Optimizing the IP router update process with traffic-driven updates

Wouter Tavernier*, Dimitri Papadimitriou†, Didier Colle*, Mario Pickavet* and Piet Demeester*
* Ghent University, IBCN - IBBT, Department of Information Technology
Gaston Crommenlaan 8 bus 201, B-9050 Gent, Belgium
Email: {wouter.tavernier, didier.colle, mario.pickavet, piet.demeester}@intec.ugent.be
† Alcatel-Lucent Bell
Copernicuslaan 50, B-2018 Antwerpen , Belgium
Email: dimitri.papadimitriou@alcatel-lucent.be

***Abstract*—The update process in an IP router after a failure is detected, is a complex process involving more than just recalculating shortest paths. When statistics related to forwarded traffic are not taken into account, which is generally the case, we show that packet loss can be significantly higher than is strictly needed. In this paper we present a combined control and traffic-driven routing table update scheme which minimizes the packet loss during the switchover, based on an efficient use and planning of the central router process quantum. We discuss the parameters of the algorithm and show its value in a simulation environment.**

I. Introduction

During the last decade, network and internet usage has grown to proportions that haven't been seen before. Numerous network services such as IPTV or VoIP using high speed wired or wireless internet access, are now accessible for the residential market. At the same time business services such as Virtual Private Networks or Grids have a severe impact on the network requirements. It is clear that we become more and more dependent on networked services, and actually more dependent on the network itself. These result into very tight Service Level Agreements (SLAs), where five nine's availability becomes the common.

Availability depends on the maintainability and reliability of the networked system and its components. The above availability requirement results thus in reliability requirements. The latter have resulted in numerous protection and rerouting techniques at different network layers to ensure rapid detection of, and switchover during failure events within the network. Think p.e. about hardware specific methods for Bidirectional Forwarding Detection (BFD) for failure detection, protection mechanisms at the optical layer, or fast rerouting schemes either at the Multi-Protocol Label Switching (MPLS) or IP level (e.g. IP Fast Re-Routing (FRR)). Whereas proposed approaches often focus on the time performance of the recovery process, they seldomly take into account the quality of the recovery process itself (that is measured in packet loss). This is related to the fact that in general, these techniques are agnostic to the properties of the specific traffic flows that are forwarded along the corresponding paths.

Nevertheless, a plethora of techniques exists to gather statistics about traffic flows crossing routers (see section III-A). This statistical data is however seldomly used in the process of a router updating its routing and forwarding entries (*the IP router update process*) as triggered by a failure or network change. Given the fact that the update process can take up to 1 second for routers in large IP networks (see [1]), switching-over an affected, high bit-rate traffic flow only after let's say 400 ms, can have a high cost in terms of packet loss.

Therefore the goal of this paper is to improve the quality of the recovery process, by i) investigating the IP router update process, and ii) evaluating update mechanisms using traffic flow data such as to decrease packet loss resulting from the forwarding table update process as triggered by a failure or network topology change.

The rest of the paper is organized as follows. Section II describes and formalizes the IP router update process and its characteristics. The next section (section III) discusses possibilities to enhance the process in terms of reduced packet loss, and experimental validations of the formulated strategies are described in section IV. Finally the paper is ended with notes on future work and a conclusion in section VI.

II. The IP router update process

Larger IP networks such as the Internet, are partitioned into ASes (Autonomous Systems) independently administrated by Internet Service Providers (ISP) and structured in "tier levels". Every AS comprises a collection of routers. These are responsible for transferring data units (IP datagrams) across the network infrastructure, and consist of a routing engine and a forwarding engine.

When an incoming IP datagram (or packet) arrives at the incoming interface of an IP router, the router looks for an entry in its Forwarding Information Base (FIB), performing a longest prefix match on the destination IP address of the incoming packet, to forward the traffic towards its next hop. The FIB can either be manually configured or can be populated by a routing protocol. Because the forwarding decision is taken independently at each hop, the forwarding process is *connectionless*.

Peering routing engines exchange information using routing protocols. These protocols ensure that routing information is distributed allover the network (or the AS), such that

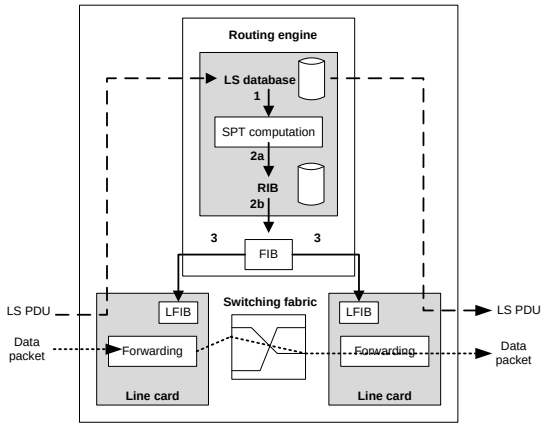


Fig. 1. The router update process

individual routers can maintain a consistent and up-to-date full view of the non-local network topology or loop-free AS-path to destination prefixes and react accordingly in case of a topological failure. In the Internet, two main levels of routing can be distinguished: intra-domain routing level determined by interior gateway protocols such as the Open Shortest Path First (OSPF, [2]) and IS-IS link-state protocols; and the inter-domain routing level implemented by the Border Gateway Protocol (BGP, [3]), a policy-based AS-path vector routing protocol which distributes routing information originated by routers belonging to different AS.

A. The LS router update process

For operational and scaling reasons, our investigation focuses on Link State (LS) protocols. Routers running LS protocols, flood LS PDUs (LS Protocol Data Units) over the network. These packets contain information about the local links and MA (multi-access) networks a router is connected to. All received LS PDUs are collected into a database (*the LS database*) which allows a router to have a view on the network link topology and to calculate (shortest) paths towards different destinations (IP addresses) or network parts (IP network prefixes). The LS database is updated by either detecting a local connectivity change (e.g. failing link or interface), or by receiving an LS PDU from a peering router. Two types of LS PDUs can be received:

- *LS Refresh*: A node in the network has sent a refresh of its status, involving no changes in the network.
- *LS Update*: A node in the network has detected a state change in its link connectivity to an adjacent node or network (addition or removal).

Only the second option is of interest for us, as it is the direct trigger for the router update process which we intend to investigate. The resulting update process can modeled in three steps (see figure 1):

1) Re-computation of the *shortest path tree* (1), based on the updated LS database.
2) Update of the *central Routing Information Base (RIB)* and the *central Forwarding Information Base (FIB)*(2a and 2b), based on the shortest path computation.
3) *Distribution of central FIB* towards the line cards' local FIB (LFIB) (3).

The recomputation of the shortest path tree is usually optimized to be recalculated in its entirety and takes about 30 to 50 μs per destination prefix. Optimizations can be done using incremental SPF (iSPF) calculation schemes (see [4] and [5]). The second step consists of updating the central RIB and FIB, using the calculated shortest paths. This uses about 50 to 100 μs per destination prefix (see also [1]). Typically this step happens in (pseudo-)parallel with step 3, which is about distributing the central FIB entries towards the line cards' LFIB. Running step 2 and 3 in (pseudo-)parallel, means that they both use the central CPU in interleaved timeslots, swapping between both processes for updating and distribution. This process can be compared to the usual process scheduling in time-sharing OSes such as Linux (commercial routers make use of a hard real time OS). The consecutive time the central CPU is spending to a task of central RIB/FIB updating or linecard distribution is determined by the used quantum of the swapping process. The quantum time in can typically be configured between 10 and 800 ms (for Linux, see [6]). Values in conformance with these were found in [1].

In practice the update process consists of a series of update-distribution batches, where in a first quantum a fixed set of prefixes are updated towards the central RIB/FIB, followed by a quantum where the same set of prefixes is distributed towards the LFIBs. By default, the cardinality of the set (the number of prefixes that are updated or distributed during a batch) is fixed during the update process. This is shown in figure 2, and will be further quantified in the next section.

B. Formalized model of the router update process

The process described in the previous section can be formalized using the following naming conventions and modeling assumptions (see figure 1):

- The set of *affected*[1] *traffic flows*[2]: $F_n = \{f_1, ..., f_n\}$ (for example in MB/s).
- The ordered *associated traffic flow rate* $bw(f_i) : F_n \rightarrow R$ of the affected flows (assumed to be in MB/s).
- The *update time* t_u: the time needed to update one prefix in the central RIB/FIB.
- The *distribution time* t_d: the time needed to distribute a prefix from the central FIB towards the linecards.
- The number x_u of *prefixes that are updated/distributed* in one batch
- The *update quantum* resulting from updating x_u prefixes is the time $t_{qu} = x_u.t_u$.
- The *distribution quantum* resulting from distributing x_u prefixes from the central RIB/FIB towards the line cards is the time $t_{qd} = x_u.t_d$.
- The *swapping time/cost* t_s between interleaved quantums.

[1]Flows are 'affected' if their routing is influenced by the received LS Update

[2]A traffic flow is a function $f : V \times V \rightarrow \mathbb{R}$ which attaches a load to every edge of the network. The function is constrained by a capacity constraint, a flow symmetry constraint and a flow conservation constraint. (see [7]).

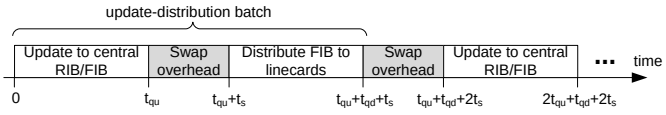


Fig. 2. Timeline of the router update process

The introduced terminology can be illlustrated by the following example. Let's assume that a network failure results into 5000 traffic flows that need to be recovered (F_{5000}). If we use the following configuration values $t_u = 100$ μs, $t_d = 90$ μs, $x_u = 500$, $t_s = 1000$ μs, this results into 10 update-distribution batches, each taking $500 \times 100 + 1000 + 500 \times 90 = 101000$ μs. Taking into account the additional intermediate swapping times between the batches, this results into $10 \times 101000 + 9 \times 1000 = 1019000$ μs, or 1.019 s to recover all flows.

C. Recovery time and recovered traffic

The recovery of a set of traffic flows (i.e. traffic directed towards a set of destination prefixes comprised as part of different RIB/FIB entries to be updated) is completed when the last corresponding RIB/FIB entry has been updated and distributed from the central RIB/FIB to the linecard. The corresponding point in time where the entire set of affected traffic flows is recovered, is referred to as the *total recovery time*. A single traffic flow can be considered as recovered, once its corresponding RIB/FIB entry has been both updated and distributed from the central RIB/FIB to the linecards.

The recovery time for a single traffic flow, given a constant x_u value, is thus dependent on the update-and-distribution batch b_i comprised in it. Only when the corresponding batch (and all previous batches) are completed, the flow has been recovered. Given n flows and the fact that only x_u flows can be updated in one batch, the batch number b_i is determined by $b_i = \lceil i/x_u \rceil$.

A traffic flow is recovered after all previous batches have been updated and distributed, having a swapping overhead on every load of a process. This results into the formula for the recovery time of flow f_i.

$$r(f_i) = b_i.(x_u(t_q + t_d) + 2t_s) - t_s$$

Applying the formula on the earlier example, we find that f_1 to f_{500} are recovered after 101 ms, while the recovery time of f_{4501} to f_{5000}, as well as the total recovery time is 1.019 s.

D. Packet loss

As long as an affected traffic flow hasn't been recovered upon a failure, packet loss occurs. The loss is proportional to the throughput of the traffic flow during the even of the update. This means that the total packet loss during the switchover operation of all traffic flows is proportional to the product of the recovery time and their corresponding average flow rate during the router update process[3]. In other terms, the drop in average capacity consumed by each outgoing flow (i.e. average bandwidth consumption), can be equated to losses:

$$loss(F_n) = \sum_{i=1}^{n} r(f_i).bw(f_i)$$

For a small example on F_4 with $bw(f_1) = 5, bw(f_2) = 2, bw(f_3) = 4, bw(f_4) = 1$ (in MB/s), and $t_u = 100$ μs, $t_d = 90$ μs, $x_u = 2$, $t_s = 70$ μs, this results into a packet loss of $260 \times 10 + 590 \times 5 = 5550$ KB/s.

The above formula illustrates that the order in which traffic flows are updated, can be of uttermost importance for minimizing the associated packet loss of the update operation. Randomly updating RIB/FIB entries, as is usually the case, can result into the situation where higher bit rate flows need to wait on others before they are updated, resulting in high cumulative loss (for example f_3 is proportionally bigger than f_2, but however needs to wait until $f_1 \ldots f_2$ are updated).

III. Optimization strategies

Given the formalized entities of previous sections, our goal is to optimize the central RIB/FIB to LFIB update process in such a way that the packet loss is minimized using a reactive strategy. This means that from the moment a network topology change is detected because of an incoming LS PDU, specific actions are taken to minimize the packet loss. The strategies we will suggest, account for available capacity on recovery link(s) even if this does not allow to infer whether next-hops are themselves able to cope with capacity increase on their outgoing links (towards next next-hop).

A. Traffic flow monitoring

As motivated previously, gathering traffic flow average rate statistics can be of high value during the event of router updates. The following methods exist to gather data such as to estimate $bw(f_i)$ on the moment of the update:

- *Online statistical counters* measure aspects of transitting traffic in a router using counters, for example the number of packets per destination prefix or used packet size distribution curves (for example see [8]).
- *Traffic sampling* by means of samplers. Instead of counting certain traffic characteristics, unmodified traffic is captured for some time interval. This sample is then used to derive certain characteristics, for example done in Netflow-alike settings (e.g. ([9]).

As we will not focus on the metrology problem, we refer to given citations for more detailed information. However we assume the following running conditions. We assume the monitoring system to be *passive* in the sense that no addtional traffic is inserted into the network for the sake of monitoring. We expect the monitoring system to be *independent* and

[3]This assumes that we exactly know the average bandwidth used during the router update. In practice the bandwidth is an approximation using monitoring techniques, as discussed in III-A. At the same time this approximation is the estimation of the needed (available) capacity on the alternate set of one or more link(s) towards the destination.

distributed, meaning that we do not assume any form of centralized monitoring entity. Next, it is our assumption that the monitoring system is based on an *open loop*, meaning that the counts or estimations do not lead to actions modifying the corresponding values. Last, but not least, for the current study we depend on the assumption of *temporal locality or persistence*, as we assume that the traffic pattern as measured during a monitoring time frame will likely be the same as during the time frame of the router update, given that the distance between the two time frames is not too large. This is related to the concept of Longe-range Dependency (LRD). Further information can be found in [10].

B. Traffic flow sorting

Once average rate data about the traffic flows is known, the most obvious way of using this information is to sort F_n in descending order of average flow rate. The resulting set $O_n = \{o_1, ..., o_n\}$ can now be defined as follows:

$$\forall o_i : \exists j \in \{1, \ldots, n\} : o_i = f_j$$
$$i > j \Rightarrow bw(o_i) \geq bw(o_j)$$

Upon the RIB information update, the corresponding loss can thus be deduced and used as a means to minimize average traffic losses during the central RIB/FIB to LFIB update process. It is now easy to see or to prove that $loss(O_n) \leq loss(F_n)$. Though the gain of this is obvious, the computational cost of sorting can be a blocking issue as the router update process needs to happen in a timeframe of hundreds of ms (the typical average fastest computational complexity is O(n log n)). However two additional observations can be made for this: at first, average flow rate sorting per destination prefix can happen in prior to the update itself (as continuous process during traffic monitoring), second, perfect sorting is not necesarrily needed because the most important gain is reached when the most important traffic flows are put in the beginning of the update batch.

C. Optimizing the number of prefixes in a quantum

The previous sections (i.e. II-C and II-D) made clear that the effect of a fixed x_u (and related quantums) during the router update process can lead to a higher impact than expected. The x_u acts as a sort of a bin packer for the updates/distributions to be executed. Using a given F_n (being sorted or not), this section evaluates the possible gain of allowing x_u to be dynamically determined per update-distribution batch and optimized depending on the specific character of F_n.

The goal of this configuration strategy is to minimize the function $loss(F_n)$ by finding the appropriate sequence of sets of prefixes. Instead of a single fixed x_u this results into a sequence $X_u = (x_{u_1}, x_{u_2}, \ldots, x_{u_l})$[4] of the number of prefixes to be updated during every adjustable quantum. Associated to these sequence is the sequence of update-distribution batches $B_u = (batch_{u1}, \ldots, batch_{ul})$. This means that every f_i is contained in an update-distribution batch of B_u. Unfortunately

[4] X_u is actually an integer partition of n. This means that $\sum_{i=1}^{l} x_{ui} = n$.

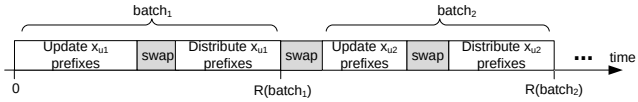


Fig. 3. Timeline for router updates having a variable x_u

the smaller the x_{u_i}'s are chosen, the more swapping time is sneaking into the total recovery time (see section II-C) of the router update, possibly resulting into additional loss. We have thus to consider an adaptive x_u setting that minimizes the packet loss by keeping the total recovery time as small as possible (update-distribution batches should be as large as possible to reduce swapping time). However, one should also avoid that the recovery of certain traffic flows is delayed longer than needed, because the longer the update-distribution batches are, the longer prefixes need to wait until their update is both updated and distributed, and thus recovered.

The associated recovery time of a flow f_i is again determined by the $batch_k$ to which f_i is attached. However, contrary to the definition in II-C, the length of update-distribution batches is not fixed anymore. Therefore, one needs to find first in which batch a flow f_i is contained:

$$batch(f_i) = batch_j$$
$$\Updownarrow$$
$$\sum_{r=1}^{j} x_{ur} \geq i \wedge \forall (s \text{ with } \sum_{r=1}^{s} x_{ur} \geq i) : s \geq j$$

The recovery time of the update-distribution batch related to prefix bin x_{uk} is given by the following formula (see figure 3).

$$r(batch_k) = (t_u + t_d) \sum_{j=1}^{k} (x_{uk}) + (2k-1)t_s$$

1) Base scenario: Finding an optimal sequence of x_u's in realtime with limited global F_n information available is not an easy task. To obtian a basic understanding of this process, we first focus on the most simple case, the general case of $F_2 = \{f_1, f_2\}$. For choosing the appropriate x_u, we have now two options: $x_u = 1$ or $x_u = 2$. For this case, we can simply choose the batch size which results into the smallest loss as determined by the following decision function $d(F_2) = loss_{x_q=1}(F_2) - loss_{x_q=2}(F_2)$. If $d(F_2) < 0$, we choose $x_u = 1$, else we choose $x_u = 2$. Let's now write out the function $d(F_2)$:

$$\begin{aligned} d(F_2) =& (t_u + t_d + t_s) bw(f_1) \\ &+ (2t_u + 2t_d + 3t_s) bw(f_2) \\ &- (2t_u + 2t_d + t_s)(bw(f_1) + bw(f_2)) \\ =& 2t_s bw(f_2) - (t_u + t_d) bw(f_1) \end{aligned}$$

The above formula illustrates that the loss of the additional swapping times of $x_u = 1$ having an effect on the second flow

needs to be compared to the additional loss of the first flow needing to wait on the second, for $x_q = 2$. We can interprete as follows: if the second flow is big enough compared to the first flow (i.e. $bw(f_2) > \frac{t_u+t_d}{2t_s} bw(f_1)$, the *base criterium*), extending the current batch is valuable.

2) Extended scenario: Unfortunately a strategy to compare the loss of all possible choices for an arbitrary F_n cannot be easily found. Nevertheless, following the interpretation of the base scenario, we can observe that in the middle of the ordered process of updating F_n, with our current batch containing a set of flows to be updated $b_{current} = (f_i, \ldots, f_{i+s})$[5], we have two options:

1) Extend the current batch with the next flow f_{i+s+1} (*extension*)
2) Finish the current batch and put the next flow into a new update-distribution batch (*splitting*)

In the base scenario we compared the loss of letting f_1 wait on f_2, versus the loss caused by addtional swapping times on f_2. Similarly, we can again compare the additional cost of extension vs. the additional cost of finishing the update-distribution batch to guide us into the decision above.

The extension cost ec_{is} can be formulated as follows:

$$\begin{aligned} ec_{is} &= (t_u + t_d) bw(f_{i+s}) + \ldots + (s+1)(t_u + t_d) bw(f_i) \\ &= (t_u + t_d) \sum_{t=i}^{i+s} (i + s - t + 1) bw(f_t) \end{aligned}$$

The formula above expresses the fact that, by extending the current batch, the recovery time of every flow in the current batch will result into an additional delay compared to the minimal delay it can experience (compared to the earliest recovery time[6]). That additional delay when multiplied with the associated bandwidth allows to deduce the additional loss caused by the update-distribution batch extension. For example, the recovery of the first flow f_i in the given batch was already delayed with s update-distribute batches (as it was not directly distributed but put in the same batch of s next flows), and by adding an additional element (extending the batch), this operation will delay it with an additional update-distribution batch. On the contrary, the recovery of the last flow of the current batch will only be delayed with one update-distribution batch in case of extending the current batch.

Finishing the current batch on the other hand, also has an associated cost, as it will introduce additional delay for the coming flows, resulting from the additional swapping cost. This termination condition can be formulated as follows:

$$fin_{i+s} = 2t_s \sum_{t=i+s+1}^{n} bw(f_{i+s+1})$$

Our configuration strategy now consists in identifying the action with the least associated cost. The actions being: extending the current batch if $ec_{is} < fin_{i+s}$, or finishing the current batch in the other case. This action scheme can be applied recursively until the update of F_n is finished.

3) Example: The optimization strategy for F_n with $n > 2$ can for example be applied on F_4. Let's assume our current update batch contains f_1, f_2 and f_3. Following the above optimization strategy, this means that extension for f_2 and f_3 had lower cost than splitting, i.e. $ec_{11} < fin_2$ and $ec_{12} < fin_3$.

We can now either again extend the batch with f_4 or split the batch and put f_4 in a new update-distribution batch. Figure 4 shows the difference between both decisions: extension delays the recovery time for f_1, f_2 and f_3, while a split delays the recovery time for f_4 (and all flows following f_4). Table I quantitatively shows the recovery times for all flows in both scenario's and compares them with the earliest recovery time possible per flow . The decision $ec_{13} < fin_4$ will compare the difference in recovery time multiplied with the associated bandwidth to compare packet loss of both options.

IV. Experimental results

The goal of this section is to evaluate the different router update process models that were described in the previous sections. For this a simulation environment is used. The section is structured as follows: first the environment and the methodology will be described, second a set of three experiments is discussed.

A. Environment and methodology

Because no existing simulation environment was available for evaluating the local router update process, a custom C++ framework was developed in which classes were made for generating different sets of F_n, and for simulating the router update process and its traffic-driven optimizations as modeled.

Simulations were executed on 5000 traffic flows (F_{5000}). Each traffic flow in the generated F_{5000}-set was linked to a traffic rate between 1 KB/s and 100 MB/s ($bw(f_i)$), generated using a bounded pareto distribution. To obtain representative results, all experiments were repeated 100 times, and the resulting values were averaged. The following parameters were fixed: $t_u = 100$ μs, $t_d = 100$ μs.

B. Results

1) Packet loss vs. (minimum) batch size: As there is little known on the "ideal update-distribution batch size", the goal of the first experiment is to evaluate the packet loss of the router update process over different sizes of the update-distribution batch (x_u is varied in the X-axis between 0 and 5000). As described earlier, the batch size determines the number of prefixes that are updated towards the central RIB/FIB (during the update quantum) and subsequently are distributed towards the line cards (during the subsequent distribution quantum). This exercise was done using 4 variants of the router update process, using a value of 5 ms for the swapping time t_s:

1) The default update process with *fixed batch sizes* (see section II-C), using *unsorted F_n data*

[5]Flows prior to f_i have already been updated in an ordered manner (f_1 to f_n)

[6]The earliest recovery time for a flow is when it is the last flow in an update quantum having no earlier update quantums.

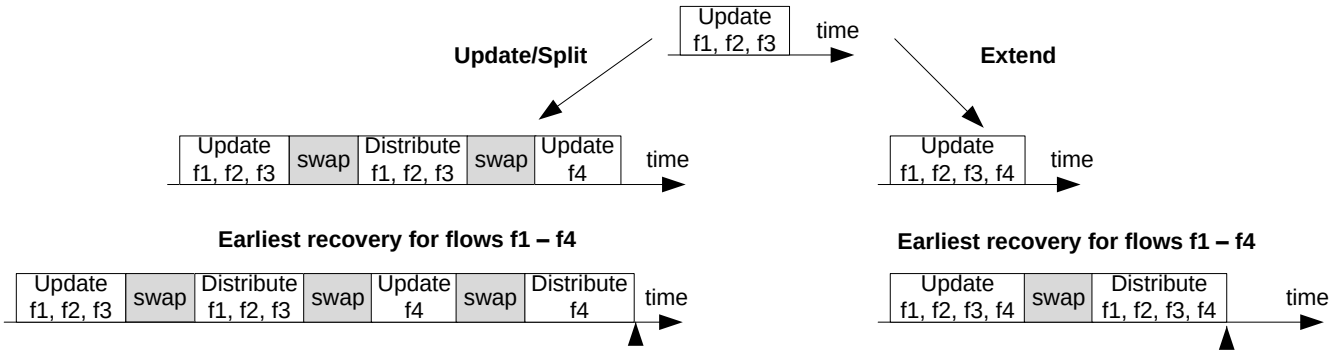


Fig. 4. Compare extension vs. split-scenario for F_4

	Earliest recovery time	Update/split scenario		Extension scenario	
		Recovery time	Diff. with earliest recovery time	Recovery time	Diff. with earliest recovery time
f_1	$t_u + t_d + t_s$	$3t_u + 3t_d + t_s$	$2(t_u + t_d)$	$4t_u + 4t_d + t_s$	$3(t_u + t_d)$
f_2	$2(t_u + t_d) + t_s$		$(t_u + t_d)$		$2(t_u + t_d)$
f_3	$3(t_u + t_d) + t_s$		0		$(t_u + t_d)$
f_4	$4(t_u + t_d) + t_s$	$4t_u + 4t_d + 3t_s$	$2t_s$		0

TABLE I
RECOVERY TIME COMPARISON OF F_4-SCENARIO: SPLIT VS. EXTENSION

2) The default update process with *fixed batch sizes*, using *sorted F_n data* (traffic flows f_i are sorted in descending order of associated traffic rate, see section III-B)
3) The optimized update process with *variable minimal batch sizes* ($x_{ui} \geq$X-axis-value), using the update strategy as defined in III-C on *unsorted F_n data*
4) The optimized update process with *variable minimal batch sizes* ($x_{ui} \geq$X-axis-value), using the update strategy as defined in III-C on *sorted F_n data*

The resulting figure 5a shows that the total packet loss increases almost linearly with respect to an increasing update-distribution batch size. This phenomenon can be observed for all applied variants. In addition, the experiment illustrates that there is a significant packet loss difference between variants using sorted F_n sets versus variants using unsorted F_n sets. This confirms intuition, given the observations in section III-B. The difference however becomes smaller for increased batch sizes. This can be understood because the value of sorting becomes lower if large batches of traffic flows are updated and distributed at the same time.

Another important aspect following from this experiment is that, the bigger the (minimal) batch size becomes, the less flexibility it leaves for adaptation, and as a consequence, the lesser gain can be achieved from dynamically optimizing the length of the quantum. This must be understood in two ways: at first, if there is no constraint on the batch size regarding minimum size, which should be the case, it is clear that packet loss of the optimization strategies is considerably lower compared to the alternatives. At the other hand if there is a constraint on the minimal batch size (represented by the variation in the X-axis), two factors are influencing the possible gain of the optimization strategies: the swapping time and the available room for optimization. The first follows from the fact that, the larger the batch becomes, the lower the weight of the swapping time becomes. When this weight becomes too low, it does not matter anymore if the update-distribution batches are extended or splitted during the update process because the trade-off resulting from the swapping time (see section III-C2) becomes very one-sided. This is validated in experiment IV-B2. The second factor is related to the partitioning possibilities that are left, given a minimal batch size. It is clear that using a batch size of 2500 traffic flows or larger, forces the update process to split the updates either into one batch containing all flows, or two batches of size $2500 + delta$.

The decrease in packet loss that can be achieved using optimized variants of the router update process within the range of lowest batch size constraints (i.e. $x_u \in [0, 60]$), can be validated in 5b. The figure underlines the possible gain of sorting and of optimizing. A striking detail is that using the optimization strategy as described in III-C2 can achieve the same gain (packet loss decrease) as sorting the F_n-set for the smaller batch sizes. The figure illustrates that the best performance can be achieved using a sorted F_n-set, with small batch sizes, optimized using the formulated strategy.

Optimizing the batch size using sorted F_n-sets leads to the best resuls. At best, this can result into a decrease of packet loss between 80 and 100 percent in the case the constraints on minimum batch size are low, leading to a major improvement compared to the situation where the batch size is fixed arbitrarily.

2) Packet loss vs. swapping time interval: The goal of the second experiment, is to evaluate the influence of the swapping time on the formulated variants of the router update process.

Therefore, a new experiment was set up, using a relatively large fixed value for the batch size ($x_u = 500$), while the swapping time interval was varied (X-axis) from 0 to 20000 μs (from 0 up to about the half of the update quantum).

The corresponding figure 5c shows linear increase in total packet loss for increasing swapping times for all router update variants. Clearly, strategies working on unsorted F_n sets suffer significantly more from increased swapping times compared to strategies using sorted F_n's. Figure 5d shows the gains in terms of percentage. In fact, sorting F_n results into an almost constant gain (between 60 and 80 percent) in terms of packet loss decrease, no matter the used swapping time interval.

As shortly mentioned in the previous experiment, the ratio of the swapping time interval (t_s) over the batch size (x_u) has an influence on the gain that can be achieved from optimization strategies. The figure depicts that a decrease in loss using optimization strategies can only be achieved from the moment the swapping time becomes larger than 5 ms (a tenth of the update quantum). In fact this gain can only be observed on the figure using unsorted F_n-sets. This is a result of the low splitting/update cost (fin_{i+s}). Having a low splitting/update cost will result into small update-distribution batches (i.e. as small as the minimal $x_u = 500$). Therefore the resulting packet loss will only decrease from the moment the swapping time, and thus the ratio of it over x_u becomes bigger.

3) Recovery time vs. (minimum) quantum time interval: The optimization strategies from section III-C influence the recovery time of different traffic flows, trying to shorten recovery times for high bit rate flows. However, in general, smaller packet loss does not necessarily mean that the total recovery time is decreased. After all, the total recovery time is increased by using smaller batch sizes, because more process swapping is involved[7]. Therefore figure 5e shows the effect on the total recovery time of the different variants versus the (minimal) batch size used[8]. As can be expected, the biggest differences in the graph can be observed for smaller batch sizes (because optimization mostly makes sense for these). As soon as a batch size of 50 ms or bigger is used, the recovery time for all variants becomes the same, and follows an increasing curve, fluctuating in steps. The fluctuation follows from the fact that the resulting F_n-set is divised into the minimum number of update-distribution batches of the given size dictated by the X-axis. This means that if the router update process is split into two batches of 4000 prefixes, that all flows only have recovered after the second batch which is only sparsely filled (1000 remaining prefixes).

Figure 5f shows the effect of increasing the swapping time interval (t_s) with respect to the total recovery time. An almost linear increase can be observed for all strategies. However, using the optimizing strategy from section III-C2 on a sorted F_n-set leads to a lower ratio of increase, compared to optimizing on an unsorted one. For the latter, the recovery time only starts to decrease (compared to a fixed batch strategy) from the moment the swapping time becomes big enough (i.e. 15 ms, or 30 precent of the update quantum).

[7]given a suffiently large swapping time, see previous experiment

[8]Because sorting F_n has no influence on the recovery time for the unoptimized strategies, only one graph is shown for vaiant 1 and 2.

V. Future work

Work in this paper can be extended such as to include not only the associated bandwidth rate of traffic flows but also their properties (e.g. elasticity, variability, (non-)real-time, etc.). Control traffic flows (e.g. DNS, ARP, etc.)) belong to the latter category and do not directly have a large influence on the packet loss during the router update process, as their associated rates are rather low. However, they do have an influence on future traffic, meaning that if control traffic is lost during the switchover, that packet loss can be significantly higher in future instances. Future research can focus on intelligent techniques to predict the value of this type of traffic and to accomodate the router update process accordingly. Systematic techniques for on-line classification of RIB/FIB updates per update-distribution batch (classifier of traffic rate per destination prefix) will also be further investigated.

VI. Conclusion

This paper modeled the router update process to improve its quality in terms of packet loss. Traffic flow statistics (i.e. traffic flow rate data) where taken into account to obtain this objective. Traffic-driven router update models were evaluated using strategies with either fixed or optimized variable sizes for the update-distribution batches. The resulting models were implemented in a simulation environment and were quantitatively characterized. Depending on the context, we showed that the formulated strategies can result into a significant decrease in packet loss of the router update process.

Acknowledgment

This work is supported by the European Commission-funded FP7-Fire ECODE project.

References

[1] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second igp convergence in large ip networks," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, pp. 33–44, July 2005.

[2] J. Moy, "OSPF Version 2," Internet Engineering Task Force, RFC 2328, Apr. 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2328.txt

[3] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271 (Draft Standard), Internet Engineering Task Force, Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4271.txt

[4] J. M. McQuillan, I. Richer, and E. C. Rosen, "An overview of the new routing algorithm for the arpanet," *SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 1, pp. 54–60, 1995.

[5] P. Narváez, K.-Y. Siu, and H.-Y. Tzeng, "New dynamic algorithms for shortest path tree computation," *IEEE/ACM Trans. Netw.*, vol. 8, no. 6, pp. 734–746, 2000.

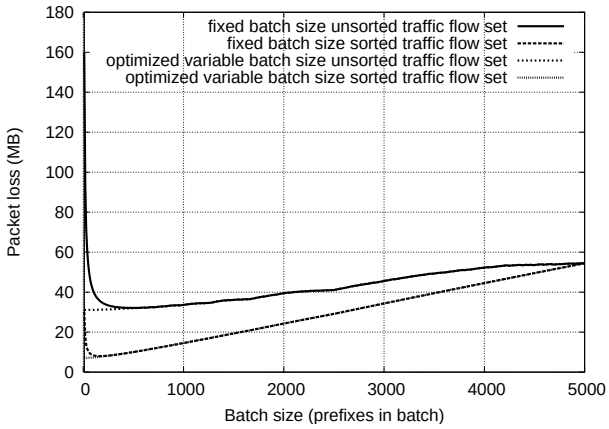
[6] D. P. Bovet and M. Cesati, *Understanding the linux kernel*, o' ed. O'Reilly, d 2003, no. ISBN : 0-596-00213-0.

[7] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, February 1993.

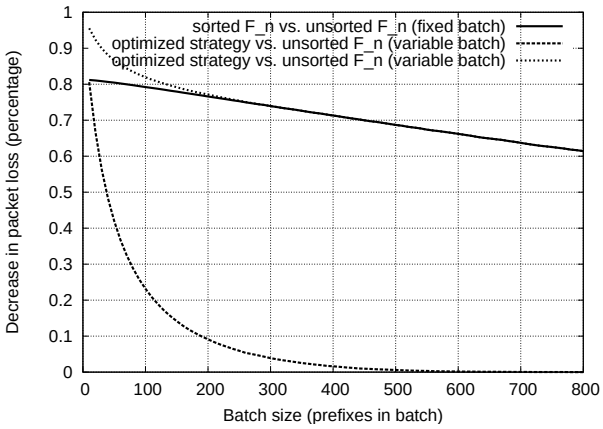
[8] N. L. for Applied Network Research (NLANR) Project, "Passive measurement and analysis," http://www.nlanr.net/PMA/.

[9] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 245–256, 2004.

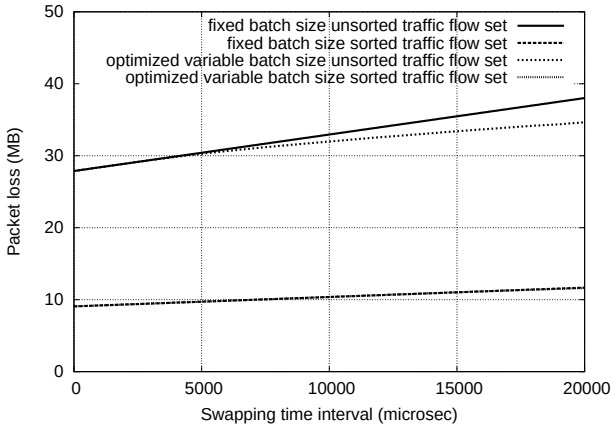
[10] W. E. Leland, M. S. Taqq, W. Willinger, and D. V. Wilson, "On the self-similar nature of Ethernet traffic," in *ACM SIGCOMM*, D. P. Sidhu, Ed., San Francisco, California, 1993, pp. 183–193.



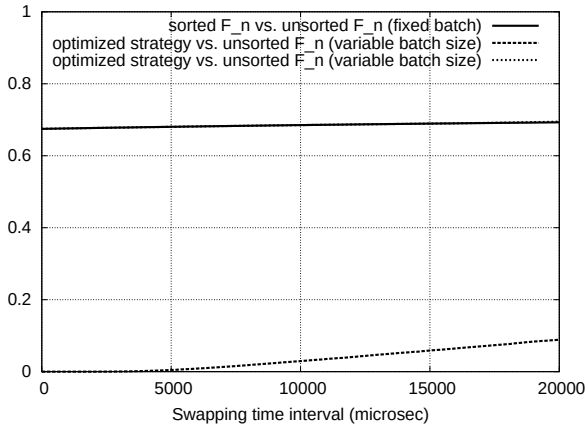
(a) Packet loss vs. (minimal) quantum



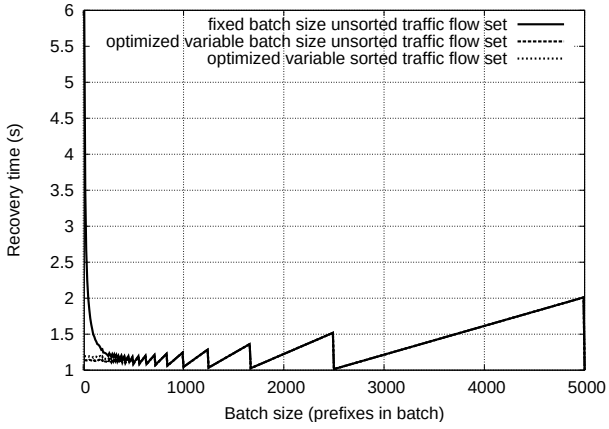
(b) Decrease in packet loss vs. (minimal) quantum



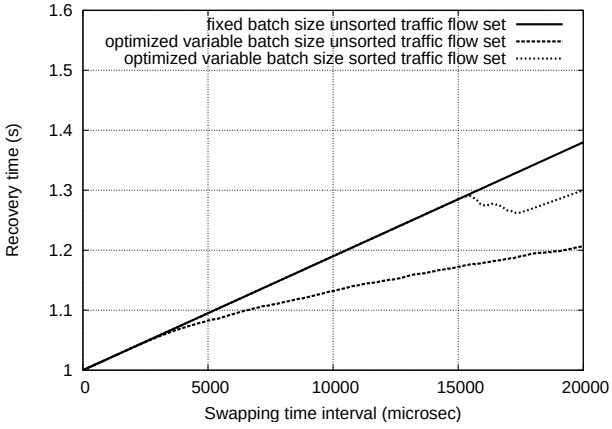
(c) Packet loss vs. swapping time interval



(d) Decrease in packet loss vs. swapping time interval



(e) Recovery time vs. (minimal) quantum



(f) Recovery time vs. swapping time interval

Fig. 5. Simulation results