistance(C) = 1$ and $TypeDistance(D) = 0$, which results in $SumDis(X_5) = 3$.

Observation 3.1. *An intuitive observation about $TypeDistance(p_i)$ is that its favorable values lie in very low or very high range. If it is very low then information about a packet should be kept in the cache. On the contrary, if it is very high then this information can be purged in case of overflow. But when the value is somewhere in the middle, neither too low to keep the record nor high enough to remove it, then it becomes a challenging task for an algorithm to take optimal decision.*

This leads to another difficulty measure very similar to *SumDis*, which we call *MaxDis*. For a sequence X,

$$MaxDis(X) = \max_{1 \leq i \leq |\mathcal{P}|} TypeDistance(p_i)$$

For the sequences X_3, X_4 and X_5 from the previous example, $MaxDis(X_3) = MaxDis(X_4) = 0$ and $MaxDis(X_5) = 1$.

Another point to notice about $SumDis$ and $MaxDis$ is that their cache-consciousness makes them very useful in extending the definitions of existing measures of presortedness into this problem domain without much trouble. *Exc* and *Rem*, described later, are two simple examples of such extension.

3.3 *Exc*

The number of operations required to rearrange a sequence of incoming packets to decrease the number of lookup operations may be the prime concern of an algorithm. **Exchange** is such a simple rearranging operation. We define *Exc* as the minimum number of exchanges needed to rearrange an input packet sequence to make its *SumDis* measure zero. For example, X_5 needs only one exchange operation to make $SumDis(X_5) = 0$. If we exchange the only type-D packet with the first type-B packet we can achieve that; hence $Exc(X_5) = 1$.

3.4 *Rem*

In some models it is possible to preempt or drop accepted packets. As a result, an algorithm can try to make a sequence nicer by dropping some packets from specific positions. *Rem* is defined as the minimum number of packets that must be dropped to reduce the *SumDis* measure of a sequence to zero. We must drop at least one packet, e.g. the only type-D packet, from X_5 to have $SumDis(X_5) = 0$ and as a result $Rem(X_5) = 1$

4 Design Issues Regarding Algorithms Based on Niceness

In this section we discuss issues regarding optimal offline and online adaptive algorithms for the difficulty measures proposed in the last section under different buffer models. Any online algorithm in this problem domain has to deal with two important issues. First, it has to find a suitable cache replacement strategy

to manage the high-speed cache. Second, it has to perform buffer management under different buffer models. For an offline optimal algorithm the problem is just buffer management as it has the absolute knowledge of the future and hence is capable of taking optimal caching decisions.

4.1 Cache Replacement Strategies

Total number of misses made by an online algorithm is highly dependent on the cache replacement strategy it employs. There are many such strategies available, e.g. Least-Recently-Used (LRU), First-In-First-Out (FIFO), Flush-When-Full (FWF) etc., which have been heavily studied in networks and operating systems literature over the years. It is empirically known that LRU and its variants perform best in such cases [1]. Recently similar result is supported in theory using cooperative analysis [3, 8].

4.2 Buffer Management

Common buffer models available for use in routers include FIFO non-preemptive model, FIFO preemptive model and bounded delay model (see Section 2). Restrictions in those models make some operations invalid and hence we assume that some measures that we have proposed are not applicable or at least not best suited for specific models. For example, in both FIFO-based models its not possible to change the order of the packets once they have been accepted. Hence Exc may not be the best measure of niceness in those models. Similarly in the FIFO non-preemptive model, preemption or dropping of packets is not possible which contradicts the idea of Rem. Our intuition is that may be these measures are still applicable but their accuracy in these models need critical examination.

4.3 Conflicting Goals

Every router has a packet scheduler that plays the important role of determining the order of packets belonging to a particular flow to effectively ensure that the QoS service requirements are met. What this scheduler tries to do is that it ensures each of the client is enjoying a proportional bandwidth. To ensure that, it interleaves packets from different clients and which in effect creates disorder i.e. makes sequence less nice. If we employ an algorithm in the scheduler to fix this, the result will be deterioration or even failure QoS guarantee. But if we let it through into the buffer we may not be able to fix that because of restrictions imposed by buffer models. Conflict between these two issues requires serious attention.

4.4 Bursty Traffic and Locality of Reference

One very important property of internet traffic is the presence of bursty data. Most of the times there is high locality of reference among packet destinations

as requests from users tend to be targeted toward closely located servers. We believe an adaptive online algorithm should be able to take full advantage of this property. Presence of a burst can easily be identified from the values of $MaxDis$ and $SumDis$.

5 Conclusions and Future Work

In this paper we have proposed the idea of niceness of a packet sequence and established five novel difficulty measures to quantify this property of any sequence. Even though we have not managed to find optimal algorithms for the measures, we have shed light on different design issues regarding such algorithms. But perhaps the most interesting contribution of this work is the adaptation of adaptive analysis, largely studied in sorting problems, in network related problem domain. We believe more research works will stem from our introductory contribution.

A short list of open problems include -

1. Creating optimal algorithms for the difficulty measures discussed in this paper, specially for $MaxDis$ and $SumDis$.
2. Finding a partial ordering among the proposed difficulty measures as present in sorting problem due to Petersson and Moffat [14]. From the definitions of the measures of niceness we can observe that $MaxDis$ and $SumDis$ are somewhat similar to $Runs$ with cache-consciousness. Discovering such relationships can be an interesting research topic.
3. Analyzing the problem using cooperative analysis [3, 8].
4. Critical examination of accuracy and feasibility of using particular measure of niceness in different buffer models.

Finally, some simple questions resonate more than all these. What is the best possible use of this knowledge of niceness? Is it just for theoretical analysis or should it be used to take decisions whether or not to rearrange packets to make a sequence nicer? If yes, how will that affect the nagging issues like congestion control, flow control and QoS of the network?

Our future research seeks answers to all these questions.

References

[1] Peter B. Galvin Abraham Silberschatz and Gerg Gagne. *Operating System Concepts*. John Wiley & Sons, 2002.

[2] N. Andelman, Y. Mansour, and A. Zhu. Competitive queueing policies for QoS switches. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms(SODA'03)*, pages 761–770, 2003.

[3] S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. On the separation and equivalence of paging strategies. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*, 2007.

[4] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[5] W.H. Burge. Sorting, trees, and measures of order. *Information and Computation/Information and Control*, 1(3):181–197, 1958.

[6] Gene Cheung and Steven McCanne. Optimal routing table design for IP address lookups under memory constraints. In *INFOCOM '99*, volume 3, pages 1437–1444, 1999.

[7] T. Chiueh and P. Pradhan. High-performance IP routing table lookup using cpu caching. In *INFOCOM '99*, pages 1421–1428, 1999.

[8] R. Dorrigiv and A. López-Ortiz. Adaptive analysis of on-line algorithms. In *Robot Navigation*, 2007.

[9] Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992.

[10] Alexander Kesselman, Zvi Lotker, Yishay Mansour, Boaz Patt-Shamir, Baruch Schieber, and Maxim Sviridenko. Buffer overflow management in QoS switches. In *Proceedings of ACM Symposium on Theory of Computing(STOC'01)*, pages 520–529, 2001.

[11] Alexander Kesselman, Yishay Mansour, and Rob Van Stee. Improved competitive guarantees for QoS buffering. In *Proceedings of the 11th Annual Europian Symposium on Algorihtms(ESA'03)*, pages 361–372, 2003.

[12] Andreas Moestedt and Peter Sjodin. IP address lookup in hardware for high-speed routing. In *Proceedings of IEEE Hot Interconnects 6 Symposium*, pages 31–39, August 1998.

[13] Xiaojun Nie, David J. Wilson, Jerome Cornet, Gerard Damm, and Yiqiang Zhao. IP address lookup using a dynamic hash function. In *Proceedings of the 18th Annual Canadian Conference on Electrical and Computer Engineering*, pages 1642–1647, 2005.

[14] Ola Petersson and Alistair Moffat. A framework for adaptive sorting. *Discrete Applied Mathematics*, 59:153–179, 1995.

[15] M.A. Ruiz-Sanchez, E.W. Biersack, and W. Dabbous. Survey and taxonomy of IP address lookup algorithms. *IEEE Network*, 15(2):8–23, 2001.

[16] Sartaj Sahni and Kun Suk Kim. An O(log n) dynamic router-table design. *IEEE Transactions on Computers*, 53(3):351–363, 2004.

[17] Kari Seppnen. Novel IP address lookup algorithm for inexpensive hardware implementation. In *WSEAS Transactions on Communications*, pages 76–84, 2002.