Online Identification of Groups of Flows Sharing a Network Bottleneck

David A. Hayes, *Senior Member, IEEE,* Michael Welzl, Simone Ferlin, David Ros, and Safiqul Islam

***Abstract*—Most Internet hosts today support multiple access technologies and network interfaces. Multipath transport protocols, like MPTCP, are being deployed (e.g., in smartphones), allowing transparent simultaneous use of multiple links. Besides providing increased resilience to link failures, multipath transports may better exploit available (aggregate) capacity across all interfaces. The safest way to ensure fairness is to assume that any subflows of a multipath end-to-end connection may share bottleneck links, but knowledge of *non*-shared bottlenecks could allow multipath senders to exploit more capacity without being unfair to other flows. The problem of reliably detecting the existence of (non)-shared bottlenecks is not trivial and is compounded by the fact that bottlenecks may change due to traffic dynamics. In this paper we focus on practical methods to reliably group flows that share, possibly dynamic, bottlenecks online and in a passive manner (i.e., without injecting measurement traffic). We introduce a novel dynamic clustering algorithm that we apply to update our previous shared bottleneck flow grouping (SBFG) method standardized by the IETF, based on delay statistics. We also adapt an offline SBFG method based on wavelet filters to enable it for online operation. These SBFG methods are evaluated by a simple testbed, rigorous simulation and real-world Internet experiments in a testbed comprised of multihomed hosts. Our results suggest that there is no clear winner, and selection of the "best" SBFG method will have to consider tradeoffs regarding accuracy, lag, and application requirements.**

***Index Terms*—Shared bottleneck detection, Internet congestion control, dynamic clustering algorithms, multipath congestion control.**

I. Introduction

INTERNET congestion control is based on the assumption that any number of end-to-end flows may share a common bottleneck; if they really do, and which flows there are, is generally unknown. This is perhaps most prominently visible in the design of MultiPath TCP (MPTCP) [2]: MPTCP tries to efficiently utilize the available capacity of multiple end-to-end paths, yet its congestion control is carefully tuned to be able to fairly compete with other end-to-end flows supposing all flows may traverse a shared bottleneck.

While MPTCP's congestion control enables safe usage of multiple paths without knowledge of shared bottlenecks, it could leverage such knowledge to be slightly more aggressive when flows do *not* share a bottleneck. In our prior work, we have used an earlier version [3] of one of the algorithms presented in this paper to show that MPTCP can indeed be tuned to be more aggressive when it *infers* that there are no shared bottlenecks between its flows [4]. This affects the global fairness to other flows—but the per-flow fairness goal itself is also somewhat elusive [5], and a fairness discussion of MPTCP is not the focus of this paper (as are details of implementing our algorithms inside MPTCP, for which we refer to [4]). More importantly, without knowledge about flows sharing bottlenecks, *such options do not even exist*.

Having practical means to determine shared network bottlenecks *online*, i.e. while data transfers are ongoing, would allow for a great variety of network improvements. As another example, ensembles of flows could be controlled jointly, with multiple benefits including reduced overall latency and better control of the relative fairness between them—for UDP-based video streams [6, 7] or TCP [8]. These potential benefits motivated the Internet Engineering Task Force (IETF) to standardize a Shared Bottleneck Detection (SBD) mechanism for RTP flows [9] that is based on summary statistics. A detailed introduction to the algorithm and evaluation results of an earlier version are available in [10].

In this paper, we make the following contributions:

1) We show the first evaluation results of the IETF-standardized Shared Bottleneck Flow Grouping (SBFG)[1] mechanism in [9].
2) We replace the "divide and conquer" threshold based method of grouping flows in our SBFG mechanism from [9] with a novel dynamic clustering algorithm. This dynamism is important here because bottlenecks in the Internet are dynamic. A practical online SBFG mechanism therefore also needs to be dynamic, efficiently constructing changing numbers of groups containing changing numbers of flows. This clustering method may also be applicable to grouping problems in other fields where the number of clusters may vary over time.
3) We also compare our results with results from a notable, and very different, example of prior work—pairwise cross-correlation of wavelet filtered One Way Delays (OWDs) [11]—as well as ground truth (where it is known). To the best of our knowledge, no such comparison exists in the literature. Our results suggest that all evaluated methods perform well, with tradeoffs with respect to (a) false positives / false negatives, (b) time lag before detection. Selection of the "best" method will depend on application and implementation considerations.
4) Finally, we modify the wavelet-based mechanism from [11] to enable it to operate online and in a pas-

The authors were partially funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the authors.

The authors would like to thank Thomas Dreibholz for helping with the NorNet testbed, and Özgü Alay for her advice concerning these experiments.

[1]This term better explains the function of the mechanism, but is equivalent to SBD in [9, 10].

TABLE I
SUMMARY OF PRIOR WORK NOTING THE KEY DISCRIMINATING METRIC AND GROUPING MECHANISM.

Mechanism					Online	Passive
Metric		Ref	Grouping	Notes		
Location		[12]	Most significant 24 bits of IP address	Granularity was not fine enough to be useful. Does not detect flows sharing an actual bottleneck, only those that have potential to due to their presumed location	Yes	Yes
Throughput		[13]	Quantile-quantile plots, best fit ≈ 1	Long term distributions (Grouping used 10s of hours of data)	No	Yes
		[14]	Change in throughput when flow leaves and correlation coefficient > 0.28	Relies on greedy sources, consistent background traffic, and 30-1000 s of samples	No	Yes
Packet loss		[15]	Flows with common packet loss between received packets	Slightly different method for shared receivers and senders. Does not perform well, and worse with Random Early Discard (RED)	Yes	No[a]
		[16]	Flows experiencing packet loss within the same time window	Multipath TCP or SCTP congestion control operation specific mechanism	Yes	Yes
Congestion interval variance		[17]	Variance of flow congestion interval	Multipath TCP specific. Congestion as determined by packet loss or excessive delay	Yes	Yes
Delay	RTT[b]	[18]	Cross-correlation $>$ auto-correlation	Cross-correlation based on [15]	Yes	Yes
	Inter-packet	[19]	Incrementally groups flows to reduce entropy	Rèyis entropy. Presumes flow groups are static.	Yes	Yes
		[20]	By hand	Skewness and kurtosis.	No	No
	OWD[c]	[11]	Correlation coefficient > 0.512	Wavelet filtered. Optimal filter coefficients calculated from all the measurements for the entire duration of the test; minimizing the maximum mean square error assuming non-white noise.	No	No[a]
		[15]	Cross-correlation $>$ auto-correlation	Better and quicker than loss in the same paper	Yes	No[a]
		[21]	Difference between sorted singular values $>$ threshold	Covariance matrix of average packet delays filtered using Singular Value Decomposition (SVD)	No[d]	Yes
		[9, 10]	Divide-and-conquer threshold based splitting for each summary statistic	Shape delay-based summary statistics as well as packet loss[e]	Yes	Yes
		This paper	Two new methods explored: dynamic iterative clustering, and correlation coefficient > 0.5	Dynamic clustering using the summary statistics defined in [9]. Changes the offline algorithm in [11] to dynamically choose wavelet filter coefficients and handle passive monitoring	Yes	Yes

[a] Methods that were active, but authors suggest would work passively.
[b] Round Trip Time.
[c] One Way Delay.
[d] Making it online is discussed in [21].
[e] Allows for additional statistics, e.g. ECN based.

sive fashion so that we can compare it with the other mechanisms.

All of the investigated SBFG mechanisms are traffic agnostic. Taking advantage of particular traffic sending patterns could prove useful in some cases, but is beyond the scope of this work, which looks at the more general case. Being online and passive already makes the mechanisms very broadly applicable, as they only need to observe existing incoming and outgoing packets. Additionally, being traffic agnostic means that they do not need to be adjusted for different usage scenarios but can work "out of the box". These factors can facilitate deployment.

After a look at prior work in section II, we introduce our divide-and-conquer and dynamic clustering methods based on summary statistics in section III and our online passive wavelet-filtered correlation based method in section IV. Then, after discussing the different tradeoffs involved in selecting a particular mechanism (section V), we evaluate these methods in section VI through a simple testbed (section VI-A2), simulation studies, where the ground truth is known (section VI-B3), and Internet-based experiments using operational cellular networks (section VI-C3). In section VII we draw together our findings, discussing the relative merits of the summary statistics based methods with the wavelet-filtered correlation methods.

II. BACKGROUND

There have been several proposals to identify groups of flows that share a bottleneck. They can be divided into online and offline, active and passive, and they can be categorized by the discriminating metric. As previously explained, our interests are with online methods, particularly passive methods, and we investigate a number of grouping techniques. Table I summarizes the prior work in the area noting key aspects of the proposed SBFG mechanisms. As the table shows, in related work, the mechanisms in [12, 16–19] and in our own earlier work [9, 10] are applicable both online and passively, i.e. without the need to produce probe traffic or requiring applications to send data following a specific traffic pattern.

Just like the mechanisms that we propose here, many (8 of the 14 in table I) of the methods in the literature use packet delay as their base discriminating metric since this is regular and timely information, easy to obtain and hence suitable for practical use. The delay-based methods differ in how they deal with the noise in the delay signal and how the flows are grouped. Among them, only [9, 10, 18, 19] and the methods in the current paper work online and passively; furthermore, the authors of [15] suggest that their online method would work passively as well. The authors of [18] use the Round Trip Time (RTT), which is inherently more noisy than the OWD or inter-packet delay taken at the receiver because it is affected by queues along the backward path.

TABLE II
TERMINOLOGY USED IN THE DESCRIPTION OF SUMMARY STATISTIC CALCULATION (MOSTLY FOLLOWS CONVENTIONS IN [9]).

T	Base time interval over which statistics are calculated
N	Number of T intervals used in some calculations
M	Number of T intervals used in some calculations
F	Number of T intervals used in the flat portion of the weighted moving average calculation
skew_est	A measure of the skewness in the received packets' OWD (One Way Delay) distribution
var_est	A measure of the variability of the received packets' OWD distribution—Mean Absolute Deviation (MAD)
freq_est	A measure of the oscillation over time of the average OWDs
pkt_loss	The proportion of lost packets measured over NT
d_i	Measured OWD of packet i
$\bar{d}$	Mean of the measured OWD of packets in T
K	Number of packets received in T
p_s, p_f, p_d, and p_{mad}	Grouping thresholds for skew_est, freq_est, pkt_loss and var_est respectively (used in algorithm 1).
p_v	Threshold used in the freq_est calculation (see eq. (4)).

Only our own prior work [9, 10] is designed to work with dynamic bottlenecks, though it may be possible to modify some of the other techniques to work in a dynamic way (in particular [11], which we accordingly extend in this paper).

III. MECHANISMS BASED ON SUMMARY STATISTICS

Hayes *et al.* [10] describe a method for grouping flows that share a bottleneck using summary statistics which describe the shape of the distribution of OWD. Summary statistics provide resistance to short-term fluctuations, helping to mitigate difficulties the noisy delay signal presents. Key improvements to the summary statistics used and the overall method in [10] are outlined in RFC 8382 [9]. They include a more robust variability estimator and better time averaging of the statistics summarized in section III-A. This paper presents the first comparative evaluation of [9] and also extends it, proposing in section III-C a novel iterative online clustering algorithm that draws inspiration from the iterative algorithm in [22]. This improves the accuracy and robustness of the [9, 10] "divide-and-conquer" grouping algorithm briefly described in section III-B.

A. Summary statistic calculations

In this section we briefly describe the summary statistics specified in [9]. Table II summarizes the notation used in the description below. OWD[2] is used as the base metric, with packet loss used as a supplementary measure[3]. Particularly:

$$\bar{d} = \frac{\sum_{k=1}^{K} d_k}{K} \quad , \qquad \bar{\mathbf{d}} = \frac{\sum_{m=1}^{M} \bar{d}_m}{M} \tag{1}$$

where $\bar{d}_1$ is the average OWD in the interval T which has just completed, and $\bar{d}_M$ is the average OWD of the $(M-1)^{\text{th}}$ interval of duration T prior to the interval just completed. From the base metric $\bar{\mathbf{d}}$, we calculate as an estimate of the skewness:

$$\text{skew\_est} = \frac{\sum_{m=1}^{M} \text{skew\_base\_T}_m}{\sum_{m=1}^{m} K_m} \tag{2}$$

$$\text{where skew\_base\_T}_m = \sum_{k=1}^{K} s_k \ , \quad s_k = \begin{cases} +1 & d_k < \bar{\mathbf{d}} \\ -1 & d_k > \bar{\mathbf{d}} \end{cases}$$

Apart from being used to differentiate flows sharing and not sharing a bottleneck, skew_est provides a protocol independent indication of path congestion, since the degree and sign of skewness change in relation to the bottleneck load (see [10]). We also calculate the Mean Absolute Deviation (MAD) as a measure of the variation in OWD:

$$\text{var\_est} = \frac{\sum_{m=1}^{M} \text{var\_base\_T}_m}{\sum_{m=1}^{M} K_m} \tag{3}$$

$$\text{where var\_base\_T}_m = \sum_{k=1}^{K} \left| d_k - \bar{d}_m \right|$$

A measure of the fluctuation of OWD, calculated as significant mean delay crossing events, is given by:

$$\text{freq\_est} = \frac{\sum_{n=1}^{N} |x_n|}{N} \tag{4}$$

$$\text{where } x_n = \begin{cases} -1 & \bar{d}_n < \bar{\mathbf{d}} - (p_v \times \text{var\_est}) \wedge x_h = 1 \\ 1 & \bar{d}_n > \bar{\mathbf{d}} + (p_v \times \text{var\_est}) \wedge x_h = -1 \\ 0 & \text{otherwise} \end{cases}$$

h is the highest index $i \in [0, n-1]$ such that $x_i \neq 0$, and $(p_v \times \text{var\_est})$ determines how much $\bar{d}_n$ needs to cross $\bar{\mathbf{d}}$ by to be significant. As in [3], $p_v = 0.7$ in these experiments.

Since packet loss is in general an infrequent event, it is used as an additional metric to supplement skew_est at very high loads when skew_est is not a good indicator of congestion (see [10]):

$$\text{pkt\_loss} = \frac{\sum_{n=1}^{N} \text{packet\_loss\_T}_n}{\sum_{n=1}^{N} K_n} \tag{5}$$

here packet_loss_T$_n$ is the number of lost packets in the n^{th} interval of duration T, prior to the measurement interval.

The algorithm begins to make decisions after M samples (MT s), but will not reach its full accuracy until $2N$ samples due to the calculation of freq_est. This paper uses $N = 50$ and $M = 30$ as recommended in [9]. The piecewise linear weighted moving average for var_est and skew_est described in [9] is used to help reduce lag. These statistics are used in both the divide-and-conquer mechanism described next and the online iterative clustering method presented in section III-C.

B. RMCAT mechanism (rmcatSBD)

RFC 8382 [9] describes a simple divide-and-conquer threshold based algorithm to group flows sharing a common bottleneck (see algorithm 1). It was published as an output of the Real-Time Media Congestion Avoidance Techniques (RMCAT) group in the IETF; hence we call it the "RMCAT" mechanism. The algorithm first uses skew_est and pkt_loss to infer whether flows are experiencing congestion or not. This

[2]The summary statistic calculations are relative to the mean OWD. This means that there is no need to synchronise endpoint clocks. See [section IIIC, 10] and [section 1.2.2, 9] for a more detailed explanation.

[3]Potentially Explicit Congestion Notification (ECN) could also be used, however as it is not universally and consistently available at every router through the Internet, it is difficult to use it as a discriminator. For example, two flows may share a common bottleneck even though one has ECN marking and the other does not, simply because one flow has a low threshold ECN marking router in its path and the other does not.

```
Input: $X$                                              Set of flows to be grouped
Input: $S$                     Set of skew_est, where $s_1$ =skew_est for flow 1
Input: $V$                       Set of var_est, where $v_1$ =var_est for flow 1
Input: $F$                     Set of freq_est, where $f_1$ =var_est for flow 1
Input: $L$                     Set of pkt_loss, where $l_1$ =pkt_loss for flow 1
First find flows transiting a bottleneck
foreach flow $(x \in X)$ do
    Flow congestion thresholds $c_s, c_h, p_l$: skewness, skewness hysteresis, and
    packet loss
    if $((s_x < c_s \vee (s_x < c_h \wedge x \in B) \vee l_x > p_l$ then
        $x \in B$                          flow in set of flows transiting bottlenecks
    else
        $x \in U$                      flow in set of flows not transiting bottlenecks
    end
end
Successively partition groups according to metric
$G_1 \leftarrow$ Partition($F, B, p_f$, "A")
$G_2 \leftarrow$ Partition($V, G_1, p_{\text{mad}}$, "P")
$G_3 \leftarrow$ Partition($S, G_2, p_s$, "A")
$G_4 \leftarrow$ Partition($L, G_3, p_d$, "P")    Note: Only flows with $l_x > p_l$ are
  partitioned, so often no fourth level partitioning is necessary.
Input: $M$                                              Measurement statistics
Input: $G$                                              set of groups to partition
Input: $T$                   Threshold for that measure using notation from [9]
Input: $C$                                          Type of threshold comparison
def Partition($M, G, T, C$)
    $N \leftarrow \{\}$
    foreach group $g \in G$ do
        $\phi \leftarrow$ Sort($M(g)$)
        if $C$ == "A" then
            $\hat{N}_i \leftarrow$ partition where diff($\phi$) $> T$
        else
            proportional threshold
            $\hat{N}_i \leftarrow$ partition where diff($\phi$) $> T\,\phi$
        end
    end
    $N \leftarrow \{\hat{N}_1, \ldots, \hat{N}_I\}$
    return $N$
```

Algorithm 1: Divide and conquer type threshold based grouping algorithm from [9].

is important, since if a flow experiences no congestion along the path, it is not traversing a bottleneck and the summary statistics do not contain information relevant for grouping. All flows experiencing congestion by this measure are placed in a single group, which is progressively divided into subgroups according to each summary statistic.

This method is simple and functional, however, it treats each statistic separately and does not take the collective statistical proximity of the flows into account. Next, we propose a more advanced flow grouping that is able to take advantage of the multidimensional nature of collective statistics.

C. Cluster based grouping mechanism (dcSBD)

Clustering algorithms are used in many fields and are a very large area of research in their own right. Their application to shared bottleneck detection has the following requirements:

1) There is an unknown and variable number of groups.
2) Groups are not static, but dynamic, changing as the network conditions change. No prior work surveyed in section II considers this.
3) Grouping must be timely[4].

In a complex network with multiple bottlenecks it is possible that a flow could share more than one bottleneck and thus be a member of more than one group of flows. This is not catered for in this algorithm, or any of the prior work in section II.

[4]Algorithms that iterate an indeterminate number of times to achieve stable groupings are unlikely to be suitable.

These requirements preclude commonly used algorithms that require a fixed number of groups such as k-means clustering. Although the system is dynamic, the shared bottleneck state, as measured by summary statistics, is unlikely to change significantly from one measurement interval to the next. This means that the best starting point for the next bottleneck grouping is the current grouping, suggesting an iterative method may be efficient.

Our method is inspired by [22] (which has similarities to the iterative entropy based clustering used in [19]). *Chinese Whispers* is a fast iterative method (linear with the number of nodes to be clustered) that is used in the natural language processing field. For SBFG, each node in the graph represents one flow, with the weights of the edges connecting the nodes representing the statistical closeness of one node to another node as a function of the grouping measures (skew_est, var_est, etc). We extend the *Chinese Whispers* nearest neighbor iterative grouping ideas to work in a dynamically changing environment which usually only has a relatively small number of nodes. The key extensions are:

1) The clustering result of the previous measurement interval is used as the starting point for the algorithm in the new measurement interval.
2) To allow the clustering to handle a dynamically changing environment[5], from each group we remove the node that is furthest from its closest neighbor before we begin to iterate on the updated graph.
3) We only iterate for a small number of iterations after each edge update[6] (typically just 2, i.e. one step back, two steps forward). This takes advantage of the assumption that the groupings do not change greatly between edge weight calculation intervals[7], and uses the measurement intervals as part of the iterative process.
4) In this application we model edge weights (node closeness) using:
    a) Euclidean distance between nodes (if the nodes were points in N-dimensional space, where N is the number of statistical measures), and
    b) Inverse sum of the squares of the difference between the connected nodes' statistics.

    Our tests have found that the inverse sum of the squares method facilitates the best grouping in this application.

The details of the clustering algorithm are outlined in algorithm 2. While we apply it specifically to grouping flows that share a bottleneck, the algorithm may be applicable to dynamic grouping problems in other fields.

IV. A Mechanism Based on Wavelet-Filtered Delay Signal Correlation Coefficients

Having introduced the summary statistics that feed into the RMCAT mechanism, the RMCAT algorithm itself and our new clustering mechanism, we now turn to the extension of the

[5]In this case the measured network path statistics change as the network dynamics change.

[6]In this application, the measurement interval for the statistics.

[7]In this application, it is the assumption that the bottleneck grouping does not change greatly between the short measurement intervals.

```
Input: C                                                          previous clusters
Input: gClass                                               previous classifications
$B \leftarrow$ Set of flows that transit a bottleneck
$U \leftarrow X \setminus B$                         set of flows that do not transit a BN
graph $\leftarrow$ MakeGraph($S, L, V, F$)
gClass $\leftarrow$ PrimeClass(gClass, $C, B, U$,graph)
for $I$ iterations do
    Recently separated flows are tested last
    foreach flow transiting a BN ($b \in B$) do
        Flow $b$ adopts the class of the closest other flow
        gClass($b$) $\leftarrow$ gClass(ClosestFlow (graph,$b$))
    end
end
Flows with the same class are clustered together
C $\leftarrow$ sets of flows with same class
return gClass,C

Companion Functions:
def MakeGraph(gClass, $B, U, S, L, V, F$)
    This reduces the 4 dimensional relationship between flows to a single value
    using GraphMeasure()
    foreach flow $b_i | i = 1 \ldots n, B = \{b_1, b_2, \ldots, b_n\}$ do
        foreach flow $b_j | j = i \ldots n, B = \{b_1, b_2, \ldots, b_n\}$ do
            graph($i, j$) $\leftarrow$
              GraphMeasure($[s_i, s_j], [v_i, v_j], [f_i, f_j], [l_i, l_j]$)
            graph($j, i$) $\leftarrow$ graph($i, j$)
        end
    end
def PrimeClass(gClass, $C, B, U$,graph)
    In terms of the graph, flows are nodes
    Each flow that transits a BN starts with a unique class
    foreach flow not transiting a BN ($u \in U$) do
        gClass($u$) $\leftarrow$ 0                                   No BN classification
    end
    foreach flow transiting a BN ($b \in B$) do
        if gClass($b$) = 0 then
            gClass($b$) $\leftarrow$ unique class
        end
    end
    foreach $c \in C$, i.e.cluster of flows do
        Separate furthest flow in cluster
        $f \leftarrow$ furthest flow by graph($c_1, c_{!1}$)
        if furthest flow ($f$) by measure, significantly far then
            gClass($f$) $\leftarrow$ new unique class
        end
    end
    return gClass
def GraphMeasure($[s_1, s_2], [v_1, v_2], [f_1, f_2], [l_1, l_2]$)
    Options include:
        a measure based on the Euclidean distance
        a measure based on the inverse sum of the squares
        return measure
def ClosestFlow(graph,b)
    The closest flow, by graph measure, to b, excluding itself
    $f \leftarrow$ index of closest graph($b$, !$b$))
    Sanity check closeness to avoid the lone flow problem
    if graph($b, f$) isn't too distant then
        return $f$
    else
        return $b$                                                         no change
    end
```

Algorithm 2: Interactive dynamic clustering algorithm. Note that packet loss is always used—although inaccurate when very small, it also only makes an insignificant contribution to the clustering measure when very small.

```
Input: $d_f$                          OWD measurement of packet from flow $f$
foreach packet received do
    incrementally calculate skew_est as in eq. (2)
    $\hat{d}_f \leftarrow$ SampleHold($d_f$,stepsize)          Allow passive measurements
end
$S \leftarrow \hat{d}_f$                                    $S$ is the delay signal to be filtered
if new sample interval then
    foreach flow $f \in F$ do
        D$_f \leftarrow$ StepWaveletDecomp($\hat{d}_f$)
        Online calculation of wavelet filtering thresholds
        if new $T$ then
            if resetthresholds $\wedge$ skew_est$_f < c_s$ then
                $w_{\mathrm{th}} \leftarrow$ CalcWaveletThresh($D_f$)
            else
                if skew_est$_f > c_h$ then
                    resetthresholds = 1
                end
            end
        end
        $\hat{D}_f \leftarrow$ FilterWaveletDetails($D_f, w_{\mathrm{th}}$)
        $\hat{S}_f \leftarrow$ StepRecomposeSignal($\hat{D}_f$)
    end
end
similar to the standard DCW algorithm
if end of correlation test interval then
    foreach flow $f \in F$ do
        foreach flow $g \in F, g \neq f$ do
            $\rho \leftarrow$ CorrCoef($\hat{S}_f, \hat{S}_f$)
            if $\rho >$ corr_thresh then
                flow $f$ and flow $g$ are in the same group
            else
                flow $f$ and flow $g$ are in different groups
            end
        end
    end
end

def CalcWaveletThresh($D$)
    return $w_{\mathrm{th}}$=median($|D|$)
def FilterWaveletDetails($D$,$w_{\mathrm{th}}$)
    Soft thresholding (see https://se.mathworks.com/help/wavelet/ref/wthresh.html)
    return $\hat{D}$ = sign($D$) $\cdot$ $[(|D| - w_{\mathrm{th}})]^+$
                    where $[x]$+ limits $x$ to positive values, $[0, \infty)$
def StepWaveletDecomp($\hat{d}_f$)
    Matlab Simulink stepping through wavelet decomposition (Daubechies 6)
    return wavelet decomposition
def StepRecomposeSignal($\hat{D}_f$)
    Matlab Simulink stepping through wavelet recomposition (Daubechies 6)
    return recomposed signal
```

Algorithm 3: Online Passive Delay Correlation with Wavelet denoising (opDCW) based shared bottleneck detection. Experiments use a correlation window of 100 samples (10 samples per T over a $10T$ interval) based on the findings in [11].

mechanism from [11] to make it also applicable for passive online usage.

Correlating a time varying metric is a powerful tool for grouping flows that share a common bottleneck. This method relies on time synchronized samples of the metric being used in order to examine shared correlations. Packet delay is a good frequent signal for doing this, but signal noise makes it difficult to use "as is". The use of summary statistics in section III was one way of dealing with the noise that does not require synchronized samples. Kim *et al.* [11] propose an alternative method that uses a wavelet based filter to denoise the OWD signal, utilizing Matlab's wavelet toolbox[8] to analyze the entire OWD probe signal time series and select the optimal filter thresholds offline. After filtering the OWDs, it calculates pairwise correlation coefficients to determine if flows should be grouped. Kim *et al.* [11] call their method Delay Correlation with Wavelet denoising (DCW). For direct comparison, we adapt DCW for passive use with a simple sample-hold prefilter to provide a valid delay value for each synchronized sample; calling it passive DCW (pDCW) in our experiments.

An online method does not have the entire time series to work with, and thus does not have prior knowledge of future delay dynamics to use to determine good filter coefficients. However, wavelet transforms are very efficient and can be calculated in an online manner. We adapt the method in [11] to dynamically choose reasonable filter thresholds when we estimate that the flow is experiencing congestion along its path using the skewness estimate test described in [9]. We call this method online passive DCW (opDCW). Algorithm 3 details our online wavelet based filtering and correlation based flow grouping mechanism.

[8]Matlab wavelet toolbox: https://se.mathworks.com/products/wavelet

V. Selecting an SBFG method: tradeoffs and practical considerations

As our experimental results will show (section VI), there is no clear-cut answer to the question, "which algorithm is better?". There are several factors that need to be considered when picking a particular algorithm, and these factors can be application- and scenario-dependent. In what follows we discuss the main tradeoffs and considerations that may guide the choice of an algorithm.

Which mechanism is more suitable depends on the application it is to be used in and the availability of per packet delay samples. The summary statistic methods only require the summary statistics to be fed back to the sender, so this could be an advantage where one wishes to limit the amount of feedback. OpDCW (note pDCW is an offline method) has a much lower lag in detecting bottlenecks. Where this is of high importance to the application opDCW, or some other filtered correlation method, will be the better choice.

Where to best locate the SBD determination algorithm, and where to collect the necessary input data, depends not only on the algorithm but also on the use case. For example, SBD mechanisms can operate on the sender and/or the receiver side of all hosts in a multi-party communication (e.g., a video conference). Pure receiver-side operation has the advantage that a receiver can passively obtain OWD samples from time stamps in incoming packets from various senders. Then, a grouping decision can be made, and the result can be relayed to the sender, thereby reducing the amount of feedback to a function of the frequency at which the sender wants to react to this information. However, a receiver may not see all the relevant bottlenecks due to traffic between a sender and a different receiver; this is explained in more detail in Section 3 of [9]. Sender-side operation of a mechanism such as opDCW requires the receiver to feed back per-packet delay samples. Such samples can be combined into a single feedback packet using the RTCP Transport Layer Feedback Message as described in [23]. As explained in [24], RTCP feedback messages can be generated often enough in practical multimedia communication conditions.

All of the four methods tested in section VI work on the assumption that the perturbation caused by the queueing delays at a bottleneck have measurable similarities for all flows sharing that bottleneck. There are however possible scenarios where this is not the case, e.g:

(i) The bottleneck delay signal is too small with respect to other delay perturbations packets experience along their path, such as other queues and delays due to lower layer loss correction on wireless links.

(ii) The flow sending pattern dominates the delay characteristics (self inflicted delays). This may be similar for flows from the same source, even though they do not traverse any shared bottlenecks.

With item i, this may mean measurement based SBFG is not possible in that particular scenario without some additional correlatable signal, e.g. some sort of packet marking. RFC 8382 suggests adding such a statistic when it is globally available. With item ii, a more sophisticated sending pattern aware algorithm is required in order to remove the sending pattern noise from delay samples. Item ii is a topic for further research requiring the delay contribution from the flow's sending pattern to be removed as unwanted noise.

Regarding complexity, rmcatSBD requires sorting for each statistical measure, so its time complexity is $O(MN \log N)$, where M is the number of measures and N is the number of flows. DcSBD calculates a fully meshed graph, $O(MN^2)$, and sorts the distances within each group, $O(GK \log K)$, for G groups of size K for one iteration; its complexity is therefore $O(MN^2 + IGK \log K)$ where I is the number of iterations. OpDCW and pDCW require pairwise correlation which depends on the correlation sample window size W and the number of combinations, so their complexity is $O(WN^2)$. DcSBD is generally more complex than rmcatSBD. Comparing rmcatSBD and opDCW, the least complex algorithm is scenario dependent, however for the simulation scenario in section VI-B it is rmcatSBD.

VI. Experimental Evaluations

The objective of the experimental evaluation is to understand the issues and compare the operation of the following four test mechanisms under conditions where their operation can be understood, but where SBFG is difficult. To this end, we use a simple testbed, more complex repeatable `NS2` [25] simulations, and cross-Internet tests on the NorNet testbed [26]. We evaluate the following algorithms:

1) The summary statistic based method described in [9] (rmcatSBD).
2) The method using the summary statistics in [9] with the dynamic clustering algorithm described in algorithm 2 (dcSBD).
3) The offline wavelet filtered cross-correlation method in [11], but with sample-hold OWD sampling (pDCW).
4) An online passive version of the method from [11] described in algorithm 3 (opDCW).

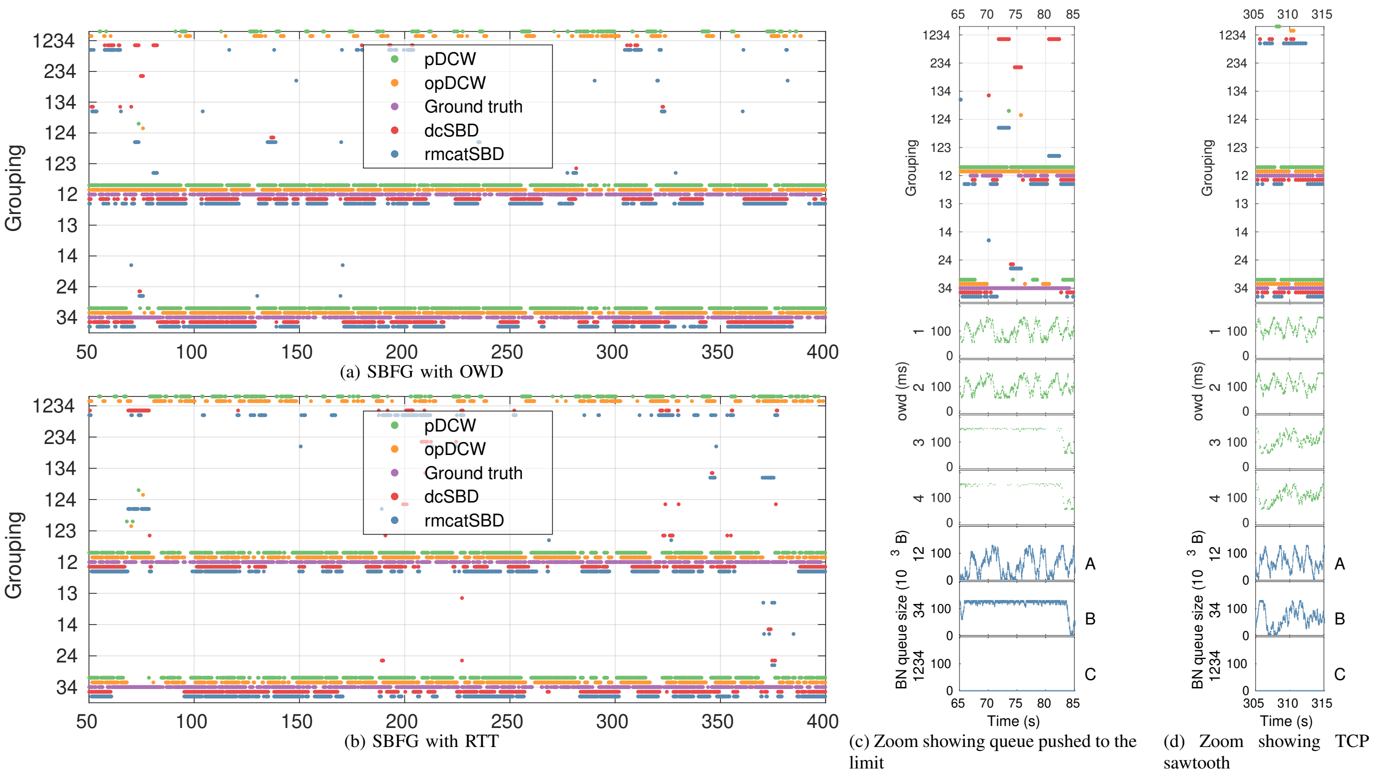
Our choice of mechanisms for experimental comparison was further guided by the following rationale. First, we want to compare our algorithms with an alternative [11] based on the same metric (i.e., OWD) but using a fairly different method. Also, we do not want to focus on protocol-dependent proposals. Finally, we are interested in methods designed to keep track of shifting (i.e., dynamic) bottlenecks and thus dynamic flow groups.

For the testbed experiments in section VI-A we can estimate the ground truth reasonably well and use it as a reference in the results. For the simulation experiments in section VI-B, the "ground truth" is known, so we compare the four mechanisms with this ground truth to determine their accuracy.

A. Simple Testbed Experiments

1) Setup: We run small-scale experiments in our extended Teacup testbed[9] to investigate SBFG for the four methods (see Fig. 1). The experiments use 10 Mbps access links (A and B) and a 25 Mbps link to the sink (C). The setup is

[9]http://caia.swin.edu.au/tools/teacup/



(a) SBFG with OWD

(b) SBFG with RTT

(c) Zoom showing queue pushed to the limit

(d) Zoom showing TCP sawtooth

Fig. 2. The testbed scenario illustrating the 4 SBFG mechanisms with respect to an estimate of "ground truth". Figure 2a and Figure 2b show the grouping decisions of the different algorithms with the estimate of ground truth in the center. On the zoomed in graphs, the additional lower graphs show, from the bottom, the sampled queueing delays at each bottleneck (refer to Fig. 1) and OWD measurements on which the grouping decisions are based.

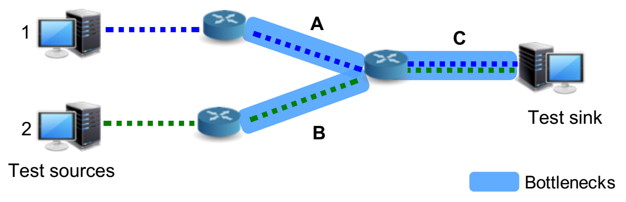


Fig. 1. Teacup testbed experimental setup

symmetric with a 50 ms base OWD (base RTT 100 ms), source to sink. UDP background traffic using D-ITG [27] is generated to give a long term average bottleneck filling level of about 80%. The background traffic consists of 4 exponentially distributed UDP packet flows and one Pareto distributed heavy tail on, exponentially distributed off UDP source traversing each access link to the sink. The Pareto distributed flows ensure the short term traffic dynamics on each link can be different in an otherwise completely identical setup. The test traffic in these tests are 2 greedy TCP New Reno flows from each of the sources generated using iperf[10]. We use BDP length queues at A, B, and C, with flows sending 1000 B packets. We also vary the queue length to half a BDP and two BDPs. We add additional background traffic to the TCP ACK return path to create an additional fluctuating return path bottleneck at C (loaded to 90% on average).

2) *Testbed results:* The results of the experiment are shown in Fig. 2a and 2b using both OWD and RTT as base metrics respectively. The graph depicts the grouping decisions (one every 350 ms) with the ground truth estimate represented by the purple dots in the middle of the decisions of the four tested mechanisms. The summary statistic methods are below the ground truth: dcSBD below it in red, rmcatSBD below dcSBD in blue. The wavelet filtered correlation methods are above the ground truth: opDCW above ground truth in orange, and pDCW above opDCW in green.

There are two distinct bottlenecks on links A and B which carry flows {1,2} and {3,4}, however they are not static, but fluctuating to some extent. There are no other bottlenecks on the paths from source to sink. Referring to Fig. 2a, generally all four algorithms mostly group correctly, though no algorithm always manages to follow the fluctuations. The summary statistic based methods do not group well unless the bottleneck remains quite stable, which makes sense since they are averaging statistics over a number of seconds.

Figures 2c and 2d zoom in these results to have a closer look at some salient aspects. The zoomed figures are composites with the probed queue sizes for each bottleneck in blue at the bottom (plot sub-sampled) from which an estimate of ground truth is determined (see section VI-B2 for a detailed explanation of accuracy and ground truth determination). The right hand side shows the bottleneck labels (A,B,C) from Fig. 1, with the left hand side showing the groups of flows that traverse that particular bottleneck. The middle graph in the composite shows the OWD measurements received by the different sources. Referring to Fig. 2c, during $t = [60, 70]$ bottleneck B is completely saturated. No algorithm is able to correctly group flows {3,4} during this period. There is no variation in OWD, and there are also intervals where no

[10]https://iperf.fr

packets arrive due to TCP time-outs due to high packet loss. Figure 2d shows a distinct TCP saw-tooth at bottleneck A, reflected in the OWD plot. This causes difficulties for all of the algorithms since they work on the assumption that OWD measurements are dominated by the characteristics of the bottleneck, and in this case they are not—they are dominated by the flows' sending characteristics. How to remove this sending pattern noise is a direction future SFBG research address.

RTT can be used as an alternative to OWD measurements, however it does include noise on the reverse path. Figure 2b shows how all algorithms would perform for the same experiment if RTT was used instead of OWD. In this experiment there is some background traffic on the return path causing reverse path queueing at C that sometimes dominates the RTT measurements. When this happens flows are often grouped together as {1,2,3,4} due to the reverse path delay. How significant this is will depend on the scenario.

We next evaluate the algorithms further in a completely controlled simulation environment without a TCP saw-tooth and where the ground truth can be exactly determined.

B. Simulation Experiments

1) Setup: We use the simulation setup from [10], see Fig. 3. The potential bottlenecks are labeled A to G. Each of the first six (from the left) bottlenecks carry traffic from a different set of two test sources, with the last bottleneck carrying traffic from all four test sources[11]. Each flow shares links in various combinations with other flows, causing correlated noise on the measurements and making SBFG more difficult than would be the case in a typical real network.

Bottlenecks are instantiated by reducing the capacity of the nominated link, depending on the traffic sources traversing that link. This gives better control over when bottlenecks start and finish. The majority of traffic (> 90% of bytes) traversing the bottlenecks is background traffic. It consists of thousands of TCP sessions of varying lengths and various levels of concurrency generated by `Tmix` [28] with TCP New Reno using the TCP evaluation suite traces which were generated from real traffic traces[12]. The traffic traces have had session start times scaled to achieve an offered load of around 25 Mbps on average per `Tmix` source (see [29]). Session start times are also shuffled to remove long term non-stationarity. Each new experiment run uses a different subset of the shuffled traces, to generate the delay, loss and queue traces. In this way each SBFG mechanism is tested on identical data and can therefore be directly compared.

Key simulation parameters are given in table III. Link delays are fixed for the duration of each run, with each new run using delays generated randomly according to a normal distribution of mean μ and standard deviation σ to provide some perturbation to topology. In a similar manner, bottlenecks have uniformly random buffer sizes from 133–266

[11]Test sources send "generic" UDP packets. Evaluation of the effect of various types of traffic (e.g. voice, video, TCP, etc) on the performance of the passive algorithms is beyond the scope of this work.

[12]Traces: http://heim.ifi.uio.no/michawe/research/tools/combined_traces.tbz

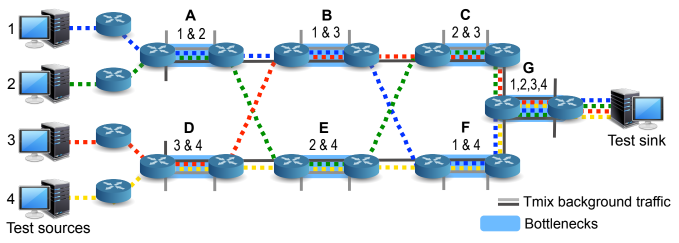


Fig. 3. Simulation setup (from [10]). Bottlenecked flows at each link are shown below each link's name.

TABLE III
SIMULATION PARAMETERS (BN–BOTTLENECK, BG–LINKS CONNECTING TMIX TRAFFIC GENERATORS, AND OTHER–ALL OTHER LINKS, SEE FIG. 3).

Link rates (Mbps)				Test src rates		link delay	
A,B,C,D,E,F		BG*	Other	1 & 2	3 & 4	(ms)	
not BN	BN			400 pps	200 pps	μ	σ
110†	30–50‡	100	110	(1.6 Mbps)	(0.8 Mbps)	6	2

† 220 Mbps for G ‡ 75 Mbps for G * Access link for background traffic

packets (200 packets for the last link). This can be less than a Bandwidth-Delay Product (BDP) when it is a bottleneck, though smaller queues make delay based shared bottleneck detection more difficult since the dynamic range of the delay signal is smaller for the same noise. We use `NS2` [25] for the simulations.

2) Accuracy tests: Determining the accuracy of a mechanism in a dynamic network environment is difficult, and it is only feasible if the ground truth can be quantified in some way. This is done for our simulations where we have a complete knowledge of the network dynamics, but unfortunately cannot be done for the NorNet experiments (section VI-C) as there are too many unknown factors. Questions to be considered are:

1) If a shared bottleneck is formed, but there is a delay in detecting it (summary statistics on their own introduce a delay), should this be counted as a false negative until such time as the grouping is detected, or should this delay not count as a false negative, but instead be quantified as a measured delay to the correct grouping?
2) Conversely, if a shared bottleneck abates, and there is a delay for a mechanism to recognize this, should this be counted as a false positive, or should this instead be just quantified as a delay to the correct grouping?
3) How stable should a bottleneck be before it counts as a bottleneck?
4) If flows pass through serial combinations of differently shared bottlenecks (a scenario which is beyond the scope of the investigated algorithms), should these be treated as false negatives or ignored?
5) How should partially correct groupings be handled?

In considering these questions we believe the following approach fairly illustrates the accuracy of the four investigated mechanisms for the experiments we conduct.

We measure the decision delay rather than counting decision errors due to the delay. This gives a clearer picture of a mechanism's utility in a dynamic environment and makes the measure independent of the average length of time bottlenecks exist in the experiments. It also allows the combination of accuracy and delay to be used to compare the relative efficacy of the mechanisms in a variety of scenarios where the relative importance of delay and accuracy are different. We use "queue empty" occurrences at each router as the base measure for independently determining whether a bottleneck exists or not. Specifically, a link is considered to be a bottleneck if the queue into that link does not empty over the period T, where $T = 350$ ms—the base measurement period for the rmcatSBD and dcSBD mechanisms.

Using a delay adjusted moving average of 15 T intervals (5.25 s), we consider a bottleneck to be stable when the associated queue does not empty in at least 50% of those intervals. Flows that pass through multiple bottlenecks, competing with different flows on the different bottlenecks are possible in this simulation setup. Such scenarios are ignored in the accuracy determination[13].

Partially correct groupings are considered to be wrong. This is harsh for groups of large numbers of flows, however, it is good enough for these experiments since most experiments have valid groups of only two or four flows.

3) Simulation Results: We use simulation experiments to compare the four SBFG mechanisms with "ground truth" in order to determine their accuracy. The summary statistic based methods, rmcatSBD and dcSBD, use the parameters recommended in [9]. The wavelet filtered methods sample the OWD every 35 ms and use a correlation coefficient threshold of 0.5 to determine whether the OWDs are correlated or not.

Figure 4 illustrates the operation of the different mechanisms for one particular simulation run. The composite figure has the measured queue sizes for each bottleneck in blue at the bottom (plot sub-sampled) from which the ground truth is determined. The right hand side shows the bottleneck labels from Fig. 3, with the left hand side showing the groups of flows that traverse that particular bottleneck. The middle graph in the composite shows the OWD measurements of the different flows received by the sink (plot sub-sampled). The top graph depicts the grouping decisions (one every 350 ms) with the ground truth represented by the purple dots in the middle of the decisions of the four tested mechanisms. The summary statistic methods are below the ground truth: dcSBD below it in red, rmcatSBD below dcSBD in blue. The wavelet filtered correlation methods are above the ground truth: opDCW above ground truth in orange, and pDCW above opDCW in green.

All four techniques seem to perform quite well in this scenario. The wavelet filtered correlation methods (pDCW and opDCW), often have a shorter delay[14] before flows sharing a dynamic bottleneck are grouped properly, but do not follow the transience of a bottleneck, as indicated by the ground truth, quite as well as the summary statistic based methods (rmcatSBD and dcSBD). For example, refer to the bottleneck at $t \approx [550, 630]$ s in Fig. 4 for flows 1 and 4. Visually, pDCW and opDCW also tend to have significantly more spurious groupings (false positives) than rmcatSBD and dcSBD. pDCW and opDCW seem to perform similarly, though opDCW seems to sometimes have a longer delay before a correct grouping.

To evaluate the accuracy quantitatively we repeat this experiment 50 times with different, but comparable background and source traffic. The grouping results are compared with the ground truth determined from the bottleneck queues (see Fig. 5). Figure 5a summarizes the accuracy of the four algorithms with respect to ground truth, depicting:

1) The proportion of decisions that correctly group flows sharing a common bottleneck:
$$100 \times \frac{\text{positives}}{\text{positives} + \text{false negatives}}.$$
2) The proportion of decisions that correctly do not group flows that should not be grouped:
$$100 \times \frac{\text{negatives}}{\text{negatives} + \text{false positives}}.$$
3) The F1 score, i.e., the harmonic mean of precision and recall:
$$100 \times \frac{2 \times \text{positives}}{2 \times \text{positives} + \text{false positives} + \text{false negatives}}.$$

As with all box-and-whisker plots, boxes span the middle 50% of results collected from 50 simulation runs, and whiskers extend up to 1.5 times the box span. Both wavelet-based methods perform well correctly grouping flows that share a bottleneck but have more difficulty in refraining from grouping flows that do not share a bottleneck. The summary statistic based methods are less accurate in correctly grouping flows than the DCW based methods, with the dynamic clustering version giving similar performace to the RMCAT version; however they very accurately refrain from grouping flows that do not share a bottleneck with less false positives—performing equally well due to their common skew_est based method of determining whether a flow is transiting a bottleneck or not. The F1 score has the summary statistic methods perform significantly better than the wavelet based methods, however the F1 score does not include negatives in the calculation which may be a more important factor in some applications. We will use the F1 score for comparisons of the algorithms with respect to delay offsets of flow from the different sources, numbers of flows sharing a common bottleneck and number of parallel bottlenecks.

Figure 5b summarises the detection delays in terms of: (a) Start—time delay from when a bottleneck occurs until when flows are correctly grouped, and (b) Stop—time delay from when a bottleneck abates until when flows are correctly no longer grouped. The wavelet based correlation methods are significantly faster at grouping flows when a bottleneck starts than the summary statistic based methods–calculating summary statistics introduces a lag. pDCW is faster, having the benefit of calculating ideal filters based on the entire trace. The wavelet based correlation methods have more difficult in stopping to group flows when the bottleneck abates. The

[13]Flows that pass through multiple bottlenecks have a combined summary statistic delay "signature" from each bottleneck. This delay "signature" will be different from that of a flow passing through a single bottleneck, and also different from flows passing through different combinations of bottlenecks. For cross-correlation methods in these circumstances, a flow will show low correlation coefficients with a number of other flows, making accurate grouping almost impossible.

[14]Remember that pDCW is an offline method, so the delay is very long in practice. It is here for comparison, since it shows what one could obtain if one was able to choose the ideal filter coefficients at the start.

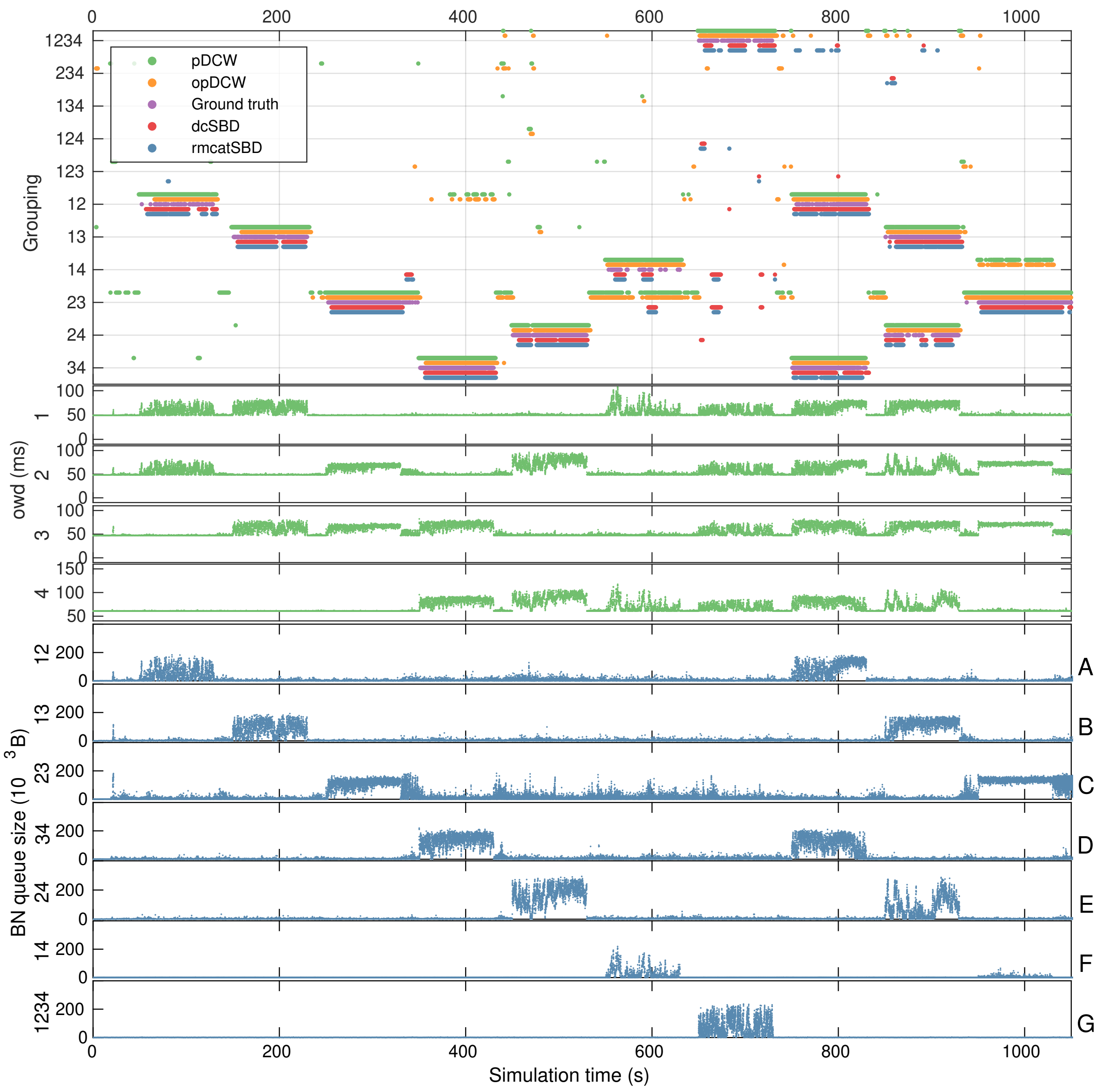


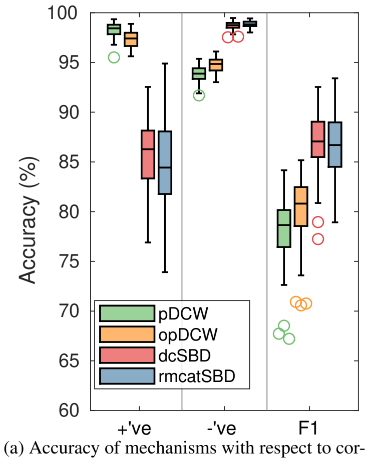
Fig. 4. Single scenario illustrating the 4 SBFG mechanisms with "ground truth". The lower graph shows the (subsampled) measured queueing delays at each bottleneck. The middle graph shows the OWD measurements on which the grouping decisions are based. The upper graphs shows the grouping decisions and accuracy, with dots appearing in the same vertical order as in the legend. Note that the 3-flow groups of 123, 124, etc. are groupings that should not occur given the simulation setup.

summary statistic methods have similar delays for both the start and stop of bottlenecks.

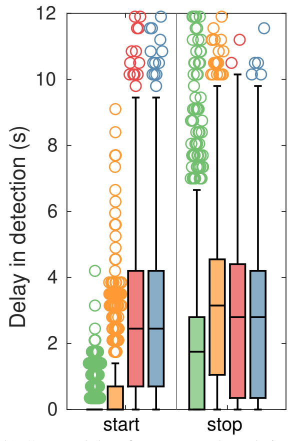
Differences in propagation delays between the sources that occur after a bottleneck can make it more difficult to group flows sharing a common bottleneck. Figure 6 shows the results from experiments when we add delays, $L_D = \{0, 10, 20, 50, 100, 200, 500, 1000\}$ ms, after the bottlenecks so that $\{0, \frac{L_D}{3}, \frac{2L_D}{3}, L_D\}$ ms, is added to flows from sources 1 to 4 respectively. All of the tested algorithms cope well with lags $\{0, 10, 20, 50, 100\}$ ms. At higher lags $\{200, 500, 1000\}$ ms the wavelet filtered correlation methods begin to struggle, since correlation becomes more difficult. The effect is minor for the summary statistic methods, because the summary interval is significantly longer than the lags.

Next we evaluate the accuracy of the different algorithms when grouping higher numbers of flows that share a common bottleneck. Using the same simulation set up depicted in Fig. 3, we increase the number of test flows traversing the network while balancing the background traffic to keep the offered load similar. Figure 7 shows that the accuracy is consistent for pDCW, opDCW and rmcatSBD. The accuracy of dcSBD begins to drop as the number of flows sharing a common bottleneck increases. With higher numbers of flows, the dcSBD algorithm sometimes finds subgroups within the bigger group of flows, leading to a reduction in the accuracy metric used here. There is also an increase in detection delay for dcSBD due to its 1 step back, 2 steps forward iteration (see section III-C). This can be fixed by simply increasing the number of forward steps in each iteration.

Finally we look at how increasing numbers of parallel bottlenecks that the algorithms need to group flows for influence accuracy. To do this we use multiple parallel instances of the



(a) Accuracy of mechanisms with respect to correctly grouping (+'ve), correctly not grouping (-'ve) and the F1 score.



(b) Start—delay from ground truth indication of a bottleneck starting to correctly grouping flows. Stop—delay from ground truth indication of a bottleneck ceasing to correctly detecting that flows cease to share a bottleneck.

Fig. 5. Accuracy and delay to detection of the different shared bottleneck detection mechanisms. Groups of four show pDCW, opDCW, dcSBD and rmcatSBD from left to right. Each box represents results from 50 runs.

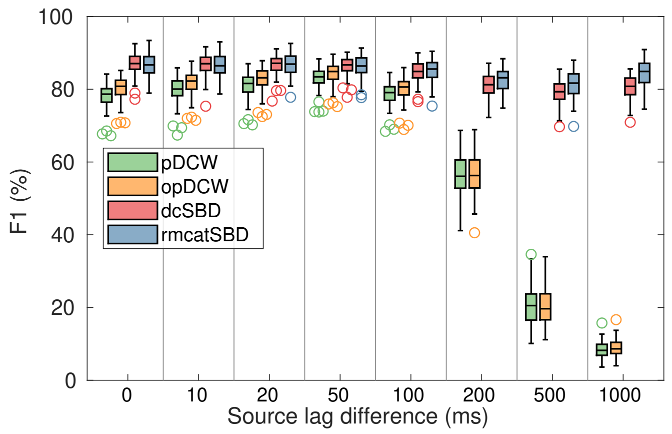


Fig. 6. Accuracy with respect to differences in source propagation delays (L_d) occuring after the bottleneck. Groups of four show pDCW, opDCW, dcSBD and rmcatSBD from left to right. Each box represents results from 50 runs.

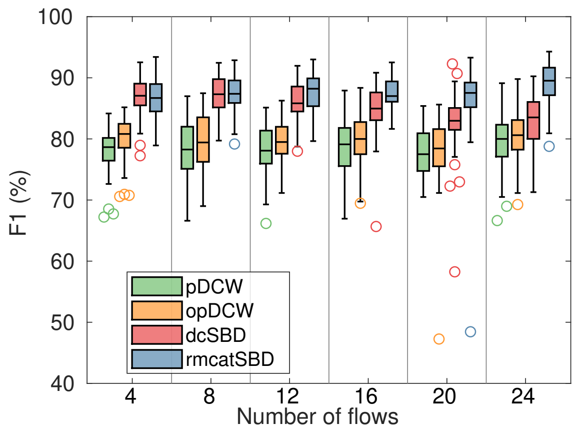


Fig. 7. Accuracy with respect to the number of flows to be grouped. Groups of four show pDCW, opDCW, dcSBD and rmcatSBD from left to right. Each box represents results from 50 runs.

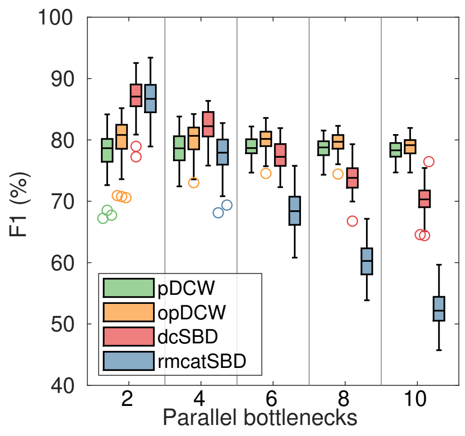


Fig. 8. Accuracy with respect to the number of parallel bottlenecks. Groups of four show pDCW, opDCW, dcSBD and rmcatSBD from left to right. Each box represents results from 50 runs.

same simulation setup (see Fig. 3), each with different traffic at a comparable offered load. Figure 8 shows that the wavelet based correlation methods perform consistently. Both dcSBD and rmcatSBD show a drop in accuracy as the number of parallel bottlenecks increase, with rmcatSBD's drop especially significant. This is due to the simple threshold discrimination used by rmcatSBD that struggles to separate large numbers of parallel bottlenecks.

Each of the algorithms has its limitations and strengths depending on the scenario. The wavelet correlation methods struggle with differing relative source lags. DcSBD's clustering method can sometimes find sub-groups in larger bottleneck groupings. RmcatSBD can struggle to discriminate between groups when there are large numbers of bottlenecks. Overall the online opDCW modification of DCW has a similar accuracy, though longer delay to detection, when compared to the offline pDCW, indicating the dynamic choice of thresholds for wavelet filtering is good enough for this purpose.

C. NorNet Experiments

1) Setup: To evaluate the effectiveness and robustness of SBFG in real operational networks, we show results from the measurement traces from the NorNet testbed experiments used in [10]. In summary, the NorNet testbed consists of wired NorNet Core (NNC) [26] nodes and a wireless (cellular) NorNet Edge (NNE) [30] node interconnected via the Internet.

Figure 9 illustrates the experimental setup. On the right-hand side, we show the NNC sites and their respective geographic locations. The green solid lines indicate the NNC sites that generate the traffic we use to represent application traffic, which we want to measure, while the red solid lines indicate the NNC sites that generate the synthetic background traffic used to congest the links and create fluctuating bottlenecks. Note that all traffic generated at each NNC site (except for (F) and (G)) will first go through the `Tbox` (Tunnel Box) in Oslo, before going out again on the Internet towards the NNE node (NNE-1), which is connected to the Internet via two different cellular network providers.

This topology creates a difficult shared bottleneck scenario: Operators (I) and (II) have very different wireless links each

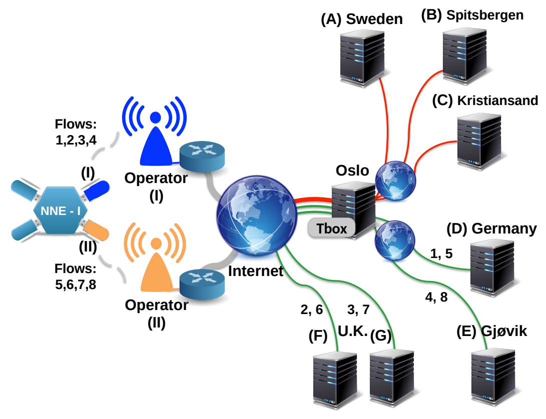


Fig. 9. NorNet setup. Our sink is the multi-homed NNE-1 node located in Oslo, Norway. The main bottlenecks are on the wireless links, with a secondary bottleneck possible at the Oslo Tunnel Box (TBox), see [10].

with a different data rate. The queue on (I) appears to use some sort of Active Queue Management (AQM), while the queue on (II) rarely drops packets. We use Operator (I) as a destination for flows {1,2,3,4} and Operator (II) as a destination for flows {5,6,7,8}. In our experiments we attempt to create dynamic bottlenecks on the wireless links of both Operators (I) and (II), links that have a varying channel capacity. In addition to these, there are a number of other potential bottlenecks, such as the Tbox itself. We believe that this setup creates a realistic test environment, however it is not possible to determine the *ground truth* in a real network as was done in the simulation experiments in section VI-B, since it is not possible to have queueing measurements at every potential bottleneck in each flow's path across the Internet.

2) NorNet Configuration: Although the "ground truth" can not be reliably determined for our experiments, we use `traceroute` and `STAB` [31] to identify the tight common links along the paths from the NNC nodes to NNE-1. We find these to be the wireless links, and on occasion the link through the Oslo Tbox. We dimension the background traffic load in such a way that the wireless links will usually have the tightest bottlenecks during the experiment. Even then, the uncontrolled Internet traffic may generate other unknown bottleneck during the course of the experiments.

Our synthetic background and application traffic flows are generated from all NNC sites towards NNE-1 via Operators (I) and (II), with UDP, using the `D-ITG` tool [27]. The objective is to emulate a realistic traffic pattern that causes a dynamically fluctuating bottleneck. Node (A) generates one flow with exponentially distributed inter-packet intervals with mean rate of 92 packets per second (pps) via (I), and another 460 pps flow via (II). (B) and (C) generate more complex and bursty long range dependent traffic (Hurst = 0.8). (B) and (C) each generate 8 flows with Pareto distributed on times, and exponentially distributed off times with the following means: 4 with 2s/2s and 4 with 5s/5s on/off intervals at 72 pps via (I), and 8 flows with 1s/1s, 2s/2s,…, to 8s/8s on/off intervals at 90 pps via (II). Background flows have variable packet sizes with an average 1000 Byte per packet.

We generate test traffic of different rates. Machines (D), (E), (F) and (G) generate the traffic we measure to test our

TABLE IV
GROUPING STATISTICS FROM THE NORNET EXPERIMENT. PERCENTAGES

Grouping	rmcatSBD			dcSBD			pDCW			opDCW		
	Dcn[a]	RL[b]		Dcn[a]	RL[b]		Dcn[a]	RL[b]		Dcn[a]	RL[b]	
	(%)	μ	σ	(%)	μ	σ	(%)	μ	σ	(%)	μ	σ
1234	22	4.3	5.4	18	6.1	5.0	147	5.1	5.6	146	11.6	10.4
14	18	2.9	3.3	30	7.8	8.1	14	1.6	1.1	16	2.6	1.9
23	13	3.1	3.4	27	11.3	14.7	25	2.0	1.6	27	3.4	3.1
Total [c]	53	-	-	75	-	-	187	-	-	188	-	-
Others [d]	21	-	-	29	-	-	5	-	-	4	-	-
5678	31	7.6	10.0	28	10.0	9.7	56	6.2	6.7	54	6.8	7.9
58	30	5.7	7.9	40	10.7	11.0	36	2.2	2.2	38	4.7	4.7
67	41	8.0	10.7	47	14.5	15.7	43	2.6	2.5	47	5.3	5.0
Total [c]	102	-	-	115	-	-	135	-	-	138	-	-
Others [d]	9	-	-	10	-	-	2	-	-	2	-	-
12345678	1.3	5.4	5.4	0.2	-	-	7.8	3.5	2.5	7.2	2.8	2.4
123458	2.7	5.0	5.4	0.2	-	-	5.6	2.7	2.8	6.1	2.6	2.1
145678	1.0	7.5	8.4	0.0	-	-	1.7	1.6	1.0	1.6	1.7	1.2
1458	1.9	4.6	7.6	1.0	3.7	2.4	2.0	1.5	1.3	2.7	3.0	3.5

[a] Decisions (Dcn). The percentages are the number of decision instances relative to the number of times congestion is indicated for at least 2 of flows "1234" (1475 instances) or flows "5678" (2644 instances) at the decision time. The third part of the table gives percentage based on 4119 total instances. Percentage values higher than 100% indicate that the algorithm was finding shared bottleneck groupings it should not.
[b] Run Length (RL) – Length of consecutive grouping decisions, mean (μ) and standard deviation(σ). Decisions are made every 350 ms in this experiment.
[c] Total: Total of main combinations within this group of four.
[d] Other: Other combinations within this group of four.

detection mechanism (flows labeled 1, 2,…, 8 in Fig. 9). The test traffic is a minor contributor to the bottlenecks in these experiments. Sites (D), (E), and (F) generate flows with exponentially distributed inter-packet times with an average of 100 pps with 50 Byte packets. (G) generates a constant rate of 100 pps with 50 Byte packets.

3) NorNet Results: Unfortunately it is not possible for us to get the "ground truth" from all the different queues that may form due to bottlenecks, including the two designed bottlenecks on the radio side of the cellular interface. To exacerbate the problem, the capacity of (I) and (II) changes with the radio conditions (see Fig. 9). What we hope to see is sporadic flow groupings of {1,2,3,4} and {5,6,7,8}, but not of {1,2,3,4,5,6,7,8} since that is very unlikely in this set up. However, since the T-Box (see Fig. 9) can also sometimes be a bottleneck, we also expect {1,4} and {5,8} to sometimes separate from their respective group of four.

Table IV summarises the results form the NorNet experiment. The table is split into three parts: (i) variations around the {1,2,3,4} flow groupings, (ii) variations around the {5,6,7,8} flow groupings, (iii) and other combinations that have some statistical significance in the experiment. The grouping decisions are expressed as a percentage with respect to estimates of there being a bottleneck (i.e. congestion)[15] when the decision is made. Allowing for the influence of

[15]We use the method for inferring bottlenecks in RFC 8382 [9], section 3.3.1 item 1; counting it as a bottleneck if at least 2 flows of the group of four are inferred to transit bottlenecks.

the T-Box, the ideal result would be percentages near 100% for each algorithm in the total rows of the table. Percentages significantly less than 100% indicate that a particular algorithm missed potential SBFGs, i.e. there is a prevalence of false negatives. This is especially so for rmcatSBD (53%), and to a lesser extent dcSBD (75%), in the {1,2,3,4} flow groupings. Conversely values significantly over 100% indicate that an algorithm has determined SBFGs that should not exist due to there being no bottleneck, i.e., there is a prevalence of false positives. This is especially so for pDCW (187%) and opDCW (188%) in the {1,2,3,4} flow groupings. All algorithms seem to perform better, in this respect, for the {5,6,7,8} flow groupings with percentages closer to 100%. This seems to be due to the higher queueing delays (around double), so stronger bottleneck signal, experienced by these flows.

The rmcatSBD and dcSBD are more likely to split the flows {1,2,3,4} into two groups than pDCW and opDCW. The perturbation of the T-Box seems to cause a bigger difference in summary statistics than in correlation coefficient for these flows. Conversely, pDCW (7.8%) and opDCW (7.2%) are more likely to group all the flows together in the {1,2,3,4,5,6,7,8} group, a very unlikely grouping in this topology, than rmcatSBD (1.3%) and dcSBD (0.2%).

RL in table IV refers to the number of consecutive SBFG decisions for a particular group of flows. We present the mean and standard deviation. The standard deviations, relative to the means, are generally large, which is to be expected given the heavy tailed distributions used for the background traffic, and Internet traffic in general. High counts of RLs close to 1, e.g. groups {1,4} and {2,3} with pDCW, indicate either very short bottlenecks or possible noise in the decisions. DcSBD seems less prone to this, and to generally have comparatively longer RLs.

In section VI-B3 it was noted that dcSBD's 1 step back and 2 steps forward clustering iterations can cause delays in grouping higher numbers of flows, which may account for its performance of not grouping {1,2,3,4,5,6,7,8} group and being less prone to short RLs. We test this with 1 step back and $N = \{3, 4, 8\}$ steps forward clustering iterations, finding no statistical significant improvement: 0.1% increase in {5,6,7,8} groupings, 0.4% increase in {1,2,3,4} groupings, no change in {1,2,3,4,5,6,7,8} groupings, and no change in RLs.

VII. Conclusions and Future Work

Four different algorithms for grouping flows that share bottlenecks have been evaluated via a simple testbed, simulations and real networks. Simulations have the important advantage of being able to determine the "ground truth" exactly, however, they will never truly model the nuances of a real network.

Both rmcatSBD and dcSBD use summary statistics to characterise the effect of the bottleneck, and similarity in these statistics to group flows sharing a common bottleneck. This paper has evaluated the method in RFC 8382 [9] (rmcatSBD) and updated its divide and conquer grouping technique with a novel dynamic clustering method. In addition, the wavelet filtered DCW method [11] was adapted for passive use (pDCW) and a novel method of online calculation of wavelet filter coefficients enables an online version (opDCW).

Generally, all four methods achieved a high degree of accuracy with respect to ground truth in the simulation tests, though their performance varied depending on the scenario, each with different strengths and limitations. When there were no differences in delays from sources, the wavelet based correlation methods (pDCW and opDCW) more often correctly grouped flows, however the summary statistics methods seemed to be significantly less susceptible to wrongly grouping flows that do not share a common bottleneck than the wavelet filter based methods (see Fig. 5a). An inherent problem with summary statistics is that they introduce delay; it takes enough samples over a certain time interval to calculate them. This is clearly indicated in the accuracy tests that showed on average about a 2-3 s delay from the ground-truth indication to the correct detection of the bottleneck (see Fig. 5b). How important delay is depends on the intended application of the techniques.

Differences in propagation delay after the bottleneck cause difficulties for the wavelet based correlation methods (pDCW and opDCW) with them both performing poorly from source delay differences spanning 200 ms (see Fig. 6). RmcatSBD struggles to discriminate between groups when there are high numbers of parallel bottlenecks ($\geq$ 6, see Fig. 8), while dcSBD sometimes finds subgroups within the larger flow grouping when there are higher numbers of flows that share a common bottleneck—slightly reducing its performance ($\geq$ 16, see Fig. 7). Over the whole range of tests conducted, dcSBD has the most consistent F1 score, though not the best in any.

The online opDCW performed very closely to pDCW, which used Matlab to optimally compute the filter coefficients from the entire traffic traces. This makes it a viable online version of DCW, though relative significance of the filter coefficients to the mechanism's overall efficacy warrants further investigation. Increasing the correlation coefficient threshold for these methods may help reduce the problem of grouping flows that should not be grouped, but may also reduce their ability to correctly group flows sharing a bottleneck. This is also a topic for further research.

References

[2] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, *TCP Extensions for Multipath Operation with Multiple Addresses*, RFC 6824 (Experimental), Internet Engineering Task Force, Jan. 2013.

[3] D. Hayes, S. Ferlin, and M. Welzl, *Shared bottleneck detection for coupled congestion control for RTP media*, Internet Draft draft-ietf-rmcat-sbd-01, work in progress, Internet Draft, Jul. 2015.

[4] S. Ferlin, O. Alay, T. Dreibholz, D. A. Hayes, and M. Welzl, "Revisiting congestion control for multipath TCP with shared bottleneck detection," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, Apr. 2016, pp. 1–9.

[5] B. Briscoe, "Flow rate fairness: Dismantling a religion," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 63–74, Mar. 2007.

[6] S. Islam, M. Welzl, S. Gjessing, and N. Khademi, "Coupled congestion control for RTP media," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, Aug. 2014.

[7] S. Islam, M. Welzl, and S. Gjessing, *Coupled Congestion Control for RTP Media*, RFC 8699, Jan. 2020.

[8] S. Islam, M. Welzl, K. A. Hiorth, D. Hayes, G. Armitage, and S. Gjessing, "ctrlTCP: reducing latency through coupled, heterogeneous Multi-Flow TCP congestion control," in *2018 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS): GI 2018: 21st IEEE Global Internet Symposium (INFOCOM18 WKSHPS GI'18)*, Apr. 2018.

[9] D. Hayes, S. Ferlin, M. Welzl, and K. Hiorth, “Shared bottleneck detection for coupled congestion control for RTP media,” RFC Editor, RFC 8382, Jun. 2018.
[10] D. Hayes, S. Ferlin, and M. Welzl, “Practical passive shared bottleneck detection using shape summary statistics,” in *Proceedings of IEEE LCN*, Sep. 2014, pp. 150–158.
[11] M. S. Kim, T. Kim, Y.-J. Shin, S. S. Lam, and E. J. Powers, “A wavelet-based approach to detect shared congestion,” *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 763–776, Aug. 2008.
[12] S. Savage, N. Cardwell, and T. Anderson, “The case for informed transport protocols,” in *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, 1999, pp. 58–63.
[13] H. Balakrishnan, M. Stemm, S. Seshan, and R. H. Katz, “Analyzing stability in wide-area network performance,” in *Proceedings of ACM SIGMETRICS*, 1997, pp. 2–12.
[14] L. Wang, J. N. Griffioen, K. L. Calvert, and S. Shi, “Passive inference of path correlation,” in *Proceedings of ACM NOSSDAV*, 2004, pp. 36–41.
[15] D. Rubenstein, J. Kurose, and D. Towsley, “Detecting shared congestion of flows via end-to-end measurement,” *IEEE/ACM Trans. Netw.*, vol. 10, no. 3, pp. 381–395, Jun. 2002.
[16] S. Hassayoun, J. Iyengar, and D. Ros, “Dynamic window coupling for multipath congestion control,” in *Proceedings of IEEE ICNP*, 2011, pp. 341–352.
[17] W. Wei, Y. Wang, K. Xue, D. S. L. Wei, J. Han, and P. Hong, “Shared bottleneck detection based on congestion interval variance measurement,” *IEEE Communications Letters*, vol. 22, no. 12, pp. 2467–2470, Dec. 2018.
[18] O. Younis and S. Fahmy, “FlowMate: Scalable on-line flow clustering,” *IEEE/ACM Trans. Netw.*, vol. 13, no. 2, pp. 288–301, Apr. 2005.
[19] D. Katabi and C. Blake, “Inferring congestion sharing and path characteristics for packet interarrival times,” Massachusetts Institute of Technology, Tech. Rep. TR-828, Dec. 2001.
[20] P. Varga, “Analyzing packet interarrival times distribution to detect network bottlenecks,” in *Proceedings of IFIP EUNICE*, C. D. Kloos, A. Marìn, and D. Larrabeiti, Eds., vol. 196, 2006, pp. 17–29.
[21] M. M. Yousaf, “Accurate shared bottleneck detection and file transfer delay prediction for grid scheduling,” PhD thesis, University of Innsbruck, Austria, Dec. 2008.
[22] C. Biemann, “Chinese whispers: An efficient graph clustering algorithm and its application to natural language processing problems,” in *Proceedings of the First Workshop on Graph Based Methods for Natural Language Processing*, ser. TextGraphs-1, 2006, pp. 73–80.
[23] Z. Sarker, C. Perkins, V. Singh, and M. A. Ramalho, “RTP Control Protocol (RTCP) Feedback for Congestion Control,” Internet Engineering Task Force, Internet-Draft draft-ietf-avtcore-cc-feedback-message-06, Mar. 2020, Work in Progress, 15 pp.
[24] C. Perkins, “RTP Control Protocol (RTCP) Feedback for Congestion Control in Interactive Multimedia Conferences,” Internet Engineering Task Force, Internet-Draft draft-ietf-rmcat-rtp-cc-feedback-05, Nov. 2019, Work in Progress, 12 pp.
[25] *Network simulator (ns-2.35)*, 2011.
[26] E. G. Gran, T. Dreibholz, and A. Kvalbein, “NorNet Core - A Multi-Homed Research Testbed,” *Computer Networks*, Jan. 2014.
[27] S. Avallone, S. Guadagno, D. Emma, A. Pescape, and G. Ventre, “D-ITG distributed internet traffic generator,” in *Proceedings of the International Conference on the Quantitative Evaluation of Systems (QEST)*, Sep. 2004, pp. 316–317.
[28] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, “Tmix: A tool for generating realistic TCP application workloads in ns-2,” *ACM SIGCOMM Computer Communications Review*, vol. 36, no. 3, pp. 65–76, Jul. 2006.
[29] D. Hayes, D. Ros, L. Andrew, and S. Floyd, “Common TCP evaluation suite,” IRTF, Internet Draft draft-irtf-iccrg-tcpeval-01, Jul. 2014.
[30] A. Kvalbein, D. Baltrūnas, K. Evensen, J. Xiang, A. Elmokashfi, and S. Ferlin-Oliveira, “The NorNet Edge Platform for Mobile Broadband Measurements,” *Computer Networks*, Jan. 2014.
[31] V. J. Ribeiro, R. H. Riedi, and R. G. Baraniuk, “Locating available bandwidth bottlenecks.,” *IEEE Internet Comput.*, vol. 8, no. 5, pp. 34–41, Oct. 26, 2004.

David A. Hayes received his Ph.D. in Telecommunications Engineering from the University of Melbourne, Australia. Currently he works at Simula Metropolitan Center for Digital Engineering in the Mobile Systems and Analytics department where his work focuses on performance analysis of protocols and network devices in 5G networks and beyond.

Michael Welzl is a full professor at the University of Oslo, Norway, since 2009. He received his Ph.D. and his habilitation from the University of Darmstadt / Germany in 2002 and 2007, respectively. Michael’s main research focus is the transport layer; he is active in the IETF and IRTF.

Simone Ferlin is a software researcher at Ericsson AB in radio networks. She received her Dipl.-Ing. degree in Information Technology with major in Telecommunications from Friedrich-Alexander Erlangen-Nuernberg University, Germany in 2010 and her PhD degree in computer science from the University of Oslo, Norway in 2017. Her interests lie in the intersection of cellular networks and the Internet, with her research focusing on computer networking, QoS and cross-layer design, transport protocols, congestion control, network performance, security, and measurements. Her dissertation focused on improving robustness in multipath transport for heterogeneous networks with MPTCP. She actively serves on technical boards of major conferences and journals in these areas.

David Ros received his Ph.D. in Computer Science from the Institut National de Sciences Appliquées, Rennes, France. He is a Chief Research Scientist at Simula Research Laboratory, Oslo, Norway. He has contributed to IETF and IRTF documents in the field of congestion control, and co-chaired the IRTF’s Internet Congestion Control Research Group. His active research interests include future evolution of the Internet’s transport layer and architectural issues in IP networks.

Safiqul Islam received his Ph.D. in Computer Science from the University of Oslo, Norway. Currently, he is a Postdoctoral Fellow at the Department of Informatics, University of Oslo. His research interests include performance analysis, evaluation, and optimization of transport layer protocols. He is active in the IETF and IRTF where he has contributed to several IETF/IRTF Working Groups.