

Homework Assignment 2

Handout 2

Due: In 2 weeks

Directions: You may collaborate but write up solutions yourself. No late HW.

1. **Converting a bridge into a monitor:** Alyssa P. Hacker is working for DEC and has learnt of the spiffy bridge invented at DEC. Alyssa decides to convert the bridge into a LAN traffic monitor that will passively listen to a LAN and produce statistics about traffic patterns. The marketing person tells her that she needs to monitor traffic between arbitrary source-destination pairs (i.e., how many packets were sent from say station A to station B .) Thus for every active source-destination pair like A, B , Alyssa must keep a variable $P_{A,B}$ that measures this quantity. When a packet is sent from A to B , the monitor (listening in promiscuous mode on the LAN) will pick up a copy of the packet. If the source is A and destination is B , the monitor should increment $P_{A,B}$. The problem is do this in 64 usec, the minimum interpacket time on the Ethernet. The bottleneck is to lookup the state $P_{A,B}$ associated with a pair of 48 bit addresses A, B .

Fortunately, the bridge hardware has a lookup engine that can lookup a *single* 48 address in 1.4 usec. The problem that Alyssa must solve is to apply Principle 8 to use a lookup engine that can lookup single addresses into one that can lookup address *pairs*. Can you help Alyssa design a new product and get a raise.

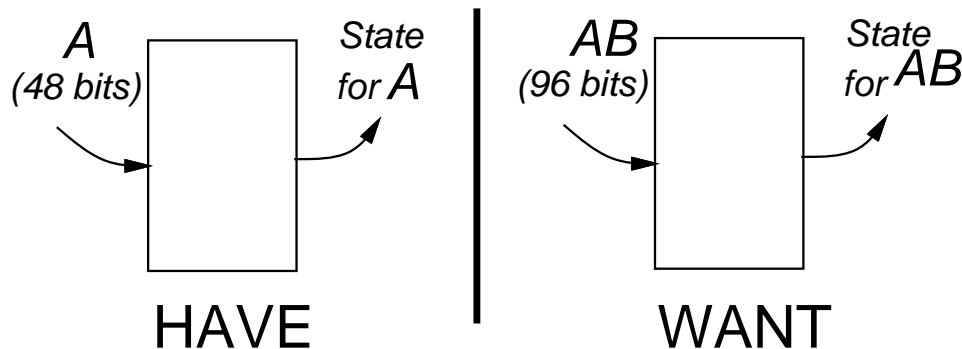


Figure 1:

2. **Incrementally reading a Large Database** Suppose a user reads a large database continuously (e.g., a web page). The web page can change and the reader only wants the incremental updates since he last read the database. Thus in the Fig 4, if a reader last read at 2 pm and reads again at 6 pm, he only wants the differences: Coke to Pepsi, and Wheaties to Cheerios, If, on the other hand a different user reads at 3 pm and then at 6 pm, he only wants the difference: Wheaties to Cheerios. The problem is to find a way for the database to efficiently perform such incremental queries (i.e., all changes since 2 pm). It is unreasonable for the database to remember what each user has previously read since there may be millions of users. Nevertheless, consider the use of Principle 12 (add redundant state) to help optimize incremental queries. Describe the client request format and server request processing.

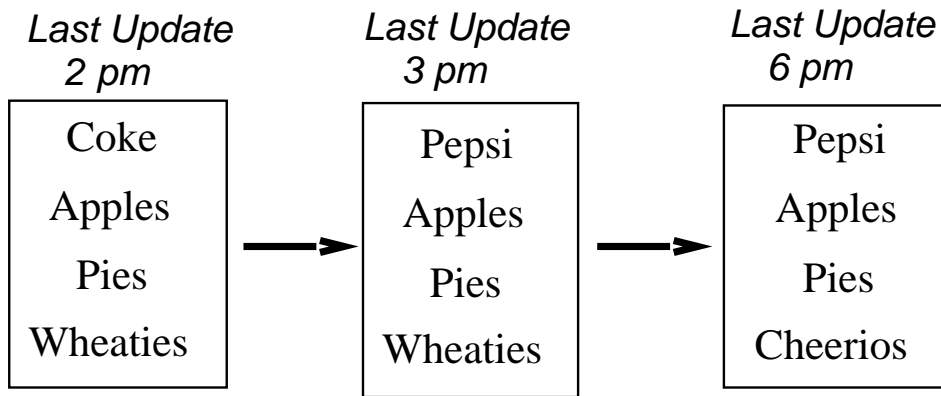


Figure 2:

- **3. Caching Prefixes:** Suppose we have the prefixes 10^* , 100^* , and 1001^* . Hugh Hopeful would like to cache prefixes instead of entire 32 bit addresses. Hugh's scheme keeps a set of prefixes in the cache (fast memory), in addition to the complete set of prefixes in slow memory. Hugh's scheme first does a best matching prefix search in the cache; if a matching prefix is found, the next hop of the prefix is used. If no matching prefix is found, a best matching prefix search is done for the entire database and the resulting prefix is cached. Periodically, prefixes that have not been matched for a while are flushed from the cache.

Alyssa P. Hacker quickly gives Hugh a counterexample to show him that his scheme is flawed, and that caching prefixes is tricky (if not impossible). Can you?

- **4. Encoding prefixes in a constant length:** We said in the text that encoding prefixes like 10^* , 100^* , and 1000^* in a fixed length could not be done by padding prefixes with zeroes. It clearly can be done by padding with zeroes and adding an encoding of the prefix length. We want to study a more efficient method.

- How many possible prefixes on 32 bits can there be?
- Show how to encode all such prefixes using a fixed length of 33 bits. Make sure that 10^* , 100^* , and 1000^* encode to different values.
- Can you use this fixed length encoding of prefixes to have the multiple hash tables used in Binary Search on Hash Tables be packed into a single hash table? Why might this help in decreasing the chances of hash collisions for a given memory size?

- **5. Motivating the iSLIP pointer increment rule:** The following is one unfairness scenario if pointers in iSLIP are incremented incorrectly. For example, suppose that input port A always has traffic to output ports 1, 2, and 3, whose grant pointers are initialized to A . Suppose also that input ports B and C also always have traffic to 2. Thus initially A , B , and C all grant to 1 who chooses A . In the second iteration, since input port B has traffic to 2, 2 and B are matched.

- Suppose 2 increments its grant pointer to C based on this second iteration match. Between which port pairs can traffic be continually starved if this scenario persists?
- How does iSLIP prevent this scenario?