Distributed Counting along Lossy Paths without Feedback*

Vitalii Demianiuk[1,2], Sergey Gorinsky[1], Sergey Nikolenko[2,3], and Kirill Kogan[1]

[1] IMDEA Networks Institute
{vitalii.demianiuk, sergey.gorinsky, kirill.kogan}@imdea.org
[2] Steklov Institute of Mathematics at St. Petersburg
sergey@logic.pdmi.ras.ru
[3] Neuromation OU, Tallinn, 10111 Estonia

Abstract. Network devices need packet counters for a variety of applications. For a large number of concurrent flows, on-chip memories can be too small to support a separate counter per flow. While a single network element might struggle to implement flow accounting on its own, in this work we study alternatives leveraging underutilized resources elsewhere in the network and implement flow accounting on multiple network devices. This paper takes the first step towards understanding the design principles for robust network-wide accounting with lossy unidirectional channels without feedback.

1 Background and problem settings

Scalability chalenges. Per-flow packet counting in network devices is a crucial functionality in network operation, management, and accounting [1–5]. When a packet arrives to a network device that needs to perform per-flow counting, fast-path processing in the device determines the flow associated with the packet and increments the flow counter. Packet counting is traditionally done in a single network device, but it becomes prohibitively expensive—if at all feasible—to maintain per-flow counters as the number of flows and link speeds grow. Proposed solutions either sacrifice counting accuracy or adopt complex memory architectures. Our paper explores an alternative of network-wide packet counting.

Horizontal vs. vertical counter split. To count packets in a flow, network devices involved in distributed accounting have to lie on the flow's path; at the very least, each flow traverses two switches, its source and destination. Assuming reliable communication, a flow counter can be allocated in any one of the network elements in its path; we call this representation a *horizontal split*. In this work, we relax the constraints on interconnecting links as much as possible, assuming an unreliable unidirectional communication without feedback. Then horizontal

* This work was partially supported by a grant from the Cisco University Research Program Fund, an advised fund of Silicon Valley Community Foundation and by the Regional Government of Madrid on Cloud4BigData grant S2013/ICE-2894.

split becomes infeasible since packets can be dropped before they ever reach the counter, so some part of the counter should be allocated on the source network element. We split the counter into two chunks for source and destination switches; we call this a *vertical split*. Since any given switch stores only a fraction of the counter, it needs less memory to support counting the same number of flows. With our relaxed assumptions on interconnecting links, it is crucial to make distributed execution robust to packet reordering and loss; moreover, we also assume that each packet is allowed to carry only a few bits, which can significantly complicate the operation of distributed counters.

Problem statement. An asynchronous network delivers a flow f of packets from source switch S to destination switch D, where p_i is an i-th packet of f at S, $i = 0, \ldots, |f| - 1$. Our goal is to compute $|f|$, i.e., number of packets received by ingress switch S from a flow f. Switches S, D maintain partial counter states of at most n bits. The proposed distributed counter representations should be able to exactly reconstruct the counter value after a flow terminates; during a flow's lifetime, counter values returned by queries should not decrease in time and cannot exceed the actual counter value. We study the problem of correctly executing counters under space constraints despite potential packet reordering and loss. We assume that a packet can be prepended by at most t bits.

2 Proposed method

Splitting a counter between source and destination switches is a ubiquitous model since it does not make any assumptions about routing. Robustness of the distributed accounting to packet reordering and loss certainly has its fundamental limits, e.g., the loss of *all* packets in the network disables stateful communication. To characterize the limits of achievable robustness, we represent delivery disruptions with two parameters:

- *reordering parameter* R is the maximal extent of packet reordering, i.e., the destination switch can receive packet p_j before packet p_i only if $j \leq i + R$;
- *loss parameter* L is the length of a maximal interval of consecutive losses, i.e., the destination switch receives at least one packet from any range $p_i, \ldots, p_{i+L}$.

To overcome delivery disruptions, both S and D use t bits of the n-bit counter chunk as *synch bits* to synchronize the two counter chunks. These t synch bits are the most significant bits in S's chunk c_1, least significant in D's chunk c_2, and middle bits in flow f's merged two-chunk counter c, which counts up to 2^{2n-t}. Upon receiving a packet p, switch S records synch bits from its counter c_1 into packet header $h[p]$ and increments the n-bit counter. When p arrives to D, the latter computes the difference between packet header $h[p]$ and the t synch bits in counter c_2. If this difference is between 1 and 2^{t-1}, switch D adds it to c_2. Upon the completion of flow f, the controller managing accounting network infrastructure collects the c_1 and c_2 values from S and D to obtain $|f|$, i.e., the total number of flow f's packets received by switch S: the controller sets the n most

Algorithm 1: Two-switch counting.

```
procedure SourceUpdate(p)
    h[p] = c1 >> (n - t)
    c1 = (c1 + 1) mod 2^n
end procedure

procedure DestinationUpdate(p)
    diff := Difference(h[p], c2, t)
    If 1 <= diff <= 2^(t-1) then
        c2 := c2 + diff
end procedure

function Difference(a,b,t)
    δ := (a + 2^t - b mod 2^t) mod 2^t
    return δ
end function

procedure TotalCount(c1, c2)
    c := c2 << (n - t)
    c := c + Difference(c1, c, n)
end procedure
```

significant bits of counter c to c_2 and then adds to c the difference between c_1 and the n least significant bits of c. Algorithm 1 consists of the SourceUpdate, DestinationUpdate, and TotalCount procedures described above.

Theorem 1. *Algorithm 1 correctly counts up to 2^{2n-t} packets under the following conditions:*

$$L + R < 2^{n-1} \quad \text{and} \quad R \leq 2^{n-1} - 2^{n-t}. \tag{1}$$

Proof. To prove correctness, we have to show that each update of c_2 by switch D is correct, i.e., c_2 becomes equal to $i \gg (n-t)$ after D receives any packet that updates c_2 . We prove it by induction. When D receives its first packet, the packet's index is at most $L + R$ (this can happen if the first L packets of the flow are lost, and packet p_{L+R} arrives to D first, before $p_L, \ldots, p_{L+R-1}$). For this packet $h[p] \leq (L + R) \gg (n-t) \leq 2^{t-1}$, therefore, it correctly updates c_2.

For the induction step, suppose $c_2 = i \gg (n-t)$ after D processes packet p_i updating c_2 and consider the next arrival of packet p_j to D that updates c_2. Fig. 1 partitions the packet sequence at switch S into groups of 2^{n-t} consecutive packets that have the same synch bits. Let I and J denote the groups of p_i and p_j respectively. For $I < J \leq I + \left\lceil \frac{L+R+1}{2^{n-t}} \right\rceil$, Algorithm 1 correctly updates c_2 due to $\left\lceil \frac{L+R+1}{2^{n-t}} \right\rceil \leq 2^{t-1}$, i.e., the difference between packet header $h[p_j]$ and c_2 is at most 2^{t-1}. Since at least one packet from the considered sequence of groups arrives to D after p_i, no packet from a group later than $I + 2^t$ arrives first due to R. By definition of j, packet p_j does not belong to groups $I + 2^{t-1} + 1$ through $I + 2^t$ because D does not update c_2 for a packet of these groups. $J \leq I$ is impossible since conditions (1) imply that $J \geq I - \left\lceil \frac{R}{2^{n-t}} \right\rceil \geq I - \frac{2^{n-1} - 2^{n-t}}{2^{n-t}} > I - 2^{t-1}$, the difference between $h[p_j]$ and c_2's synch bits is either 0 or greater than 2^{t-1}, and Algorithm 1 appropriately does not change c_2. This establishes correctness for each update of c_2.

When flow f ends, c_1 contains the n least significant bits of $|f|$, and the index of the last packet that updates c_2 differs from $|f|$ by at most $L + R + 1$ packets. Since $L + R + 1 < 2^n$, Algorithm 1 accounts for all subsequent missing packets

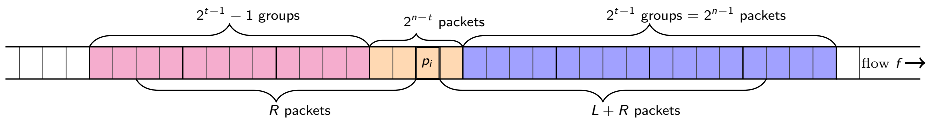

Fig. 1: Packet sequence in the proof of Theorem 1.

when TotalCount increases $c_2 \ll (n-t)$ by making its n least significant bits equal to c_1. Thus, counter c correctly computes $|f|$.

Parameter t represents a tradeoff between increasing R and decreasing L and $|f|$. Without packet reordering, when $R = 0$, Algorithm 1 correctly counts packets with loss of up to $L < 2^{n-1}$ consecutive packets. When $R > 2^{n-1}$, Algorithm 1 is never guaranteed to work correctly. Since c_2 never decreases, and $c_2 \ll (n-t)$ never exceeds the number of packets that have arrived to switch S, $c_2 \ll (n-t)$ can be used as a real-time lower bound on the number of packets arrived to S.

Algorithm 1 has attractive robustness in practical settings. For example, when a counter chunk contains 12 bits and uses 2 of them as synch bits, Algorithm 1 correctly counts up to 2^{22} packets despite the loss of up to 1023 consecutive packets and reordering stretch up to 1024 packets. Doubling the number of synch bits from 2 to 4 increases the tolerated reordering stretch to 1792 packets, reducing loss tolerance to 255 consecutive packets and decreasing supported flow size to 2^{20} packets.

3 Conclusion

In this work, we have studied distributed counter implementation under packet reordering and loss. The basic idea of our design is to exploit the state overlap between two communicating switches to maintain correctness of distributed counter state under network noise.

References

1. Lu, Y., Montanari, A., Prabhakar, B., Dharmapurikar, S., Kabbani, A.: Counter braids: a novel counter architecture for per-flow measurement. In: SIGMETRICS. pp. 121–132 (2008)
2. Ramabhadran, S., Varghese, G.: Efficient implementation of a statistics counter architecture. In: SIGMETRICS. pp. 261–271 (2003)
3. Shah, D., Iyer, S., Prabhakar, B., McKeown, N.: Analysis of a statistics counter architecture. In: HOTI. pp. 107–111 (2001)
4. Wang, N., Ho, K.H., Pavlou, G., Howarth, M.P.: An overview of routing optimization for internet traffic engineering. IEEE Communications Surveys and Tutorials **10**(1-4), 36–56 (2008)
5. Zhao, Q., Xu, J.J., Liu, Z.: Design of a novel statistics counter architecture with optimal space and time efficiency. In: SIGMETRICS/Performance. pp. 323–334 (2006)