Routing and Scheduling for *1+1* Protected DetNet flows

Gourav Prateek Sharma, Wouter Tavernier, Didier Colle, and Mario Pickavet

IDLab, UGent-IMEC,
Technologiepark 126, B-9052 Gent (Zwijnaarde)
Email: `firstname.lastname`@ugent.be

1 Abstract

Deterministic Networking (DetNet) is attracting a lot of attention lately due to their ability to provide bounded latency and zero packet loss for time-sensitive applications. In this paper, we formulate the routing and scheduling problem for *1+1* protected DetNet flows based on Cycle Specified Queuing and Forwarding (CSQF). The solution to this problem selects two paths between the two endpoints of each service request and schedules packet transmission on these paths meanwhile maximizing the accepted traffic. We have modeled the problem using Integer Linear Programming (ILP). We also propose two heuristic approaches: greedy and Tabu-search (TS) that can perform *1+1* routing and scheduling for a large number of requests in a reasonable time. Eventually, the performance of the ILP approach and heuristics is evaluated by performing simulation experiments. The results highlight the scalability of the two heuristics as compared to the ILP and superior performance of TS over greedy. The trade-off of cycle time on traffic acceptance and end-to-end delay is also presented.

2 Introduction

A wide range of real-time and safety-critical applications, such as connected cars, industrial control and media broadcasting, require deterministic performance from the network [**?**]. Proprietary bus-based technologies have been employed to build such networks that are currently hard to scale and manage. Although, packet-based network can offer best effort service but they fail to provide any Quality of Service (QoS) guarantees. Non-deterministic queuing delay in Ethernet switches can result in unbounded jitter and occasional packet losses thus rendering traditional Ethernet networks incapable of handling the above-mentioned applications. For instance, even a single packet loss or excess delay is not tolerable in TV broadcast production, as it would be detrimental to the viewing experience.
To be able to support time-sensitive applications, the network is required to provide a strict guarantee on latency, packet loss and jitter. The IEEE Time-sensitive Network (TSN) Task Group (TG) has developed a collection of standards to standardize the support of time-sensitive applications in Local Area Networks (LANs) by leveraging mechanisms such as, traffic-shaping, frame preemption, priority scheduling, etc. As the work done by the TSN WG has focused on LANs, these mechanisms are not suitable for a network spanning across multiple LANs. The IETF Deterministic Networking (DetNet) working group has been working on mechanisms that exploit L2 functionality in *Time Sensitive Networking* (TSN) so that zero packet loss along with deterministic latency and jitter can be achieved in L3. Particularly, the working group has described Cycle Specified Queuing and Forwarding (CSQF) based on Segment Routing to achieve deterministic performance [**?**] [**?**].
The end-to-end latency on a path between two points of a network can be bounded by capping the latency of each node (switch or router) in the path, as the latency component due to transmission delay and propagation delay is usually constant on a wired link. Cyclical queuing and dequeuing of packets in CSQF can be leveraged to limit the node latency. We have discussed the operation of CSQF in section 3. But it is worth mentioning here that by specifying packet transmission the end-to-end latency along any path can be bounded. Moreover,

this also ensures that the queue occupancy in any node does not exceed an upper limit. Packet losses due to congestion thus can be prevented by using queues of appropriate lengths that are not overflowed.
Avoiding packet overflows in queues guarantees zero congestion losses; however, is not the only cause for a packet loss. To provide protection against equipment (random media and/or memory) failures, dedicated protection for services is required. *1+1* protection has been a commonly exploited method to provide high reliability. The idea of *1+1* protection is to use two paths to route packets between the source and destination as opposed to a single path; the destination selects packets from one path only. In case of a node/link failure on one of the paths, the destination can just start receiving packets from the other path. Due to this simplistic architecture, this scheme has been popular in optical networks (SONET/SDH), where the destination switches to one of the path based on the switching criteria, e.g., signal strength [**?**]. However, *1+1* protection cannot be naively used for DetNet flows; packet scheduling needs to be performed along the two paths such that packets can be reliably recovered in case of a failure, as discussed in the next section.
Deploying *1+1* protected DetNet flows entails two components– (1) routing or selecting two paths between the source and destination and (2) generating reliable packet schedules along these path. The problem of joint routing and scheduling in DetNets using CSQF has been investigated in [**?**]. To the best of our knowledge, modeling of protection schemes in DetNets has not been addressed yet. Therefore, we address the Routing and Scheduling (RTSCH) problem for *1+1* protected DetNet flows by formulating a Integer Linear Program (ILP) for it. Additionally, we propose two heuristic algorithms to solve the above problem instances of reasonable size.
The rest of the paper is organized as follows. The technical background regarding CSQF and *1+1* protection is provided in section 3. Section 4 details the related works for this paper. The system model and ILP formulation for the *1+1* DetNet protection problem are described in section 5. In section 6, we have discussed (a) greedy based heuristic and (b) Tabu-search based heuristic. Section 7 presents the performance evaluation of the ilp and two heuristics in terms of various metrics. Finally, the paper is concluded in section 8.

3 CSQF and *1+1* protection

This section gives an overview of CSQF and also motivates the use of CSQF for *1+1* protection in DetNet.

3.1 CSQF Overview

IEEE 802.1Qch defines the queue mode and workflow of Cyclic Queuing and Forwarding for time-sensitive flows in LANs. CQF specifies the division of transmission time at the output interface of every node in even and odd cycles that correspond to two queues of the interface. During an odd cycle, the packets transmitted by the upstream nodes are queued in the second queue and the packets received during the previous (even) cycle are transmitted from the first queue to the downstream node. CQF requires that the packet transmitted by the upstream node in a cycle are received in the same cycle by the downstream node, so that it can be transmitted by the downstream node in the next cycle. With cyclic queuing/dequeuing, the delay experienced by the packets along the path can be bounded in terms of the number of hops. The requirement on transmission and reception in the same cycle restricts the size of network as packets transmitted on the longer links are not being received in the same cycle (without increasing the cycle time). This requirement restricts the use of CQF only for LANs and is rather unfit for large-scale DetNet.
Enhancements to CQF have led to the development of the CSQF mechanism [**?**]. By specifying the sending time of all nodes along the path between the source and destination. Thus the requirement to send and receive in the same cycle is relieved by scheduling transmissions based on path delay. Fig. 1 shows the queuing model of CSQF with three queues (more are also possible), namely, q_0, q_1 and q_2. Packets queued in one particular queue are dequeued during a cycle and other queues are used to enqueue packets. For instance, during cycle $c = 1$, in Fig. 1, q_2 is receiving packets whereas q_1 is transmitting the packets that it received during cycle $c = 0$. During this cycle, q_0 is used as a toleration queue to absorb packets received earlier (due to small variations in transmission or processing time), i.e, packets supposed to be arriving in cycle $c = 2$ but are received during $c = 1$. The roles: receiving, sending and tolerating rotate among q_0, q_1 and q_2 every cycle.
CSQF requires packets to be received, queued and transmitted at specific time instances through the network path. Each node on the path should forward the packet to the right port and select the packet to the

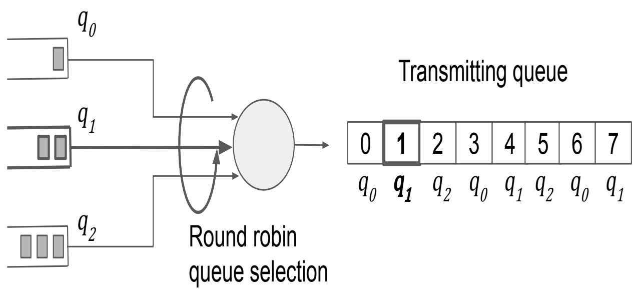

q_0
q_1
q_2
Transmitting queue
0 1 2 3 4 5 6 7
q_0 q_1 q_2 q_0 q_1 q_2 q_0 q_1
Round robin queue selection

Figure 1: Mapping of cycle times to queues in the CSQF operation.

right queue for transmission. Segment Routing (SR) can be leveraged to route and schedule CSQF-based flows. Each source in SR network appends a header containing a list of instruction to traverse through the network. An SID (SR Identification), e.g., IPv6 headers, can be used to carry instructions that specify what interface the packet should be sent from and which queue to select; thus determining which cycle it should be transmitted in. By stacking multiple SID labels not only the explicit route but also the transmission schedule along the path is defined without the need of maintaining per-flow state of the flow in network nodes. Hence, SR enables DetNet to support large CSQF flows and scalability to large networks.
As might be expected, both CQF and CSQF require the nodes in network to have the same sense of timing to schedule. This is usually achieved by distributing a reasonably stable clock using the Precise Timing Protocol (PTP) as described in IEEE 802.1AS [?]. Henceforth, we assume all network nodes are synchronized within sub-microsecond accuracy.

3.2 Scheduled *1+1* protection

DetNet services avoid packet loss because of contention between different flows by traffic regulation combined with careful packet scheduling. However, service disruption due to failures such as random media failures, memory errors, etc, needs to be eliminated using a protection mechanism. Dedicated protection schemes, e.g., *1+1*, have been widely used in packet-based networks [?]. In *1+1* protection, an extra path, in addition to the original path, is used to route packets between the source and destination. At the destination end, packets received on the two paths are de-duplicated.
Due to different end-to-end delays along the two paths, the reliability of *1+1* protection can be impacted. This phenomenon is demonstrated in Fig. 2. The Packet Replication Function (PRF) present on the source is responsible for duplicating the packets, that are received on its input interface onto its two output interfaces that are subsequently routed through two different paths p_1 and p_2. At the destination, the Packet Elimination Function (PEF) performs packet de-duplication/elimination on packets received on the last links $(p_1[-2], p_1[-1])$ and $(p_2[-2], p_2[-1])$ of paths p_1 and p_2, respectively. Packet de-duplication entails the presence of sequencing information with packets that can be done by adding a sequence number or timestamp to the packet while doing duplication at PRF.
The above mentioned functions for *1+1* protection can be implemented via IP encapsulation/decapsulation of the Detnet flows. For instance, [?] proposed to use P4 match-action tables to implement PRF and PRE at the source and destination nodes, respectively. As this paper is not concerned with the actual implementation of these functions, we will next explain only their high-level logic. The PRF simply adds the packet sequence number to the packet header and then duplicates the packet to send them over the two interfaces of the source node. With packet sequence information, packet elimination becomes quite straightforward. The overview of PEF logic is shown in Fig. 2 (b). The PEF remembers the highest sequence number seq_{lst}

of the last output cycle and only outputs ($output(pkts)$) the packets of this cycle if its highest sequence number seq_{pkts} is higher than seq_{lst}. Next, the PEF updates seq_{lst} to seq_{pkts}. In case, if $seq_{pkts} < seq_{lst}$, the packets of this cycle are discarded. The simple logic of a PEF allows it to perform packet elimination without contributing a significant latency and excessive buffering.

With such a PEF at the destination, a reliable packet elimination is only possible if the packet transmission

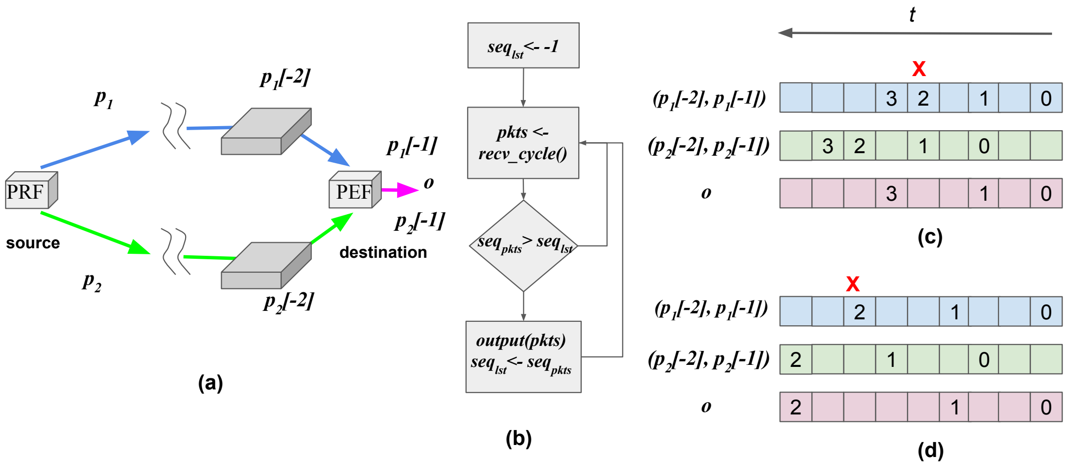


Figure 2: Reliable recovery with *1+1* CSQF. (a) *1+1* protection, (b) PEF logic, (c) Naive scheduling and (d) End-to-end delay aware scheduling.

schedules on the two paths follow the *spacing constraint*. Let's assume that each cycle has a bandwidth to send one packet; the PRF sends four packets in four cycles over the two disjoint paths, where the end-to-end latency of p_2 is higher than p_1 as shown in Fig. 2(c). The packet schedules on links ($p_1[-2], p_1[-1]$) and ($p_2[-2], p_2[-1]$) and the output o of the PEF is shown. The PEF receives $pkt_0, pkt_1, pkt_2, pkt_3$ on p_1 during cycles $0, 2, 4, 5$ and on p_2 during cycles $2, 4, 6, 7$, respectively. Supposing pkt_2 is lost somewhere on p_1, the PEF is not able to recover it from p_2, as the next packet received by the PEF with $seq > 1$ (after recovering pkt_1) is pkt_3 on p_1, though pkt_2 is received on p_2 later.

The above situation can be avoided by carefully scheduling packet transmissions on the *1+1* paths. The *spacing constraint* on packet schedules ensures that any two cycles in the packet schedule with non-zero bandwidth allocation must be separated by at least the absolute value of the difference between the end-to-end delays of the two paths. This constraint ensures reliable recovery as shown in Fig. 2 (d) where the new schedules adheres to the spacing constraint.

It is worth noting that reliable recovery is possible without satisfying the spacing constraint, though that requires an additional component: Packet Re-ordering Function (PROF). The PROF would require additional buffering to store the received packets on the two paths and then perform de-duplication. Though, de-duplication can be performed efficiently by maintaining a heap data structure for the received packets, where each heap insertion operation has $O(log(n))$ complexity; nonetheless, the buffering and additional computation could contribute a significant latency to the packet recovery process. In this paper, we therefore assume that the destination does not perform packet re-ordering.

4 Related Works

Although, the attempts to make networks more deterministic have been going on for several years; however, the standardization work by the TSN task group in IEEE 802.1 only started in 2015. These standards have sought to guarantee bounded latency and high reliability in Layer 2. For reliability, the group proposed IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER) as a new Ethernet sub-protocol

[?]. To extend these guarantees to Layer 3, IETF DetNet TG has proposed various mechanisms [?].
With coming up of TSN standards, researchers have shown keen interest in developing algorithms for the deployment of time-sensitive applications. Numerous studies have focused on generating schedules for time-triggered traffic in IEEE 802.1Qbv/TSN such that a given objective function is optimized. Craciunas et al. have used Satisfiability Modulo Theory (SMT) to optimize the network as well as application schedules in a switched multispeed time-triggered network [?]. In [?], the authors have formulated the problem of routing and scheduling of time-triggered traffic in TSNs as an ILP to provide real-time guarantees by exploiting logical centralization paradigm of software-define networking. Approaches based on SMT or ILP generate exact solutions by expressing the scheduling problem as a constrained optimization or constrained satisfaction problem that are solved by ILP or SMT solvers. The advantage of using exact approaches is that they generate provable optimal solutions. On account of scheduling problems being NP-hard, exact approaches do not scale well to flow numbers and network sizes. Alternatively, heuristic and metaheuristic approaches solve these problem by compromising on the quality of the solution for execution speed. Tabu search in [?] and incremental backtracking in [?] and [?] have been proposed to speed up the joint RTSCH problem with close to an optimal solution.
Recently, the RTSCH problem in CSQF-based network has also been addressed in various studies. In [?], the joint routing and scheduling problem for DetNets is formulated using ILP and two methods–column generation and dynamic programming, are presented to maximize traffic acceptance while solving the problem. Moreover, a scalable greedy algorithm is also proposed that is capable of solving the problem with a extra small gap. Similarly, the authors in [?] have modeled load balancing in DetNets where time-sensitive flows are split over multiple paths in order to improve network utilization.
Reactive mechanisms such as Dynamic Multipath Optimization (DMPO), through the use of continuous network monitoring and flow prioritization, can provide a level of reliability by re-routing the critical traffic during an interruption [?]. However, the restoration time in DPMO spans hundreds of milliseconds to seconds; thus, an interruption can still be detrimental for Detnet flows. Dedicated path protection schemes like *1+1* have been studied in relation to Elastic Optical Networks (EONs). Walkowiak et al. have formulated the offline problem of routing and spectrum allocation (RSA) to dedicated path protection (DPP) in EONs as an ILP. Tabu-search-based metaheuristics were proposed to solve the problem instances of reasonable size. These works are limited largely to the optical networks and hence do not address the problem of dedicated protection in DetNets. In contrast to these studies, our work focuses on the problem of *1+1* protection for DetNet flows. We assume the nodes in the given DetNet are CSQF capable and the source and the destination are capable of performing the necessary operations to provide *1+1* protection.

5 System Model and ILP Formulation

Next, we describe the system model and then formulate the *1+1* RTSCH problem as an ILP. As discussed earlier, the time at each node's output port is divided into intervals of equal duration T_c, referred as *cycles*. The number of cycles after which the traffic pattern of the application repeats is referred as *hypercycle*; therefore, it is sufficient to consider the problem for one hypercycle only. Moreover, it can be assumed without loss of generality that cycles on each node are aligned, i.e, start at the same time.

Physical infrastructure: The deterministic network infrastructure is represented using a directed graph $G = (N, E)$, where N is the set of CSQF-capable switches or routers and E is the set of physical links connecting the nodes in N. For each link $e = (n_i, n_j) \in E$ the cycles in a hypercycle are denoted by C and the available bandwidth for each cycle $c \in C$ denoted by $bw_{n_i,n_j,c}$. For instance, on an unused 10G link during each cycle of T_c=20μs, a total of 25000B or 16x1500B packets are available. The total delay introduced on the link (n_i, n_j) is represented by d_{n_i,n_j}, that includes the propagation delay, transmission delay and processing delay due to n_j.
Requests: A set of requests R needs to be RTSCH'ed with *1+1* protection on G. Each request $r \in R$ has a corresponding five-valued tuple $\mathbb{T}^r = (n_s^r, n_d^r, bw^r, D_r^{e2e}, \delta D_r^{e2e})$ that contains the metadata required for RTSCH'ing r. Here, n_s^r and n_d^r are the r's source and destination between which two DetNet flows are required; bw^r denotes the r's bandwidth requirement during one hypercycle; D_r^{e2e} and δD_r^{e2e} are the maximum end-to-end delay and jitter permissible for request r. Fig. 3 shows $\mathbb{T}^r$ for request r, $n_s^r = n_0$, $n_d^r = n_9$ and $bw^r = 64$ packets(=92kB); for $T_c = 20\mu$s and $|C| = 50$, r requires full four cycles out of fifty

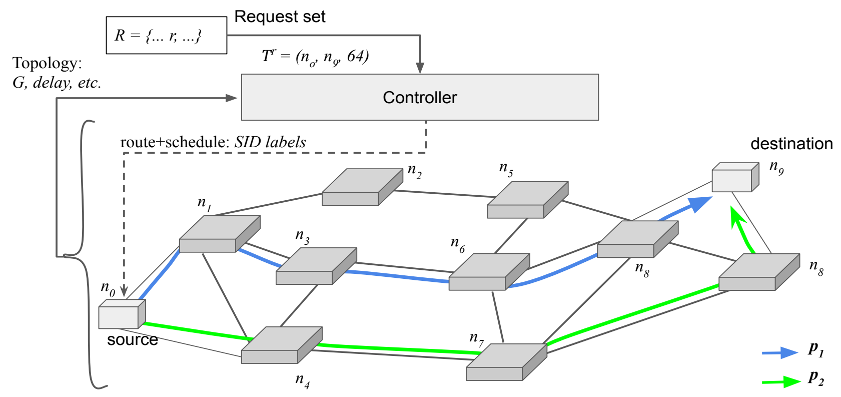


Figure 3: Illustration of the *1+1* RTSCH problem.

cycles along each link of p_1 and p_2.
Variables: Binary variable γ_p^r indicates the routing of $r \in R$ using $p \in P^{n_s^r, n_d^r}$; $\gamma_p^r = 1$, if path p is used for the routing of r; otherwise $\gamma_p^r = 0$. Continuous decision variable $\omega_{p,n_i,n_j,c}^r$ denotes the amount of bandwidth allocated to request r on physical link $(n_i, n_j) \in p, p \in P^{n_s^r, n_d^r}$ during cycle $c \in C$. In addition to binary variables γ_p^r and $\omega_{p,n_i,n_j,c}^r$, we next introduce a few helpers or dependent variables that are required in the ILP formulation.
Binary variable λ^r indicates if exactly two disjoint paths are allocated for the routing of r. $\rho_{p,n_i,n_j,c}^r$ is a binary variable that indicates if non-zero bandwidth is allocated to request $r \in R$ on link $(n_i, n_j) \in p, p \in P^{n_s^r, n_d^r}$. Binary variable $\eta_{p,n_i,n_j,c_1,c_2}^r = 1$, if at least one of ρ_{p,n_i,n_j,c_1}^r and ρ_{p,n_i,n_j,c_2}^r is 0; otherwise $\eta_{p,n_i,n_j,c_1,c_2}^r = 0$. Table 1 lists the parameters and variables involved in the system model along with a short description.
In the *1+1* RTSCH problem, given set of requests R and DetNet topology G, requests must be routed and scheduled through two disjoint paths. Fig. 3 shows an illustration where the controller takes as input DetNet topology G along with request set R and solves the *1+1* RTSCH problem. The controller itself can be distributed / replicated across multiple instances for robustness and scalability reasons [**?**, **?**] . Nevertheless, we can assume a single logical controller that is responsible for routing and scheduling detnet flows. The solution of the *1+1* RTSCH problem, i.e., routes and schedules for requests can then be used to configure the source nodes with SID label stacks, as discussed earlier.

5.1 ILP Formulation

In this section, the *1+1* RTSCH problem is formulated as an optimization problem. The goal is to find two disjoint paths between the sender and receiver of each request as well as schedule packet transmission along the two paths meanwhile optimizing an objective function, e.g., maximizing the total accepted traffic, minimizing the jitter or end-to-end delay on the two paths.

The DetNet *1+1* protection problem is formulated as a Integer Linear Program (ILP). The objective of the formulation is to maximize the total accepted traffic for *1+1* RTSCH'ing.

$$obj : \max \sum_{r \in R} \lambda^r bw^r \tag{1}$$

The constraints for this ILP formulation are listed from (2) to (13). Constraint pair (2-3) ensures exactly two paths are selected from $P^{n_s^r, n_d^r}$ for the request mapping, where λ^r indicates *1+1* routing. With this constraint pair, λ^r is 0 if $\sum\limits_{p \in P^{n_s^r, n_d^r}} \gamma_p^r = 0$ and is 1 if exactly two paths are selected. The constraint (4) enforces disjointedness on the selected paths. In this constraint, if selected paths p_1 and p_2 are disjoint, i.e, $disj(p_1, p_2) = 1$, then the LHS (=1) equals the RHS (=1); however, if they are non-disjoint, i.e, $disj(p_1, p_2) = 0$, then the LHS (=1) is greater than the RHS (=0), which shall prevented by the constraint. The number of packets (bandwidth) allocated to a request during a cycle must be conserved, i.e, the total packets

Table 1: Description of the notations used for different parameters and variables involved in the system model.

Notation	Description
$G = (N, E)$	Directed graph representation of the DetNet infrastructure, where N is the set of nodes and E is the set of physical links between the nodes.
N_e	$N_e \subset N$ is the set of all endpoints, i.e, sources and destinations.
C	The set of all cycles in a hypercycle.
$bw_{n_i,n_j,c}$	Available bandwidth (in bytes) on physical link $(n_i, n_j) \in E$ during cycle $c \in C$.
$d^{tx}_{n_i,n_j}$	Transmission delay on link (n_i, n_j).
d^{proc}_{n}	Maximum processing delay on node n.
d_{n_i,n_j}	Total delay on link (n_i, n_j); $d_{n_i,n_j} = d^{tx}_{n_i,n_j} + d^{proc}_{n_j}$.
R	Set of all requests to be RTSCH'ed on G.
$\mathbb{T}^r$	Request tuple $\mathbb{T}^r = (n^r_s, n^r_d, bw^r, D^{e2e}_r, \delta D^{e2e}_r)$ corresponding to $r \in R$, where n^r_s, n^r_d, bw^r, D^{e2e}_r and δD^{e2e}_r are the source, destination, bandwidth requirement (in bytes) of r, maximum permissible end-to-end delay and jitter, respectively.
$P^{n^r_s,n^r_d}$	The set of all paths between the source n^r_s and the destination n^r_d of $r \in R$.
γ^r_p	Binary variable indicates the routing of $r \in R$ on $p \in P^{n^r_s,n^r_d}$.
λ^r	Binary (helper) variable indicates the *1+1* routing of $r \in R$.
$\omega^r_{p,n_i,n_j,c}$	Bandwidth (in bytes) allocated to request $r \in R$ on link $(n_i, n_j) \in p$, $p \in P^{n^r_s,n^r_d}$, during cycle interval $c \in C$.
$\rho^r_{p,n_i,n_j,c}$	Binary (helper) variable indicates if non-zero bandwidth is allocated to request $r \in R$ on link $(n_i, n_j) \in p$, $p \in P^{n^r_s,n^r_d}$, during cycle interval $c \in C$.
$\eta^r_{p,n_i,n_j,c_1,c_2}$	Binary (helper) variable $\eta^r_{p,n_i,n_j,c_1,c_2}$ equals 1, if at least one of ρ^r_{p,n_i,n_j,c_1} and ρ^r_{p,n_i,n_j,c_2} is 0; otherwise $\eta^r_{p,n_i,n_j,c_1,c_2} = 0$.

transmitted in a cycle at a link should be equal to the total packets transmitted on its downstream link in the cycle shifted by the link delay. Mathematically, the amount of bandwidth allocated on a link (n_i, n_j) of path p during cycle c is equal to the bandwidth allocated on downstream link (n_j, n_k) on p during cycle $(c + d_{n_j,n_k})\% \mid C \mid$ as expressed in constraint (5). Fig. 4 illustrates the packet scheduling along network path p. The packet schedule on link (n_0, n_1) is delayed by the link delay $d_{n_0,n_1} = 5T_c$ to get the schedule for (n_1, n_2) which is delayed by $d_{n_1,n_2} = 4T_c$ to get the schedule for (n_2, n_3). Cycles 5 and 7 on link $(n1, n2)$ and cycles 9≡ 1(%8) and 11≡ 3(%8) on link $(n2, n3)$, have bandwidth equal to cycles 0 and 2 on link $(n0, n1)$. The sum of bandwidth allocated to all requests during a cycle cannot exceed the available bandwidth in the cycle; this condition is ensured by constraint (6). The total bandwidth allocated in hypercycle C should be equal to the bandwidth requirement of r; this is guaranteed by constraint (7).

The pair of constraints in (8a-8b) ensures that the same schedule is used on the first link $((p_1[0], p_1[1])$ and $(p_2[0], p_2[1]))$ of the two selected paths.

As discussed in section 3, at the receiver, any two cycles in the schedule with non-zero bandwidth should be spaced by at least by the difference of the end-to-end delays along the two selected paths. The pair of constraints in (9a-9b) enforces the minimum spacing $(|D_{p_2} - D_{p_1}|)$ on the schedules of p_1 and p_2 near the receiver end $((p_1[-2], p_1[-1]), (p_2[-2], p_2[-1]))$. For path p_1, when two cycles c_2 and c_1 have minimum bandwidth allocated, i.e, $n^r_{p_1,p_1[-2],p_1[-1],c_1,c_2} = 0$, then the LHS of the constraint pair $(c_2 - c_1)$ is at least

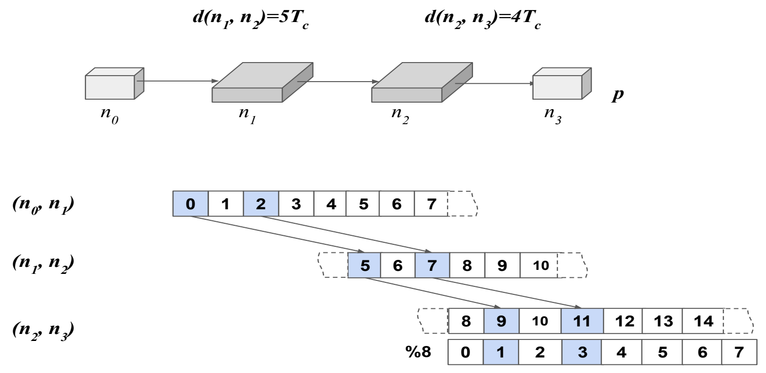


Figure 4: Relationship of packet schedules on various links of a path.

$|D_{p_2} - D_{p_1}|$. In other words, if $D_{p_2} \geq D_{p_1}$, then (9a) is active while (9b) is inactive and if $D_{p_2} \leq D_{p_1}$, then then (9b) is active while (9a) is inactive. If at least one of $\rho^r_{p_1,n',n^r_d,c_1}$ and $\rho^r_{p_1,n',n^r_d,c_2}$ is 0 ($n^r_{p_1,p_1[-2],p_1[-1],c_1,c_2} = 1$), the pair of constraints is inactive.
The binary variable $\rho^r_{p,n_i,n_j,c}$ in constraint (10) is 0 when $\omega^r_{p,n_i,n_j,c} = 0$ and is 1, if at least ω^r_{min} is allocated. For Detnet flows, it is required that the packets transmitted by n^r_s are received by n^r_d within a permissible delay D^{e2e}_r. With CSQF, the worst-case delay along a path p can be expressed as $\sum\limits_{(n_i,n_j)\in p} d_{n_i,n_j}$ [?]. The constraint in (11) imposes this condition on the two selected paths for each request. As far as the jitter with CSQF is concerned, its value is dependent on the spacing between any two cycles with non-zero bandwidth. The maximum (intra-cycle) jitter is independent of the chosen path and is equal to $2T_c$. Therefore, the cycle time should be chosen such that $T_c \leq D^{e2e}_r$. The constraint trio in (12a, 12b, 12c) ensures $\eta^r_{p,n_i,n_j,c_1,c_2}$ is 0 only if both ρ^r_{p,n_i,n_j,c_1} and ρ^r_{p,n_i,n_j,c_2} are 0 and is 1, otherwise. Variables γ^r_p, λ^r, $\rho^c_{p,n_i,n_j,c}$ and $\eta^r_{p,n_i,n_j,c_1,c_2}$ can only take binary values (0 or 1) while $\omega^r_{p,n_i,n_j,c}$ is bounded; these limits are expressed in constraint (13).

$$\sum_{p\in P^{n^r_s,n^r_d}} \gamma^r_p \geq 2\lambda^r, \; \forall r \in R. \tag{2}$$

$$\lambda^r \geq \gamma^r_p, \; \forall r \in R, \; \forall p \in P^{n^r_s,n^r_d}. \tag{3}$$

$$\gamma^r_{p_1} + \gamma^r_{p_2} - 1 <= disj(p_1, p_2), \; \forall r \in R, \forall p_1, p_2 \in P^{n^r_s,n^r_d}. \tag{4}$$

$$\begin{aligned}\omega^r_{p,n_i,n_j,c} &= \omega^r_{p,n_j,n_k,(c+d_{n_j,n_k})\%|C|} \\ &, \forall r \in R, \forall p \in P^{n^r_s,n^r_d}, \forall (n_i,n_j),(n_j,n_k) \in p, \forall c \in C.\end{aligned} \tag{5}$$

$$\sum_{r\in R} \omega^r_{p,n_i,n_j,c} \leq bw_{n_i,n_j,c}, \; (n_i,n_j) \in E, \forall c \in C. \tag{6}$$

$$\sum_{c\in C} \omega^r_{p,n_i,n_j,c} = bw^r \gamma^r_p, \; \forall r \in R, \forall p \in P^{n^r_s,n^r_d}, \forall (n_i,n_j) \in p. \tag{7}$$

$$\omega^r_{p_1,p_1[0],p_1[1],c} \geq \omega^r_{p_2,p_2[0],p_2[1],c} + (\gamma^r_{p_1} + \gamma^r_{p_2} - 2)\omega_{max}, \; \forall r \in R, \forall p_1, p_2 \in P^{n^r_s,n^r_d}, \forall c \in C. \tag{8a}$$

$$\omega^r_{p_1,p_1[0],p_1[1],c} \leq \omega^r_{p_2,p_2[0],p_2[1],c} - (\gamma^r_{p_1} + \gamma^r_{p_2} - 2)\omega_{max}, \; \forall r \in R, \forall p_1, p_2 \in P^{n^r_s,n^r_d}, \forall c \in C. \tag{8b}$$

$$c_2\rho^r_{p_1,p_1[-2],p_1[-1],c_2} - c_1\rho^r_{p_1,p_1[-2],p_1[-1],c_1} \geq$$
$$(D_{p_2} - D_{p_1}) - C_{max}\eta^r_{p_1,p_1[-2],p_1[-1],c_1,c_2}, \quad (9a)$$
$$\forall r \in R, \forall p_1, p_2(\neq p_1) \in P^{n^r_s,n^r_d}, (p_1[-2], p_1[-1]) \in p_1, \forall c_1, c_2(> c_1) \in C.$$

$$c_2\rho^r_{p_1,p_1[-2],p_1[-1],c_2} - c_1\rho^r_{p_1,p_1[-2],p_1[-1],c_1} \geq$$
$$(D_{p_1} - D_{p_2}) - C_{max}\eta^r_{p_1,p_1[-2],p_1[-1],c_1,c_2}, \quad (9b)$$
$$\forall r \in R, \forall p_1, p_2(\neq p_1) \in P^{n^r_s,n^r_d}, (p_1[-2], p_1[-1]) \in p_1, \forall c_1, c_2(> c_1) \in C.$$

$$\omega^r_{min}\rho^r_{p,n_i,n_j,c} \leq \omega^r_{p,n_i,n_j,c} \leq \omega^r_{max}\rho^r_{p,n_i,n_j,c}, \quad \forall r \in R, \forall p \in P^{n^r_s,n^r_d}, (n_i, n_j) \in p, \forall c \in C. \quad (10)$$

$$\gamma^r_p \sum_{(n_i,n_j)\in p} (d_{n_i,n_j}) \leq D^{e2e}_r, \forall r \in R, \forall p \in P^{n^r_s,n^r_d}. \quad (11)$$

$$\eta^r_{p,n_i,n_j,c_1,c_2} \geq (1 - \rho^r_{p,n_i,n_j,c_1}), \forall r \in R, \forall p \in P^{n^r_s,n^r_d}, (n_i, n_j) \in p, \forall c \in C. \quad (12a)$$
$$\eta^r_{p,n_i,n_j,c_1,c_2} \geq (1 - \rho^r_{p,n_i,n_j,c_2}), \forall r \in R, \forall p \in P^{n^r_s,n^r_d}, (n_i, n_j) \in p, \forall c \in C. \quad (12b)$$
$$\eta^r_{p,n_i,n_j,c_1,c_2} \leq (2 - \rho^r_{p,n_i,n_j,c_1} - \rho^r_{p,n_i,n_j,c_2}), \forall r \in R, \forall p \in P^{n^r_s,n^r_d}, (n_i, n_j) \in p, \forall c \in C. \quad (12c)$$

$$\gamma^r_p \in \{0,1\}, \lambda^r \in \{0,1\}, \eta^r_{p,n_i,n_j,c_1,c_2} \in \{0,1\}, \omega^r_{p,n_i,n_j,c} \in [0, \omega^r_{max}], \rho^r_{p,n_i,n_j,c} \in \{0,1\}$$
$$\forall r \in R, \forall p \in P^{n^r_s,n^r_d}, \forall (n_i, n_j) \in p, \forall c, c_1, c_2 \in C. \quad (13)$$

6 Heuristics

The ILP approach, presented in the previous section, solves the *1+1* RTSCH problem by navigating through the solution space consisting of all combinations of variable values. As the total number of variables in the formulation increases drastically with the size of the problem the resulting solution space is huge. Therefore, we next present two heuristics that have relatively low execution times and provide sub-optimal but decent solutions.

6.1 Greedy Heuristic

The pseudocode for the greedy heuristic to solve to solve the *1+1* RTSCH problem is shown in Alg. 1. The procedure `batch_rt_sch` is called with set of requests R arranged in the decreasing order of bw^r values. Thus, `greedy_rt_sch` is called first for r with the highest bw^r value and last for r with the lowest bw^r value (l. 47).
In `greedy_rt_sch`, first, all paths between the two endpoints n^r_s and n^r_d are generated and stored in $P^{n^r_s,n^r_d}$ (l. 22). We make use of Networkx's all_simple_paths procedure to generate the paths between the given endpoints [?]. The algorithm utilizes a modified depth-first search for path computation and runs in $O(N + E)$ time for each path.
Next, a pair of *for* loops is used to select the first two paths on which routing and scheduling is feasible.
For a path-pair $(p_1, p_2) \in P^{n^r_s,n^r_d} \bigtimes P^{n^r_s,n^r_d}$ pair, it is checked whether p_1 and p_2 are disjoint and if enough cycles are available to generate a feasible schedule; otherwise, the next (p_1, p_2) pair is considered (l. 27).
Consider for example a path pair p_1 and p_2, a hypercycle with eight cycles ($|C| = 8$) and each cycle with a bandwidth of 16 (1500B) packets, if the difference in the end-to-end delay along p_1 and p_2 is 4 cycles then the requests with $bw^r > 16\lceil 8/2\rceil = 32$ cannot be routed on p_1 and p_2, irrespective of the schedule. However, if the difference in the end-to-end delay is 3 cycles, then only the requests with $bw^r > 48$ are rejected. This

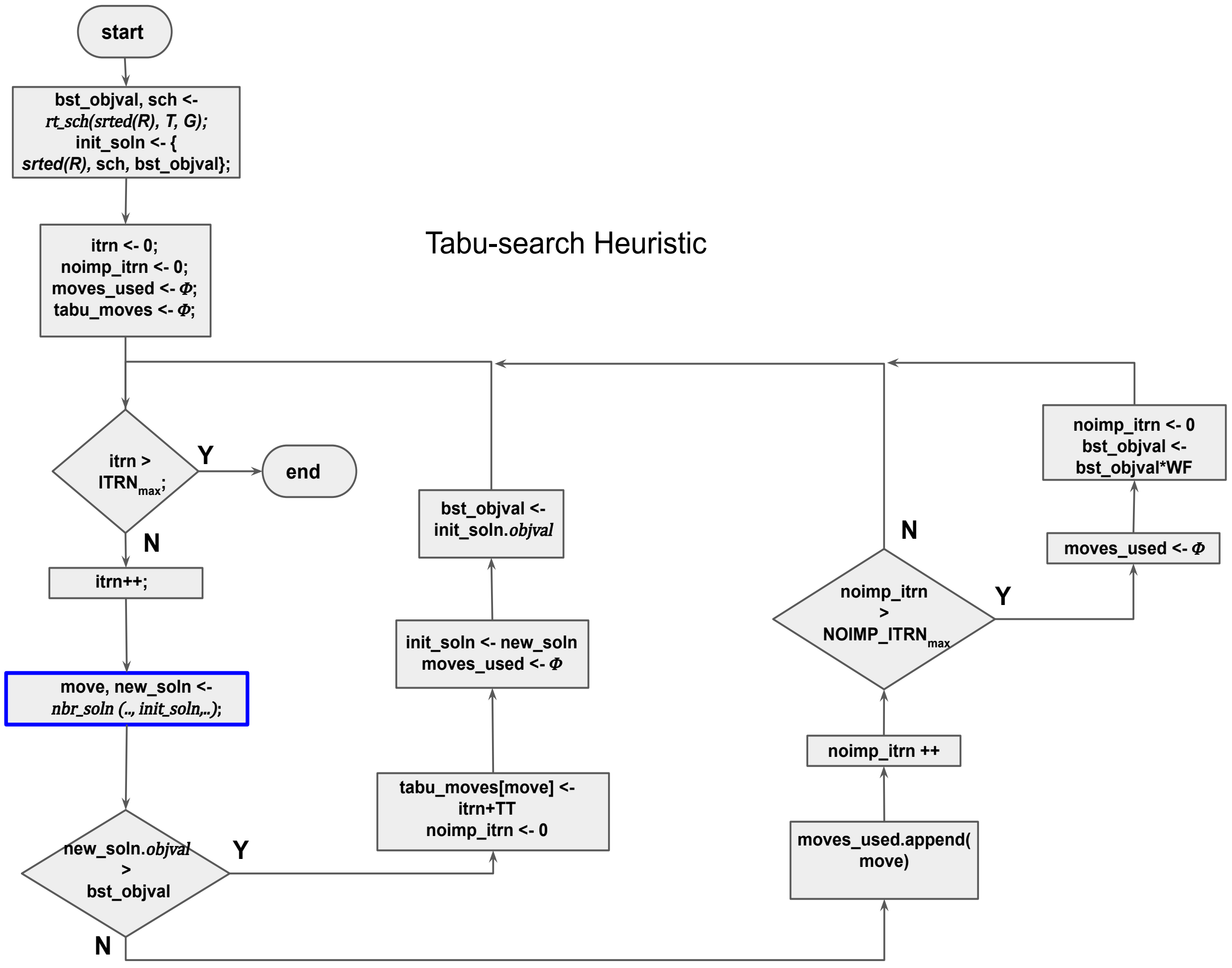
step ensures that next step (schedule computation) is not executed if scheduling is not possible for the selected path pair p_1 and p_2.
Using `gen_sch`, a schedule is generated at the destination end $((p_1[-2], p_1[-1]))$ starting with cycle c_{strt} (l. 30). The procedure `gen_sch` greedily attempts to allocate bw^r among C cycles of $(p_1[-2], p_1[-1])$, with start cycle c_{strt} and a minimum cycle spacing of δc (l. 1-20). If such schedule exists ($sch_{p_1,d} \neq \phi$) at the destination end, then using procedure `sch_pth` with direction argument $dir = -1$ (from destination to source), the schedule for all the links in p_1 is generated by (negatively) delaying the destination schedule $sch_{p_1,d}$ (l. 31-32). This is required because of the constraint (5), as discussed in section 5. If such a schedule can be generated, i.e, bandwidth required in the corresponding cycles along all the links on p_1 is available, the schedule for p_1 is stored in ω_{p_1}. Again, for the second path, by calling procedure `sch_pth` with the schedule at the source end of p_2, i.e., $\omega_{p_1}[(p_1[0], p_1[1])]$, and the direction argument $dir = +1$ (from source to destination), the schedule for all the links of p_2 is generated and stored to ω_{p_2} (l. 33-34). In case a schedule is found, the schedule $\omega_{p_1} \cup \omega_{p_2}$ along the two paths is returned (l. 36); otherwise the next (p_1, p_2) pair is checked (l. 23-34).
If a request is successfully RTSCH'ed, i.e., $\omega^r \neq \phi$ (l. 49), the link bandwidth in the corresponding cycles along the two paths is updated using procedure `upd_bw`. Moreover, the total accepted traffic (*objval*) is updated and the request schedule is stored in *sch*.
The path computation step of the heuristic has the time complexity of $O(|E||P|)$ whereas the schedule generation step run in $O(|E||P||C|^2)$ time. Thus, for a network topology of realistic size the schedule generation step dominates the overall execution time of the greedy heuristic.

6.2 Tabu-search Heuristic

On the one hand, the greedy heuristic is fast and it returns a suboptimal solution, but on the other hand, though the ILP solution is optimal it takes a lot of time to compute. There exists a third approach to generate a close-to-optimal solution in a reasonable time. By employing a metaheuristic such as Tabu-search (TS) the substantial solution space can be navigated much more efficiently.
The flowchart for TS-based heuristic to solve *1+1* RTSCH problem is shown in Fig. 5. Similar to the greedy heuristic, the TS heuristic is called with the set of requests R, request tuples $\mathbb{T}$ and infrastructure graph G. First, `rt_sch` is called to RTSCH with `srtd`(R) and the resulting greedy solution is captured in *init_soln* object and the total accepted traffic for this solution is stored in *bst_objval*. Before proceeding further, the variable initialization is done.
The core of the TS heuristic is a *while* loop that runs for $ITRN_{max}$ iterations and tries to improve the current solution *init_soln* taking into account the recent moves as well as the *tabu* moves. A move to one of the neighbour of *init_soln* is made using procedure `nbr_soln` (explained in the next paragraph). After generating the neighbouring solution *new_soln*, it is checked whether *new_soln* is better than *bst_objval*, i.e, the total accepted traffic for the neighbouring solution *new_soln.objval* is more than *bst_objval*. If yes, the move is added to the collection of tabu moves along with the tenure TT. This results in prohibition of undertaking *move* for the next TT iterations. If *new_soln* is worse than the best score, the move is stored in *moves_used* so that it is not performed again while the search through the current neighbourhood is carried out. In case no better solution is found for $NOIMP\_ITRN_{max}$ iterations, the best score is relaxed by a factor of WF so that the search in another neighbourhood can be performed from the next iteration.
The pseudocode for the procedure `nbr_soln` is shown in Alg. 2. The two kinds of moves that can be undertaken in `nbr_soln` are– i) the request swap move and ii) the path change move, chosen with probability pr_{req} and $1 - pr_{req}$, respectively. If the request swap move is chosen, the order of requests *soln.R* is modified by swapping two randomly selected requests *soln.R*$[i]$ and *soln.R*$[j]$. Otherwise, if the path change move is chosen, a request is randomly selected from *soln.R* it is re-routed and re-scheduled using procedure `pth_chng`. After completing a move, the procedure `nbr_soln` returns the selected move along with the new solution. For both kinds, a *move* is skipped and another one is considered if it was performed recently or it is tabued in the current iteration.

Algorithm 1: Procedure for the greedy-based heuristic to solve the *1+1* RTSCH.

```
Procedure gen_sch($c_1$, $bw^r$, $p$, $\delta c$, $G = (N, E)$):
    $bw_{alc}, sch, c_{nxt} \leftarrow 0, \phi, c_1$;
    for $c$ in $[c_1, C-1]$ do
        if $c == c_{nxt}$ then
            $bw_c \leftarrow \min(bw^r - bw_{alc}, bw^c_{p[-2],p[-1]})$;
            if $bw_c > 0$ then
                $bw_{alc} \leftarrow bw_{alc} + bw_c$;
                $sch[p, p[-2], p[-1], c] \leftarrow bw_c$;
                $c_{nxt} \leftarrow c_{nxt} + \delta c$;
            end
            else
                $c_{nxt} \leftarrow c_{nxt} + 1$;
            end
        end
        if $bw_{alc} >= bw^r$ then
            return $sch$;
        end
    end
    return $\phi$;
end
Procedure greedy_rt_sch($\mathbb{T}^r = (n^r_s, n^r_d, bw^r)$, $G = (N, E)$):
    $P^{n^r_s, n^r_d} \leftarrow$ all_paths($n^r_s$, $n^r_d$);
    for $p_1$ in $P^{n^r_s, n^r_d}$ do
        for $p_2$ in $P^{n^r_s, n^r_d}$ do
            $\delta c \leftarrow \max(1, \mid D^r_{p_2} - D^r_{p_1} \mid)$ ;
            if !disj($p_1$, $p_2$) or $bw^r > bw_c \lceil |C| / \delta c \rceil$ then
                continue;
            end
            for $c_{strt}$ in $[0, C-1]$ do
                $sch_{p_1,d} \leftarrow$ gen_sch($c_{strt}$, $bw^r$, $p_1$, $\delta c$, $G$);
                if $sch_{p_1,d} \neq \phi$ then
                    $\omega_{p_1} \leftarrow$ sch_pth($sch_{p_1,d}$, $p_1$, $dir = -1$);
                    if $\omega_{p_1} \neq \phi$ then
                        $\omega_{p_2} \leftarrow$ sch_pth($\omega_{p_1}[(p_1[0], p_1[1])]$, $p_2$, $dir = +1$);
                        if $\omega_{p_2} \neq \phi$ then
                            return $\omega_{p_1} \cup \omega_{p_2}$
                        end
                    end
                end
            end
        end
    end
    return $\phi$;
end
Procedure rt_sch($R$, $\mathbb{T}$, $G = (N, E)$):
    $objval$, $sch \leftarrow 0$, $\phi$;
    for $r$ in $R$ do
        $\omega^r$ = greedy_rt_sch($\mathbb{T}^r$, $G$);
        if $\omega^r \neq \phi$ then
            $objval$, $sch \leftarrow objval + bw^r$, $sch \cup \omega^r$ ;
            upd_bw($\omega^r$, $G$);
        end
    end
    return $sch, objval$;
end
```



Tabu-search Heuristic
start
bst_objval, sch <- rt_sch(srted(R), T, G); init_soln <- { srted(R), sch, bst_objval};
itrn <- 0; noimp_itrn <- 0; moves_used <- Φ; tabu_moves <- Φ;
itrn > $ITRN_{max}$;
Y
end
N
itrn++;
move, new_soln <- nbr_soln (.., init_soln,..);
new_soln.objval > bst_objval
Y
N
tabu_moves[move] <- itrn+TT noimp_itrn <- 0
init_soln <- new_soln moves_used <- Φ
bst_objval <- init_soln.objval
moves_used.append(move)
noimp_itrn ++
noimp_itrn > $NOIMP\_ITRN_{max}$
N
Y
moves_used <- Φ
noimp_itrn <- 0 bst_objval <- bst_objval*WF

Figure 5: Flowchart depicting TS heuristic to solve the *1+1* RTSCH problem.

Algorithm 2: Procedure to generate a neighbouring solution in the Tabu-search heuristic.

```
Procedure nbr_soln(itrn, soln, moves_used, moves_tabu, G):
    if $rand() <= pr_{req}$ then
        $i, j \leftarrow$ randsel(soln.R), randsel(soln.R) ;
        while $i \neq j$ and $\big((i,j) \notin moves\_used \cup moves\_tabu$ or $itrn > moves\_tabu[(i,j)]\big)$ do
            $i, j \leftarrow$ randsel(soln.R), randsel(soln.R) ;
        end
        swap_req(soln.R, i, j);
        return $(i,j)$, {soln.R, rt_sch(soln.R, G)};
    end
    else
        $i \leftarrow$ randsel(soln.R) ;
        while $i \notin moves\_used \cup moves\_tabu$ or $itrn > moves\_tabu[(i,j)]$ do
            $i \leftarrow$ randsel(soln.R) ;
        end
        sch, objval $\leftarrow$ pth_chng(soln.R[i], G);
        return $(i)$, {soln.R, sch, objval};
    end
end
```

7 Evaluation and Results

In this section, we evaluate the performance of the ILP approach and the two proposed heuristics and then compare them. First, we describe the simulation setup and list the associated parameters. Then, the results comparing the different approaches in terms of various metrics are presented.

7.1 Setup and Parameters

All the simulation experiments are performed on a Intel Xeon server machine with quad-core CPU @ 2.40GHz with 16GB of RAM memory running Ubuntu-16.04 OS. The ILP model for the *1+1* RTSCH problem has been build using the Python API of IBM's ILOG CPLEx called DOcplex (Decision Optimization CPLEX Modeling). Both heuristics are written in Python programming language and the Networkx package is used to generate paths between two nodes in the given graph.
The cycle time T_c by default assumed to be 20μs and the number of cycles per hypercycle are 50.
For the evaluation of the ILP model and two heuristics, we have considered six topologies. Fig. 6 shows the topologies hexagon: `hex1-hex6`, that we have chosen to represent the DetNet infrastructure. Here, `hex1` is a ring topology where two disjoint paths exist between any pair of nodes. `hex2` and `hex3` are ring topologies with one and two diagonals, respectively. `hex4` also contains two diagonals; though, average difference between disjoint points is small as compared to `hex3`. `hex5` contains three diagonals and `hex6` is a full mesh where each node is directly connected to the rest of nodes. We have chosen the above topologies to highlight the impact of connectivity on the average accepted traffic.
Each link in these topologies is a 10G link, thus the bandwidth per cycle $bw_{n_i,n_j,c}$ is 16x1500B packets per 20μs; the node processing delay and link delay (propagation and transmission) d_{n_i,n_j} are 40μs and 80μs, respectively.
For each request $r \in R$, the endpoints–source and destination are randomly chosen from the set of topology vertices and its required bandwidth bw^r is randomly chosen with uniform probability in the range $[50, 120]$ packets (1500B).
Table 2 list the parameters involved in the simulation experiment and their corresponding values/ranges.

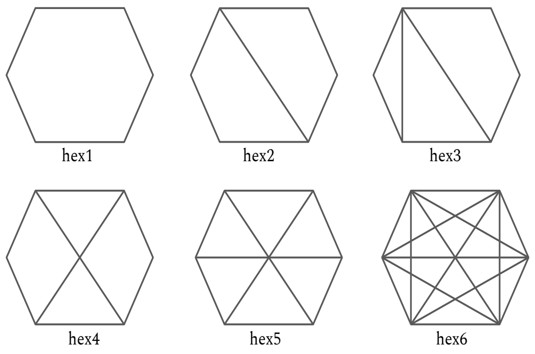


Figure 6: DetNet topologies used for the evaluation.

Table 2: Default values/range of various parameters involved in the evaluation.

Parameter	Value or range	Units
Topology	`hex1`, `hex2`, `hex3`, `hex4`, `hex5`, `hex6`	-
Request batch size ($\mid R \mid$)	{30, 50, 70}	numbers
Request bandwidth size (bw^r)	$[50, 120]$	packets
Total cycles per hypercycle ($\|C\|$)	$\{9, 18, 27, 54, 108, 216\}$	numbers
Cycle time (T_c)	$\{20, 40, 80, 160, 240, 480\}$	μs
Cycle max. bandwidth ($bw^{max}_{n_i,n_j}$)	$\{8, 16, 32, 64, 128, 256\}$	packets
Node processing delay (d_{n_i,n_j})	40	μs
Links delay (d_n)	80	μs

7.2 Performance and Calculation time Trade-off

The *1+1* RTSCH problem is solved using the CPLEX solver and the two heuristics; greedy (`GRD`) and tabu-search (`TS`). Fig. 7 shows the percentage of total for $|R| = 30$ with `hex1` topology. The low traffic acceptance for the three approaches results from the rejection of many requests due to non-adherence of the *spacing constraint* in `hex1`. On one hand, `GRD`'s performance is the lowest as it just returns the first feasible solution with greedy bandwidth allocation for each request. On the other hand, the ILP approach has the highest performance because it return the optimal solution by searching through the entire solution space. Although, the performance of the `TS` approach is inferior to the `ILP` approach it still outperforms the `GRD` approach, especially for large problem sizes as will be highlighted in the next section.
The `ILP` approach though is the most optimum approach, it may not suitable for large size instances of the *1+1* RTSCH problem. Even for a reasonable problem size, RTSCH'ing $|R| = 30$ on `hex1` topology, the CPLEX calculation exceeds 30 minutes. The calculation time increases rapidly when the topology is changed from `hex1` to `hex2`; in fact, it runs into several hours for $|R| \geq 30$ with the `hex2` topology. This is because as the connectivity of the topology increases, the solution space explodes due to the increase in the number of paths between two nodes thus resulting in very high calculation time. Fig. 8 depicts the evolution of best bound (Bst-Bnd), objective value (Obj) and integer (Bst-Int) solution with time for $|R| = 30$ and the `hex2` topolgy. Here Bst-bnd and Obj are the CPLEX's best objective function value achievable and current node's objective values, whereas Bst-int is the objective value of the best integer solution. It can be observed that most improvements in the integer solution's objective value are found in early few minutes and there is a marginal improvement afterwards; nonetheless, the CPLEX calculation time spans several hours. As an alternative to the ILP approach, the two heuristics scale well with the problem size that return a solution within a few seconds.
Table 3 lists the CPLEX calculation times for the *1+1* RTSCH problem for $|R| = \{30, 50, 70\}$ with the `hex1` topology. The calculation time for `TS` is higher as compared to `GRD`; though, the maximum calculation time (for $|R| = 70$) is still within a few seconds. We also observed that the calculation time for `GRD` and `TS` does

not vary with the topology. The ILP approach, on the other hand, is infeasible for all the problem sizes, except for $|R| = 30$ and the `hex1` topology, thus the calculation times are not indicated here. Consequently, it is unfit to solve the reasonable size instances of the *1+1* RTSCH problem and the heuristics should be employed for such instances.

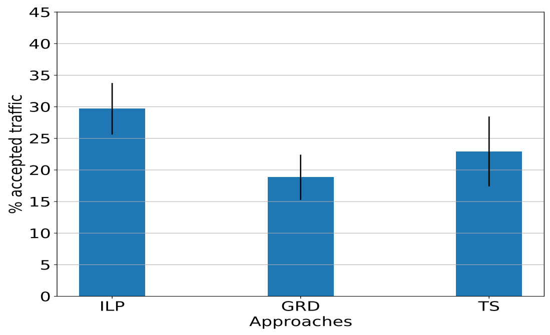


Figure 7: Comparison of the ILP and heuristics in terms of the percentage of the total accepted traffic.

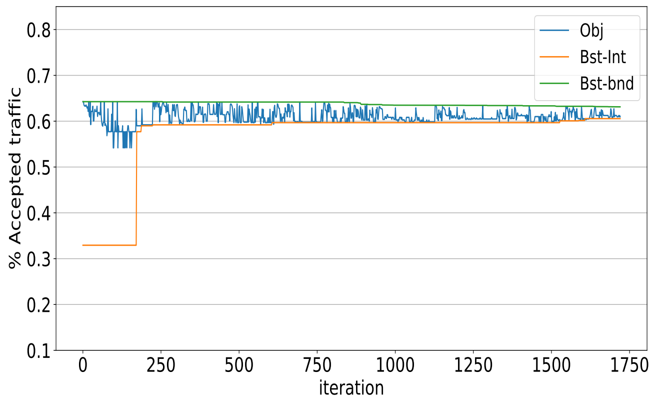


Figure 8: Evolution of the various ILP parameters with time.

Table 3: Approximate calculation time for `GRD` and `TS` with the `hex1` topology.

$\|R\|$	`GRD` time(s)	`TS` time(s)
30	0.004	2.4
50	0.008	4.0
70	0.01	7.0

7.3 Heuristic Performance

Before presenting the performance results for the heuristics, we present the results showing the percentage of the total acceptable traffic for the topologies `hex1-hex6`. Fig. 9 shows the percentage of total accepted traffic satisfying only the spacing constraint assuming each cycle has the maximum bandwidth $bw^{max}_{n_i,n_j}$. These results indicate that as the connectivity of a topology increases, from `hex1` to `hex3`, the average number of paths between any two nodes increases. As the number of paths increases, the number of requests that adhere to the spacing constraint increases resulting in an increase of the total accepted traffic.

Next, we report the percentage of the total accepted traffic taking into account the actual cycle bandwidth $bw_{n_i,n_j,c}$ instead of maximum cycle bandwidth $bw^{max}_{n_i,n_j}$. The total accepted traffic is the sum of the bandwidth requirements of all requests that can be routed and scheduled. Fig. 10 shows the percentage of the total

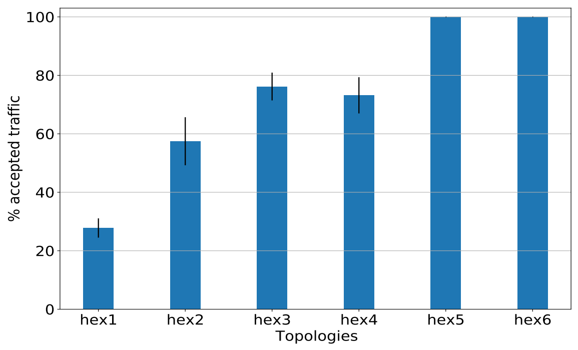


Figure 9: The percentage of the total accepted traffic for different hex topologies, considering only the spacing constraint.

accepted traffic for the sets of requests with different sizes. The total accepted traffic for the two heuristics decreases with increasing $\mid R \mid$ because increasingly cycles are consumed for the accepted traffic and lesser bandwidth per cycle is left for the other requests. The total accepted traffic also increases as the connectivity of the graph increases as the average number of possible paths for each request increases. It can also be observed that `TS` outperforms `GRD` in all the cases. For instance, in `hex4` and $|R| = 50$, the % accepted traffic for `TS` is better as compared to `GRD` by 8%.

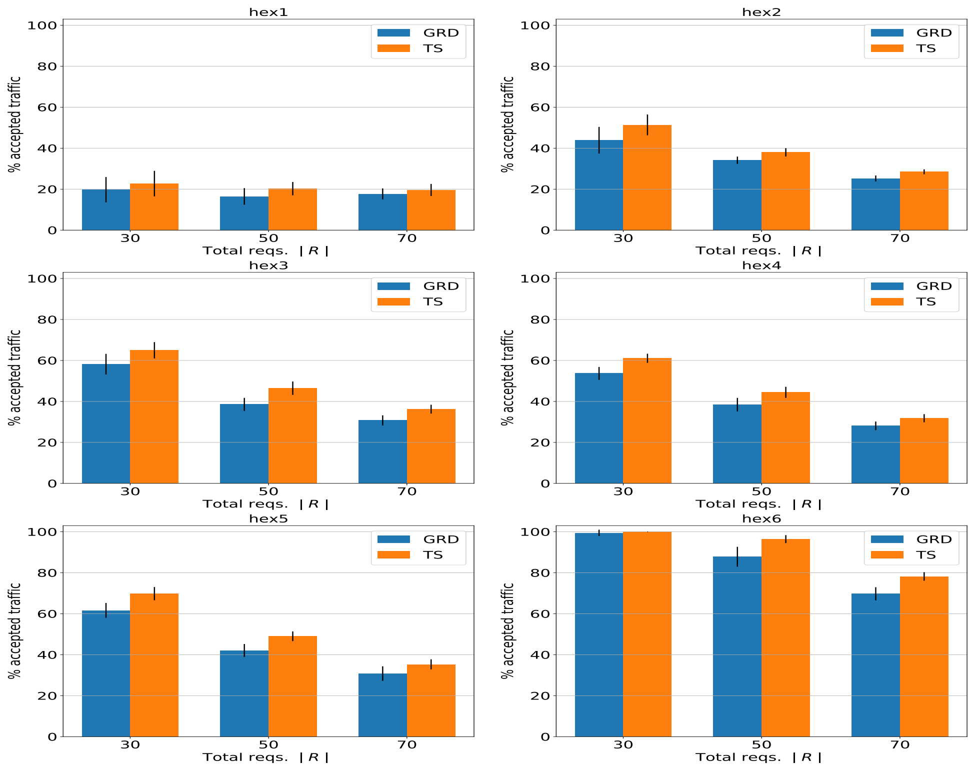


Figure 10: The percentage of the total accepted traffic with $|R|$ for different hex topologies.

7.4 Impact of T_c

Next, the influence of the cycle time on the performance of the two heuristics is reported. The COST239 network is selected for this evaluation. We selected this network as it has sufficient connectivity as shown in Fig. 11. Each node is connected to at least four neighbours, thus providing sufficient redundancy for $1+1$ protection.
Fig. 12 shows the percentage of the total accepted traffic for $\mid R \mid = 50$ on the COST239 network for cycle times between 20-480μs. It can be observed that as the cycle time is increased, the total accepted traffic increases for all topologies. As the cycle time increases, the total bandwidth of each cycle increases (though the total number of cycles in a hypercycle decreases), more requests can be mapped to the same cycle thus improving the total accepted traffic.

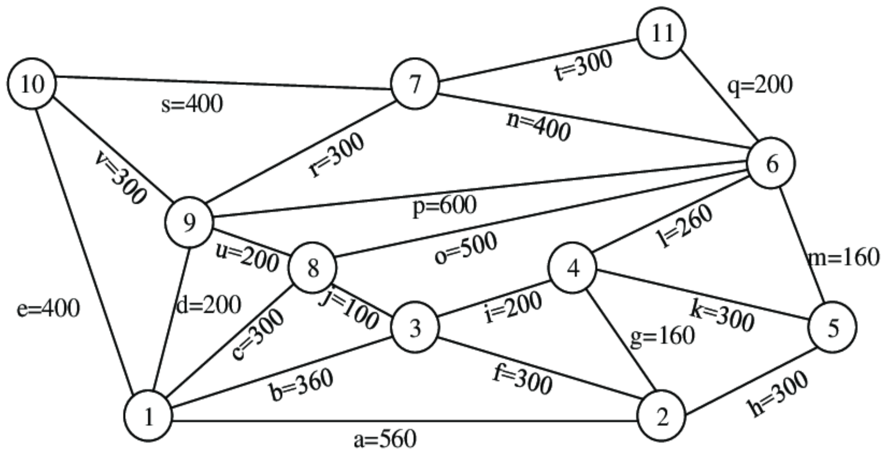


Figure 11: Topology for the pan-European COST239 network. The label corresponding to each link is its length (in kms).

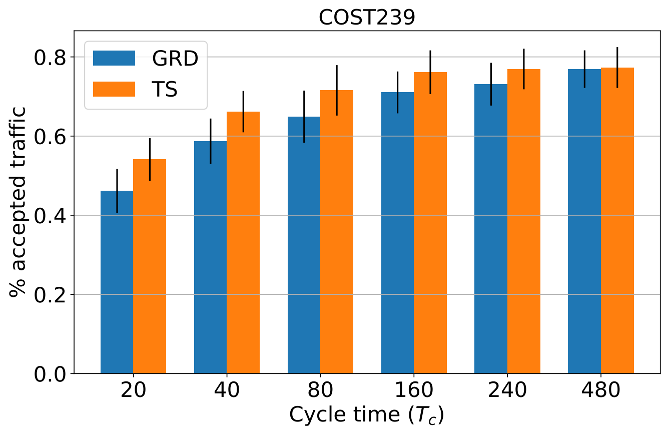


Figure 12: The influence of T_c on the percentage of the total accepted traffic for $|R| = 50$ for the COST239 network.

The cycle time also impacts the end-to-end delay between the source and the destination. As described in [?], the end-to-end delay d_{e2e} on path p with N nodes (including source and destination) and cycle time T_c are related as : $d_{e2e} = (N-1)d^{tx}_{n_i,n_j} + (N-2)(d^{proc}_{n_j} + 2T_c)$. The average end-to-end delay for $\mid R \mid = 50$ between the endpoints of the accepted requests on the six topologies is depicted in Fig. 13. As expected, the end-to-end delay increases with increasing cycle time as packets have to be queued for a longer time.

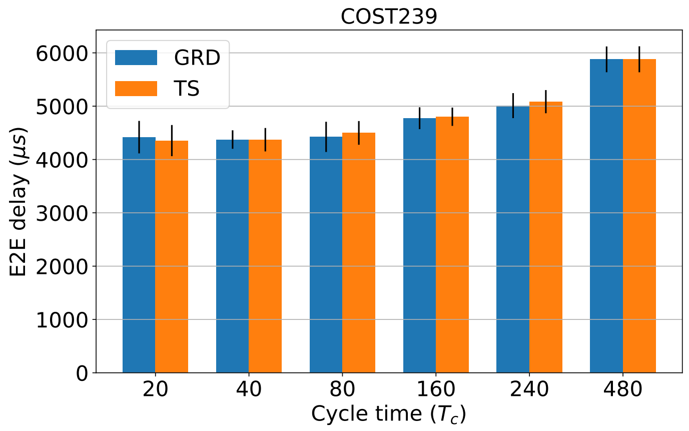


Figure 13: The influence of T_c on the average end-to-end delay for $|R| = 50$ for the COST239 network.

8 Conclusion

Packet losses due to network congestion can be addressed using DetNet mechanisms such as CSQF. To guarantee protection against failures in the network, a scheme like *1+1* is required. The problem of *1+1* RTSCH for DetNets is investigated in this paper. To allow reliable recovery, the packet schedules on the two selected paths should take into account the end-to-end delays. To this end, we formulate the *1+1* RTSCH problem as an ILP. We propose two additional approaches: greedy and Tabu-search heuristics, that are scalable for relatively large problem sizes. Performance evaluation of these approaches highlights the influence of the topology and request set size on the average accepted traffic. Although the ILP approach is exact, it is infeasible beyond small size problem instances. The `GRD` and `TS` heuristic trade-off on the solution quality ($< 10\%$) for its speed such that reasonable size problem instances can be solved in a few seconds which the ILP approach would require several hours to solve.
Moreover, the trade-off of choosing T_c on total accepted traffic and the end-to-end delay is also discussed.
Professional media flows can be reliably transported using CSQF in a DetNet. The future work will include the modeling of RTSCH for media services that are represented by Virtual Media Function Forwarding Graphs (VMF-FG) in contrast to simple services consisting of just source-destination pairs.

9 Acknowledgements

This research was funded by the Flemish FWO SBO S003921N VERI-END.com (Verifiable and elastic end-to-end communication infrastructures for private professional environments) project, by the FWO project under grant agreement #G055619N and the Flemish Government under the ”Onderzoeksprogramma Artificiele Intelligentie (AI) Vlaanderen”.