- FINAL -

Analysis and Algorithmic Performance Report for the Generalized Network Assisted Transport (GNAT)

part of the DARPA Dispersed Computing Program

Contract No.: HR0011-17-C-0050

Prepared By:
Raytheon BBN Technologies Corp.
10 Moulton St.
Cambridge MA 02138-1119

Prepared For:
SPAWAR Systems Center , Pacific
53560 Hull St. Code 56120
San Diego, CA 92152-5000

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-17-C-0050. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

REVISION HISTORY

Version	Summary of Changes	Date
-01	Release of GNAT Analysis and Algorithmic Performance Report	1-March-2021
-02	Update markings for public release	24-March-2021

APPROVALS

Prepared By: *On file*

Steve Zabele
Principal Investigator — Date

Concurrence: *On file*

Scott Loos
Program Manager — Date

Concurrence: *On file*

Maurice Foley
QA/CM — Date

CONTENTS

1 Preface

GNAT is in general able to keep network paths full, and so the raw throughput over a given path is for the most part close to the rate specified for the associated path emulator. The difference between path rate settings and the resulting *goodput* reported in TRPR plots (our primary performance reporting tool) is due to overhead. Currently, the combined overhead (Ethernet headers, IP headers, UDP headers, tunneling overhead, etc.) is around 12.5% depending on the base size of application packets and the particular experiment. Ethernet, IP and transport headers are of course an unavoidable consequence of using IP networking. In addition, approximately 1% of the overhead is due to GNAT internode signaling; for example, the reporting of queue depth information between nodes. The remaining overhead is partially due to tunneling overhead (necessary for creating an overlay network), and partially due to carrying extra packet metadata to make it easier for us to monitor and understand GNAT behavior during development. We fully expect that the total overhead can be reduced to 10% or less. In the meantime, we calculate the expected goodput rates based on 15% overhead for the experiments below.

Only loss from admitted packets is measured and accounted for: we do not consider (UDP) packets that were dropped because they were never admitted onto the network for lack of capacity or low-priority.

2 Modeling and Simulation-based Performance Assessment

2.1 Dynamic Distributed Load Balancing of Multicast Traffic Using ns-3

This example shows an implementation of multicast-over-backpressure using a network coded forwarding model, and uses the Dynamic Distributed Load Balancing of Multicast Traffic use case to demonstrate the basic GNAT concept. For reference the link capacities on the topology evolve over time as shown below: in our ns-3 model the units on the link capacity values are Megabits per second (Mbps).

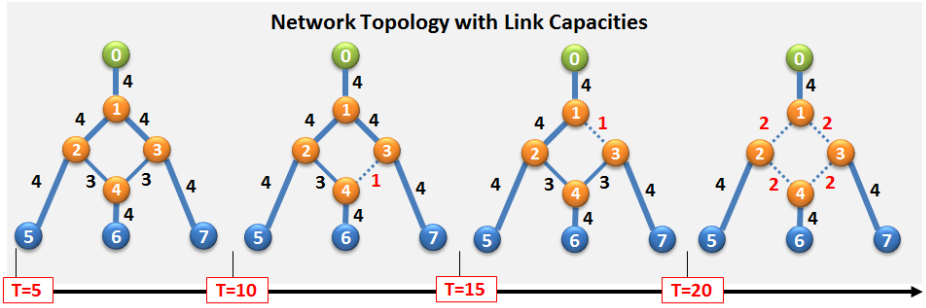


Figure 1. Example multicast problem, with time varying link capacities as shown.

A notional multipath forwarding solution for the multicast problem shown in Figure 1 is given below in Figure 2.

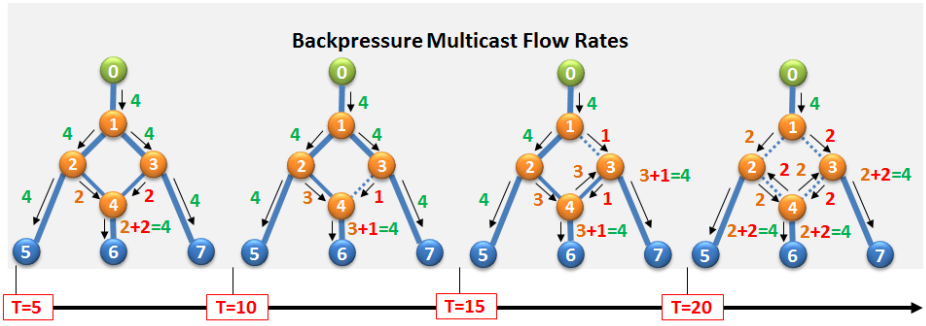


Figure 2. Notional distributed dynamic load balancing solution to the problem shown in Figure 1.

The assertion here is that a dynamic distributed load balancing solution should be able to sustain 4 Mbps throughout the sequence of link speed changes.

Results using an ns-3 implementation of multicast-over-backpressure along with network coding model are provided below. In Figure 3 we see that to a reasonable approximation the, links within the network are carrying loads that are very consistent with the notional multipath forwarding solution depicted in Figure 2 above. In Figure 4 we see that the load delivered to each of the receivers (nodes 5, 6 and 7) is a fairly steady 4 Mbps.

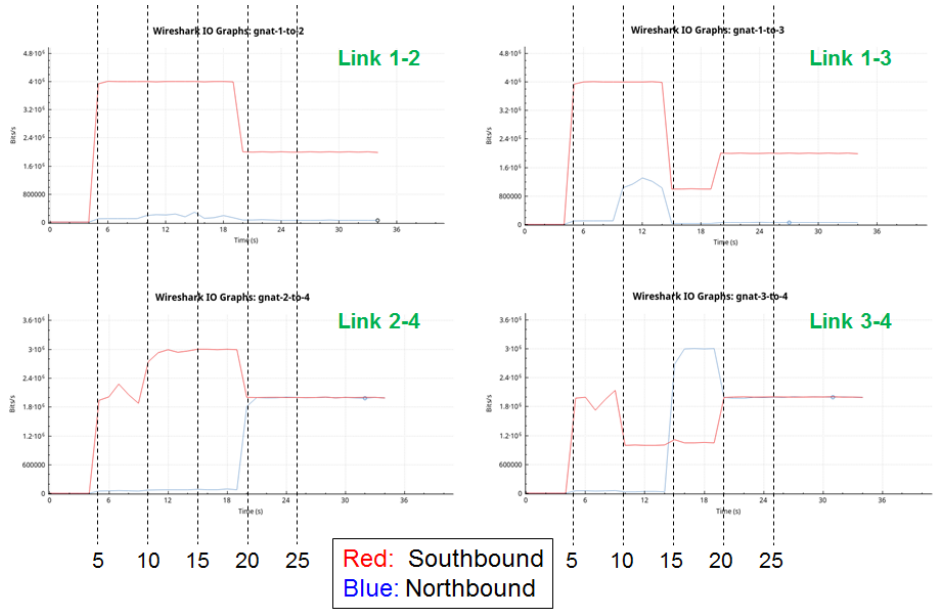


Figure 3. Link loading as a function of time, showing the dynamic load balancing response to various link changes.

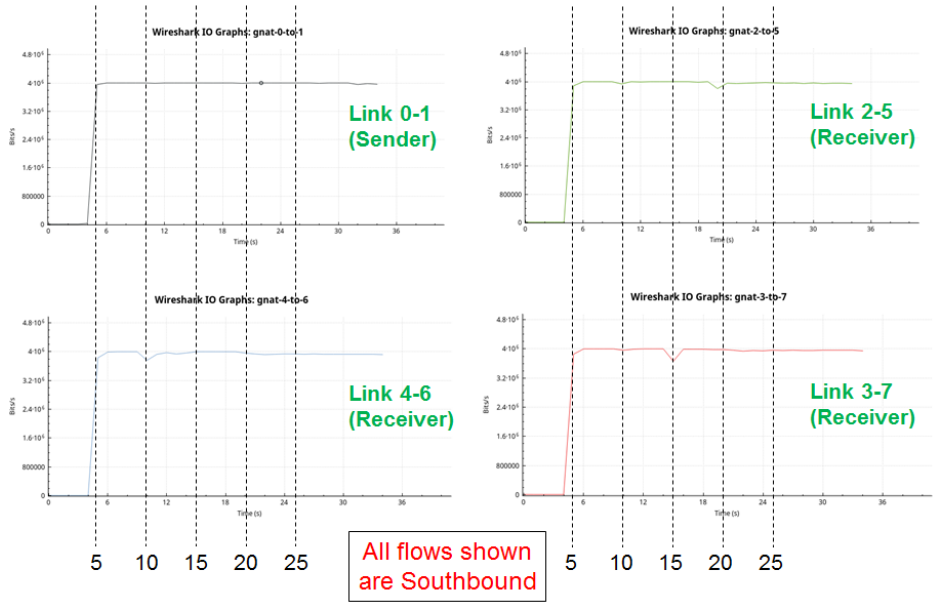


Figure 4. Load from the source node (node 0), and load delivered to each of the receiver nodes as a function of time, showing a very consistent 4 Mbps delivery rate.

2.2 MatLab-based Assessment of Latency Reduction Algorithms

2.2.1 Starting Point: IRON's Zombie Latency Reduction (ZLR)

Our original strategy for reducing forwarding latency in a backpressure network extended a technique we developed under the IRON project called Zombie Latency Reduction (ZLR). The core idea behind ZLR derives from the following principles.

First, the queue depth Q for any given flow is inversely proportional to aggregate egress rate r, with the constant of proportionality given by K: specifically, $r = K/Q$. It is well known that getting backpressure to work closer to an optimal forwarding solution requires larger values of K, which means for any given rate the queues can become quite large. Unfortunately, the queues need to be this large to get the admission controllers to admit packets at the "correct" load-balancing rates – which in turn implies large queuing delays. Hence getting near-optimal throughput using basic backpressure in general implies large delays.

Fortunately it is not necessary that queue depth values correspond to just the amount of physical packets enqueued. ZLR works by introducing virtual packets such that the sum of the physical and virtual packets yields the "correct" queue depth values, while requiring very few actual packets in the queues. Then, by forwarding only physical packets, the actual time in queue is kept small, and over all latency is improved while still retaining near optimal operating conditions for backpressure.

For GNAT, ZLR was extended to a vector formulation to support multicast traffic, basically maintaining independent virtual queue values for each destination in the multicast group.

2.2.2 Alternatives to ZLR: Stochastic Dual Gradient and Improved Stochastic Dual Gradient

As we were developing our vector-based version of the ZLR algorithm we became aware of an alternate technique that used a dual stochastic gradient approach for reducing queueing delays. (See T. Chen, Q. Ling, and G. B. Giannakis, “Learn-and-Adapt Stochastic Dual Gradients for Network Resource Allocation”, IEEE Trans. Control of Network Systems, Vol. 5, No. 4, December 2018).

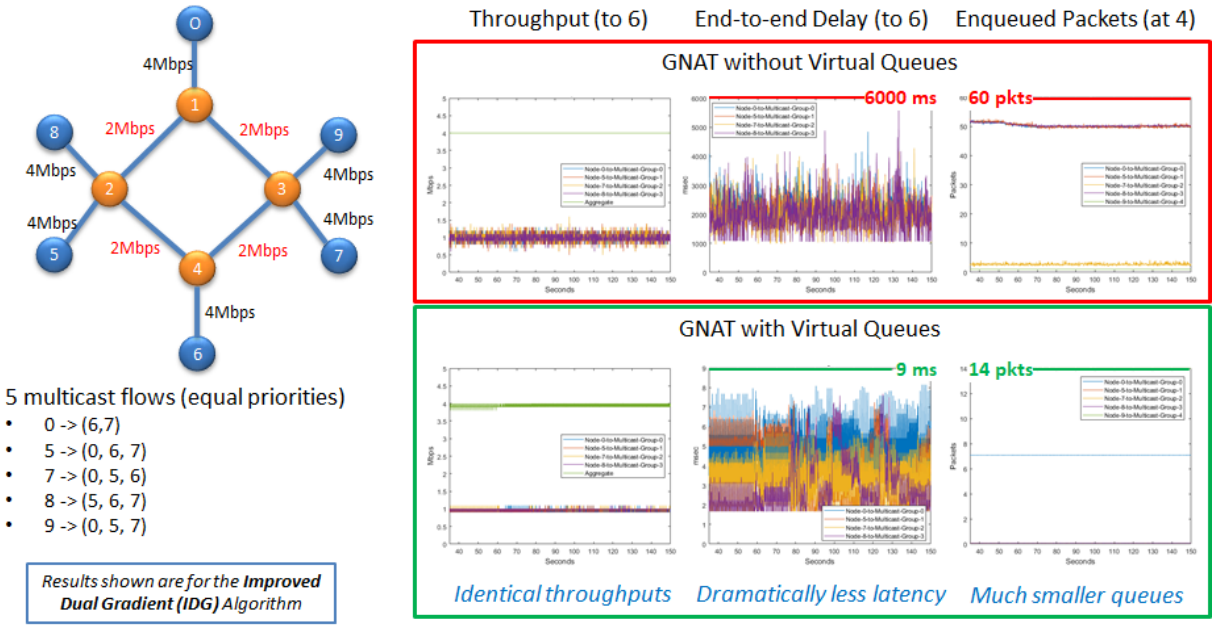
At a high level, the algorithm first tries to quickly learn the virtual queue depths λ_t using a stochastic gradient approach with decreasing step sizes, while concurrently using a separate stochastic gradient approach with fixed step sizes to learn the total queue depths $\gamma_t = (\lambda_t + \mu q_t)$. Here γ_t which is the weighted sum of the virtual queue depth λ_t and physical queue depth q_t. The second stochastic gradient estimator with fixed step sizes is effectively the same as normal backpressure.

When creating a simulation-based implementation of the stochastic dual gradient algorithm (using our MatLab-based discrete event simulation called MobSim), we discovered that a key constraint was missing from the published formulation that essentially allowed the virtual queue depth estimates to learn somewhat arbitrary values for the virtual queue depths, so that the goal of minimizing the number of physical packets was not always realized. To see this, assume that the correct value of γ_t is known, that μ is fixed, and that we need to solve for both λ_t and q_t. We then have one equation and two unknowns, which can give rise to degenerate solutions such as $\lambda_t = 0$ and $q_t = \gamma_t/\mu$ – which actually makes the physical queue sizes *larger*.

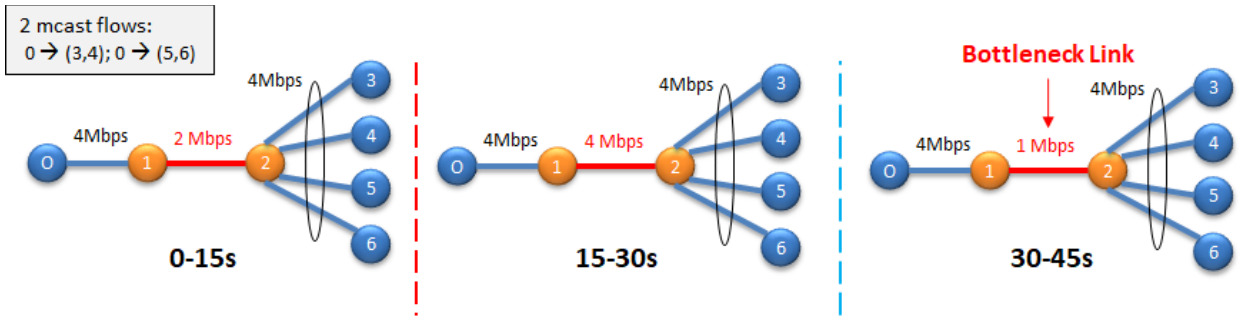
We addressed this by adding the specific constraint that the virtual queue values be at least $1/\mu$ times the number of physical packets when forwarding (typically we use $\mu = 0.2$, requiring that the virtual queue contributions be 80% of the total). We refer to this variant as the Improved Stochastic Dual Gradient (IDG).

Our MatLab simulation showed that for static scenarios, where the network characteristics and offered loads were not changing IDG did exceptionally well – meaning that the number of physical packets remained quite low and throughput variations (which occur when there are sizeable swings in queue depth difference between neighbors) were very small. This is shown in the multicast experiment shown below, where there are 5 different multicast groups with a variety of different receivers. For the baseline case without the use of virtual queues the aggregate throughput to node 6 was steady at 4 Mbps and the individual flows from each of the four multicast groups, while varying as much as 250Kbps, are all around 1 Mbps so reasonable capacity sharing. However, the end-to-end delays for the baseline case range as high as 6 seconds (!), and we see that at intermediate node 4 some of the multicast flows have as many as 50 packets enqueued.

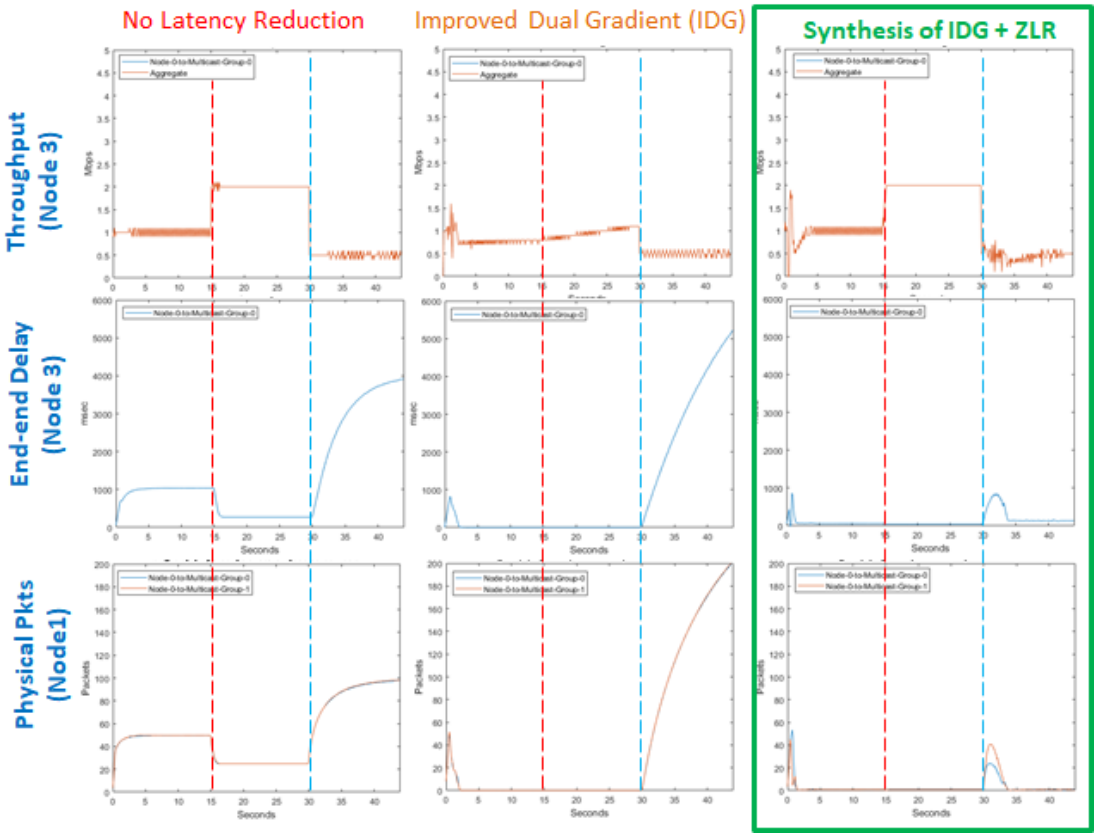
In contrast, with IDG running, the aggregate throughput remains the same, the per multicast group ingress rates are significantly smoother, the end-to-end delays are now under 9ms on the same topology, and the average number of packets enqueue at intermediate node 4 is now under 8 (for some flows there is essentially no packets enqueued at node 4 on average). In fact, with IDG keeping the aggregate rates just slightly below the network capacity, queues tend not to build up and queuing delays vanish. A clear and remarkable improvement over basic backpressure.



The problem with the Stochastic Dual Gradient and Improved Stochastic Dual Gradient algorithm is its inability to adapt once it has learned a set of virtual queue depths (the step size and learning rates are effectively zero after the initial virtual queue training time, which we illustrate with the experiment below (also conducted using MobSim). The experiment consists of two multicast flows, each to a different pair of destinations that share a bottle neck link highlighted in red in the figure below. We start with the bottleneck link set to 2 Mbps for the first 15s interval. For the second 15s interval, the bottleneck link rate is increased to 4 Mbps. For the final 15s interval the bottleneck link rate is set to 1 Mbps. We note here that the learning of the virtual queue depths by the Improved Stochastic Dual Gradient (IDG) is completed well before the end of the first 15s interval.



In the results shown below, we see that IDG seems to function reasonably well during the first interval where the network characteristics are consistent with those of when the virtual queue settings were learned. The throughput rates are low but not unreasonable, the queuing delays effectively vanish after the learning phase (as the network is being somewhat under driven) and the number of physical packets enqueued at the node at the head of the bottleneck link is correspondingly very low (effectively zero). When the rate of the bottleneck link suddenly increases to 4Mbps in the second 15s interval, we see that indeed the end-to-end delays remain close to zero, and hence the number of physical packets enqueued at the head of the bottleneck link is also zero. However, because IDG has effectively learned and remains fixed at a higher virtual queue value than is warranted at the increased link rate, the throughput during the second interval is nowhere near the expected 2 Mbps/flow provided with basic backpressure. Finally, during the third 15s interval when the rate of the bottleneck rate is reduced to 1 Mbps, we see that while the throughput rates are now at 0.5 Mbps (equal sharing of the link by the two flows), the end-end delays become large and eventually end up being much worse that basic backpressure.



2.2.3 Synthesis of IDG and ZLR: Improved ZLR

Our original design for ZLR of course specifically targeted dynamic environments, however it would often err on the side of keeping too many physical packets – which of course increases latency – to avoid situations where a destination with nothing but virtual packets enqueued is selected due to its having the largest gradient. Studying the IDG algorithm more closely during the virtual queue depth learning phase identified a nuance that, when included with ZLR, provides more IDG-like latency reduction performance without the drawbacks, as is shown in the above figure. In particular, IDG's virtual queue learning phase specifically evaluates the local virtual queue depths against the advertised queue depths from its neighbors, and if the gradient is positive it will dequeue virtual packets even if there are physical packets that can be sent for the given destination. This early adjustment addresses large disparities in queue depths (caused by queue oscillations or changes in link rates) early rather than waiting for a queue to run out of physical packets as with the original ZLR algorithm. This latter case (waiting until all physical packets have been transmitted) tends to stall the entire forwarding pipeline and in the process introduce queue depth oscillations – which increase queueing delays. This change has been incorporated into the GNAT ZLR implementation, and has shown comparable gains in performance.

2.3 MatLab Model for Computing Latency-constrained Network Capacity

In the following, we determine the maximum rate at which information can be multicast over a network subject to end-to-end latency constraints. The network model consists of N nodes which are interconnected by links which have a capacity C and a latency L. The source and sinks for the multicast group are specified and the goal is to determine the maximum achievable throughput such that information is received by all destinations within a deadline D. This model assumes that the links are reliable, the latency is fixed, there is no queuing delay and the same rate must be achieved to all multicast destinations.

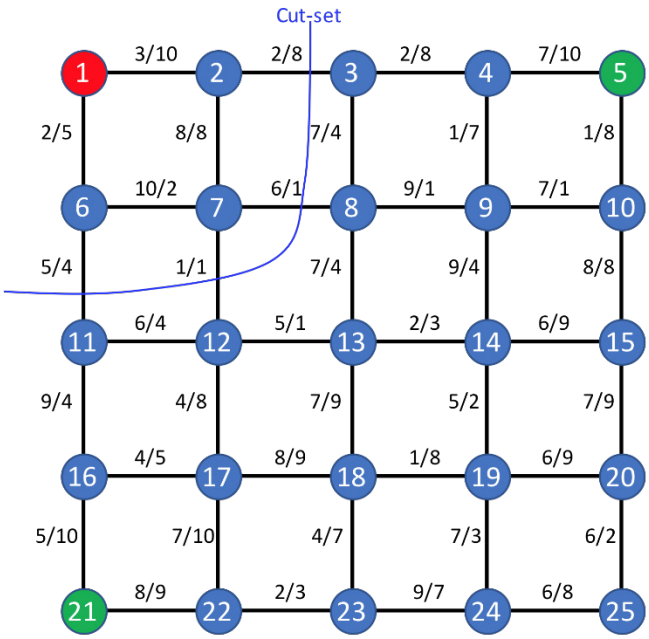
In the absence of latency constraints, the optimal solution can be obtained by using backpressure forwarding and network coding. Network coding effectively "convexifies" the multicast problem, allowing information to flow over multiple paths to destinations. The maximum multicast throughput (without latency constraints) can be determined off-line by solving a linear programming problem. The LP is formulated in terms of per-node flow-conservation equations and link capacity constraints. The size of the LP therefore scales linearly with the number of nodes, links and multicast destinations which means that quite large network problems can be solved.

The addition of the end-to-end latency constraint complicates things significantly. The hard latency constraint means that packets are not allowed to flow over any *path* with an end-to-end latency that exceeds the deadline. The linear programming approach can be extended to incorporate latency constraints; however, it requires working with flows over paths rather than with per-node flow constraints. Unfortunately, the number of paths in a network is an exponential function of the size of the network. For example, in a 9x9 grid network, there are over 3 quadrillion ($3\ 10^{15}$) paths between the diagonally opposite corners. This raises two problems: (i) finding the paths that meet the latency constraint and (ii) solving an LP with a potentially large number of paths.

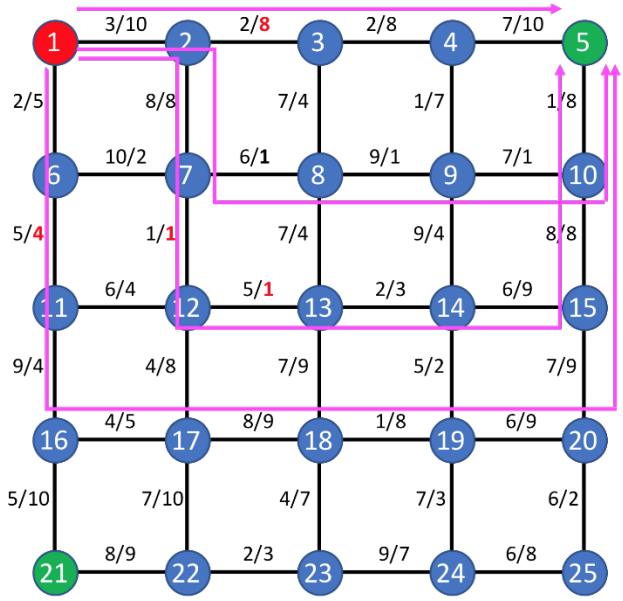
We note that solving an LP is ***not*** likely to be useful as an approach for developing actual latency-constrained multicast algorithms. Rather, since it is maximum throughput that can be achieved by *any* algorithm, it is useful for determining how well the heuristic algorithms we will develop operate.

Our approach to solving the latency-constrained maximum-throughput multicast problem leverages the recent development of efficient algorithms for representing graphs using zero-suppressed decision diagrams (ZDDs). The Graphillion library provides algorithms for constructing graphs and enumerating paths that meet certain constraints. While the size of the ZDDs is still exponential in the number of nodes and links, they can be used to efficiently represent networks up to about 100 nodes. We use graphillion to find paths that meet the latency constraint and then solve an LP program that maximizes multicast throughput over these paths. For problems where the latency constraint is "tight" – i.e., ones where the number of feasible paths is under a million – we can solve the LP exactly. For problems where the number of feasible paths is much larger, we will look at computing solutions using a subset of the feasible paths.

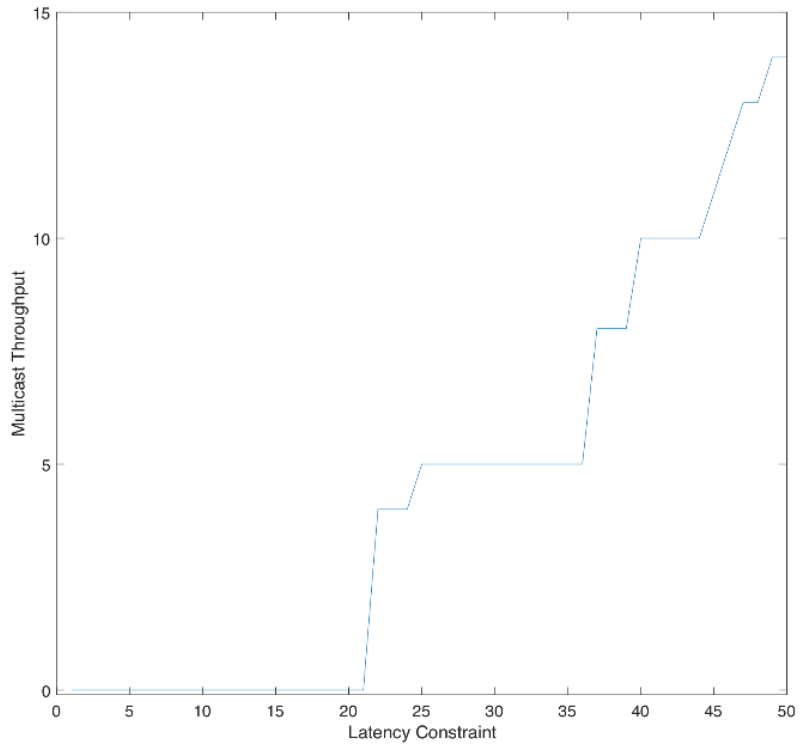
Consider the 5x5 grid network illustrated below. It contains 25 nodes and 40 links. The notation on the links indicates the latency/capacity of the link. For example, the link from node 1 to node 2 has latency 3 and capacity 10. The source is node 1 and the destinations are nodes 5 and 21. In the absence of latency constraints, the maximum multicast rate is 14. This can be seen by noting that the minimum cut set capacity is 14. The associated links are the ones that are "cut" by the thin blue line. This is lower than the maximum rate at which the source can transmit (5+10=15) and the rates at which the destinations can receive (node 5: 10+8=18; node 21: 10+9=19). Note that if node 25 were also a destination, then the maximum multicast rate would be 10, since that is the minimum cut set capacity for separating node 1 and 25 (the inbound links have capacity 2+8=10).



To achieve this capacity requires the use of multiple paths. For example, four paths are required to transmit 14 units of traffic from node 1 to node 5. The figure below shows one possible solution. The bold red numbers indicate the bottleneck link in each path. A similar mesh of paths is required to transmit 14 units of traffic from node 1 to node 21. Both sets of flows can be sustained simultaneously through the use of network coding.



If an end-to-end latency constraint is introduced, then the multicast capacity is reduced. The following plot shows how capacity varies as the latency constraint changes from 0 to 50. Note that when the latency constraint reaches 49, the multicast capacity is 14 – the unconstrained value – and thus it cannot increase any further.



The CPU time required to solve the latency constrained multicast problem depends on the size of the graph, the number of destinations and the number of paths meeting the latency budget. The 5x5 grid is solvable in a few seconds for the example above.

The following table gives an example based on a larger 8x8 grid with 2 multicast destinations. The first column is the latency deadline, the next is the number of paths to the first destination that meet the latency constraint, the third is the CPU time required to enumerate those paths, the next two columns are the number of paths and CPU time for the second destination, the sixth column is the time required to solve the linear program, the seventh is the total CPU time for the Matlab program and the final column is the optimal (maximum) multicast rate.

Deadline	Paths to A	CPU	Paths to B	CPU	LP	Total CPU	Rate
100	27,243	21.02	425,381	19.32	15.17	84.5	10
90	4,306	10.48	91,362	9.55	2.20	27.8	10
80	778	4.75	17,544	4.07	0.37	10.2	10
70	193	1.70	3,241	1.70	0.11	4.2	10
60	62	0.95	578	0.75	0.06	1.9	10
50	21	0.56	134	0.45	0.07	1.2	10
40	7	0.62	24	0.36	0.07	1.2	5
35	5	0.44	8	0.34	0.06	0.9	4
30	3	0.43	1	0.35	0.07	0.9	3

2.4 Exploring Alternative Signaling and Dequeuing Strategies using ns-3

Based on Tracy Ho's paper on using multicast-over-backpressure, optimal multicast signaling and dequeuing relies on network coding and uses per-group per-bin queue depths to select the neighbor with the largest gradient over all multicast queue gradients (by taking the sum of the negative gradients). Our current design in GNAT also relies on exchanging per-multicast group per-bin queue depth signaling which can quickly get quite large. Here we explore using only per-bin queues, where the bins are only for destination nodes (i.e., unicast signaling) and independent of the multicast group.

2.4.1 Alternative strategy:

Assumption: Each node knows the per-group virtual queues of all its neighbors.

Each node advertises

- The total-per-bin count for each destination. This is summed across all multicast group.
- The total-per-group count for each multicast group for which it has traffic.
    - Note that this is a count of bin weights and not a packet count. I.e. if a packet is marked for destinations (1,2), then it will count as 2 packets (once for destination 1 and once for destination 2).
        - Note: this would this could work better if it also included a total-per-group packet count. This would be more consistent with the admission control decision, but is a harder reach in NS3.
    - The count does not include bins for which the node's virtual queue depth is infinite.

A packet is selected for dequeuing using a 2-step approach.

First, we use the total-per-group count to select the multicast group with the largest gradient. This is an approximation to what the optimal algorithm with per-group per-bin does. With per-bin per-group information, the gradient is the sum of only the negative gradients, while our approximation gets the sum of both the negative and positive gradients (excluding the infinite gradients).

Once we select the group with the largest gradient, we select the packet from that group with the largest gradient using the per-bin queues. These are the number of packets for each destination node across all multicast groups. This is an approximation in that the optimal algorithm with per-group per-bin information would only look at the queues for packets within the target multicast group (which in information we don’t have).

2.4.2 Caveats:

We’re using different information to make two different approximations which can lead to conflicts: We select the group with the largest negative gradient (determined by total-per-group info), then we select the packet with total-per-bin info. It is possible (and it does happen) where we select a group, but there are no packets with a negative gradient according to the total-per-bin information. This can happen if different groups have a lot of packets in the queues belonging to the group we have chosen. In this case, we need to send something, else we will make the same decision again and again since there is no change.

What is done here is to select the packet with the smallest positive gradient and send that, marking the destination as all destinations in the packet for which the neighbor does not have infinite virtual queues i.e. we try really hard to avoid sending packets to dead ends as this really adds to the noise in the

goodput plots. This is particular interesting as one approximation says we're sending uphill and another says we're sending downhill.

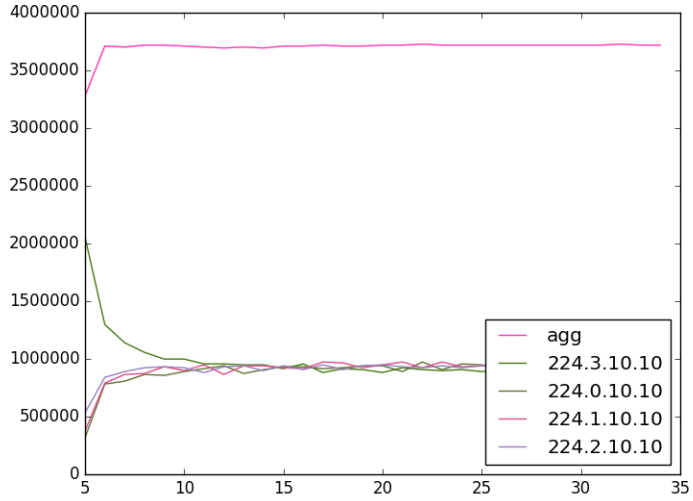
Another embellishment, which resulted in a little more smoothing, is trying to avoid breaking packets. i.e. if we can send a packet to {2,3} to a downstream node, but not to {1}, it's better to choose a packet that has only {2,3} than to break a {1,2,3} packet into {1} and {2,3}, as there is some chance you can send the {1,2,3} to a different neighbor or to the same neighbor at a later time.

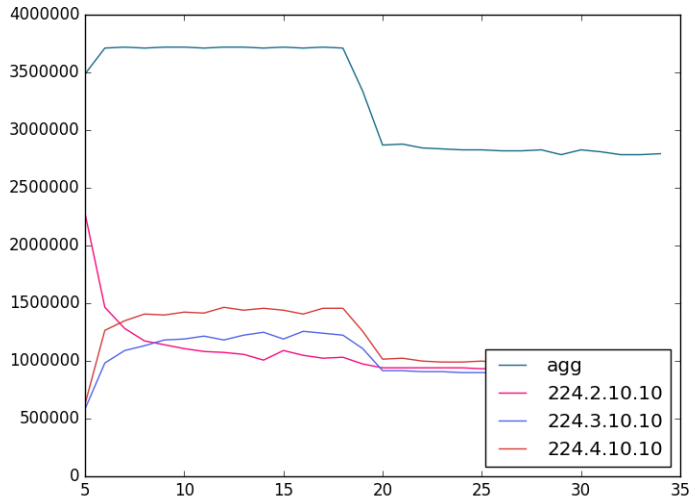
2.4.3 Experimental results.

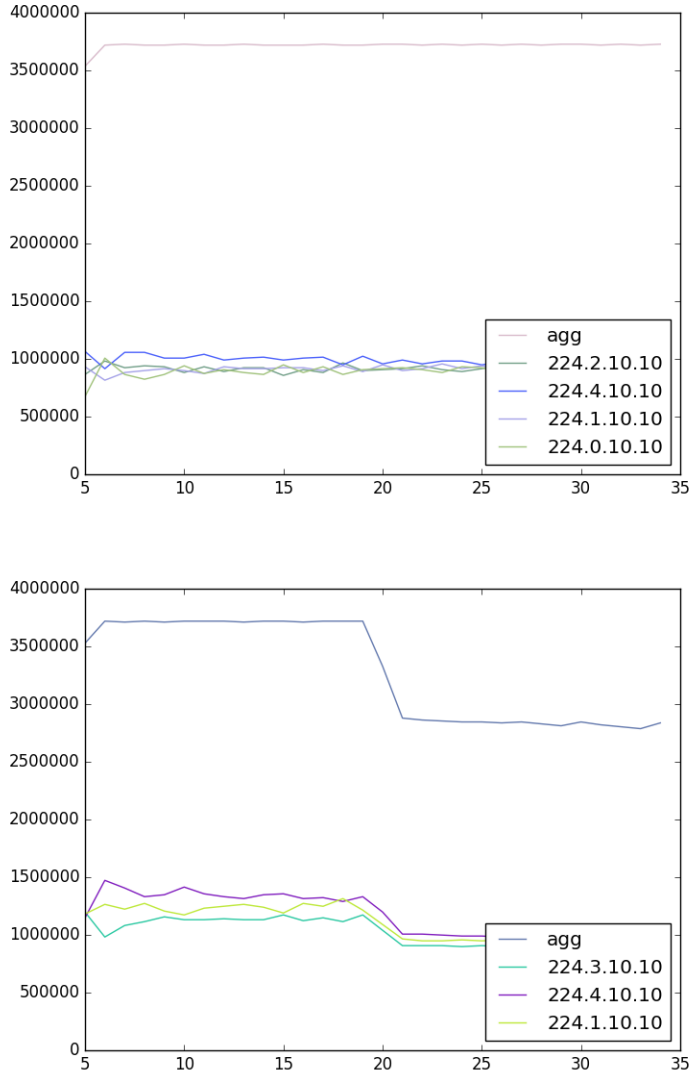
We consider a 5 group experiment with the following traffic pattern:

- 0 -> (6,7) with p = 1 (224.0.10.10)
- 5 -> (0, 6, 7) with p = 1 (224.1.10.10)
- 2 -> (5, 6, 7) with p = 1 (224.2.10.10)
- 3 -> (0. 5. 7) with p = 1 (224.3.10.10)
- 7 -> (0, 5, 6, 7) with p = 1 (224.4.10.10

Using per-group per-bin information, we get the following goodput results:


4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
5
10
15
20
25
30
35
agg
224.3.10.10
224.0.10.10
224.1.10.10
224.2.10.10

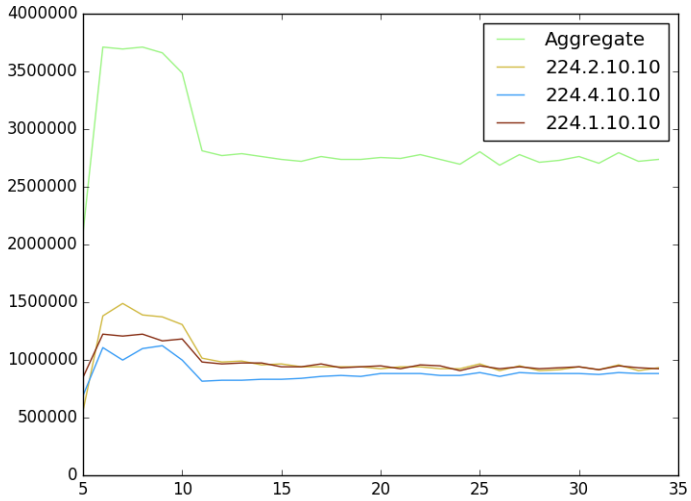

4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
5
10
15
20
25
30
35
agg
224.2.10.10
224.3.10.10
224.4.10.10

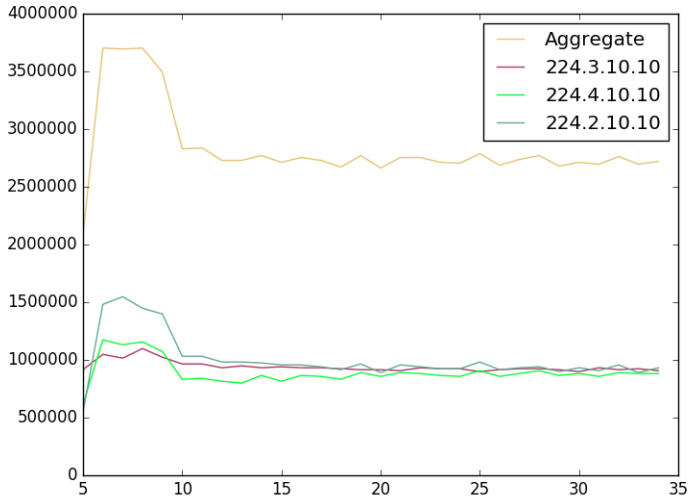


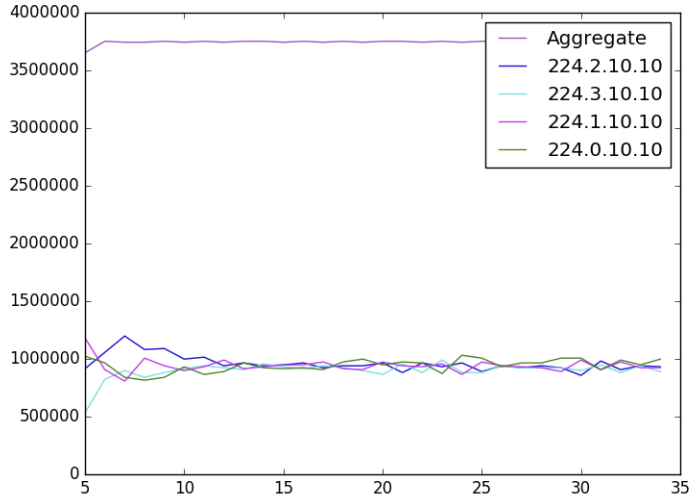
All of the flows get ~1Mbps once converged. Some flows take longer to converge ~17 seconds!

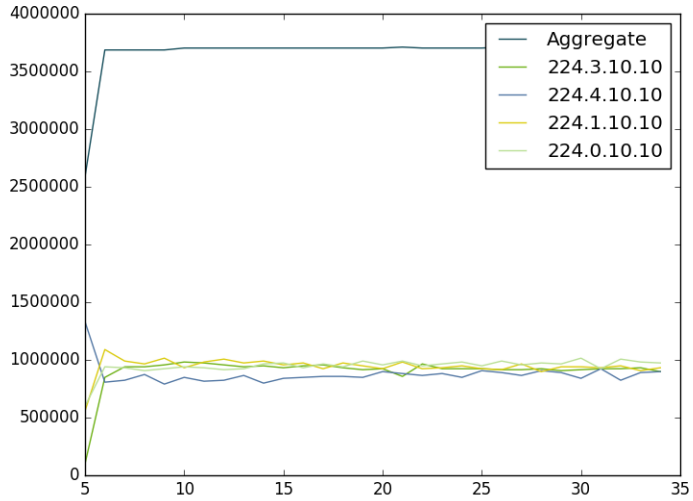
The long convergence delay appears to be a direct cause of 1) having a large value of K that allows building up a large queues at the source nodes, and 2) being able to drain packets to nodes 0 and 5 faster than we are able to drain traffic to the other nodes: specifically, nodes 0 and 5 only have 3 inbound flows each whereas the other two nodes have four flows each. In this situation, the backpressure forwarder is initially able to "split" packets entering the queue and deliver to nodes 0 and 5 at a higher rate. Once the downstream queues have built up sufficiently to where the admission controller is adding packets at the same rate that they are being delivered to the slower nodes, there are then no "extra" packets that can be split and used to deliver to nodes 0 and 5.

As experimental evidence, we repeat the above experiment with a lower value of K: specifically, for reduce K from 15000.0 pkts^2/s (1e12 bits^2/s) to 3000.0 pkts^2/s (2e11 bits^2/s). In the plots below we see that the convergence delay is much shorter.

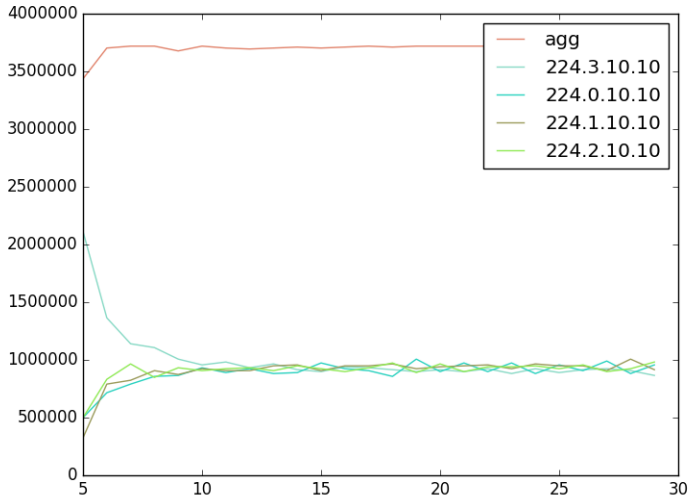


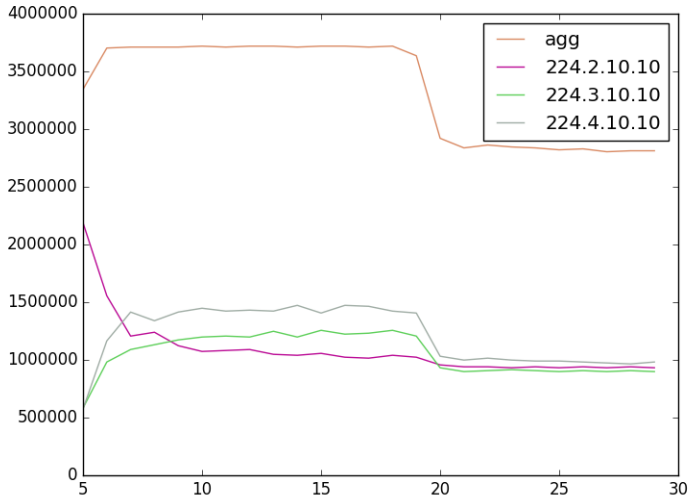


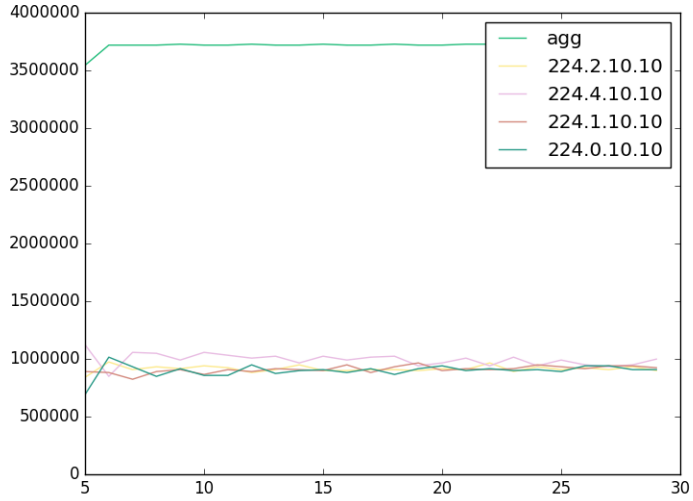

4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
5
10
15
20
25
30
35
Aggregate
224.2.10.10
224.3.10.10
224.1.10.10
224.0.10.10

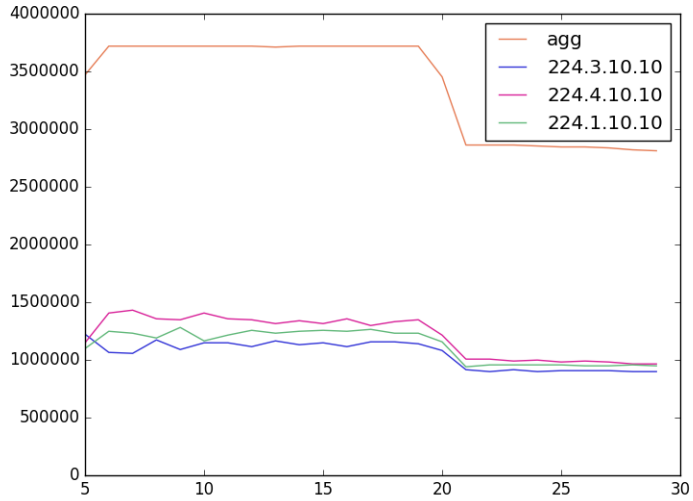

4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
5
10
15
20
25
30
35
Aggregate
224.3.10.10
224.4.10.10
224.1.10.10
224.0.10.10

Using per-group per-bin information, without network coding (and the original K value of 15000 pkts^2/s), we get the following results:



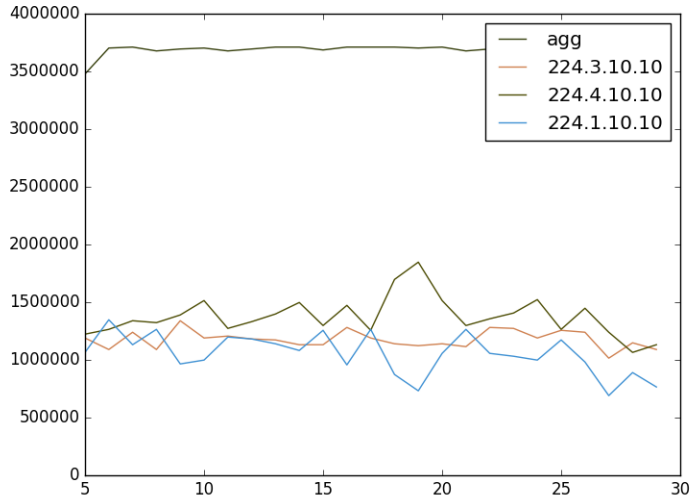


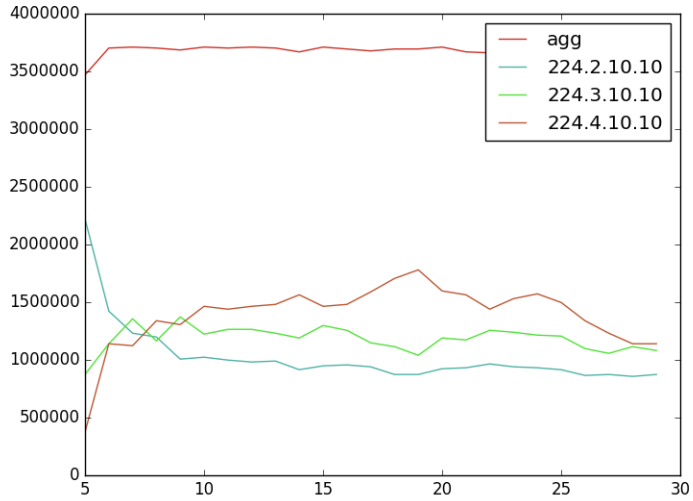


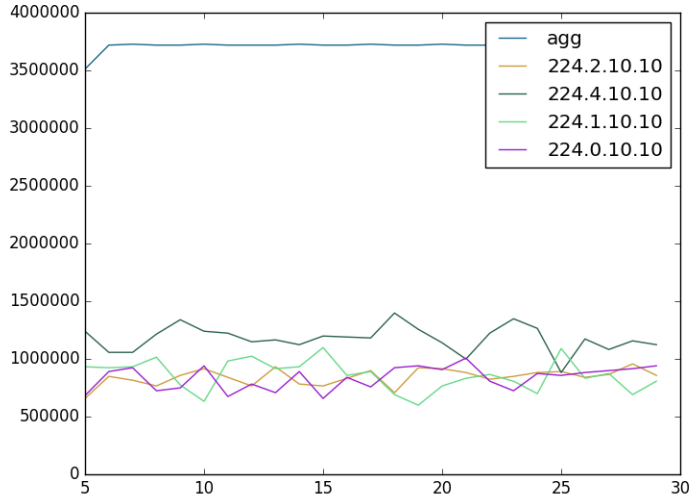


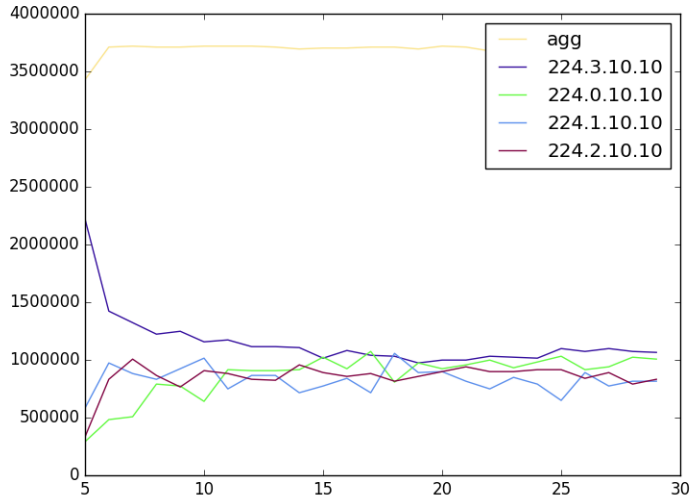
We see that the results are almost identical to that with network coding, which suggests that in this scenario network coding has no benefits in terms of goodput.

The next set of results uses total-per-group and total-per-bin as an approximation of the per-group per-bin information:

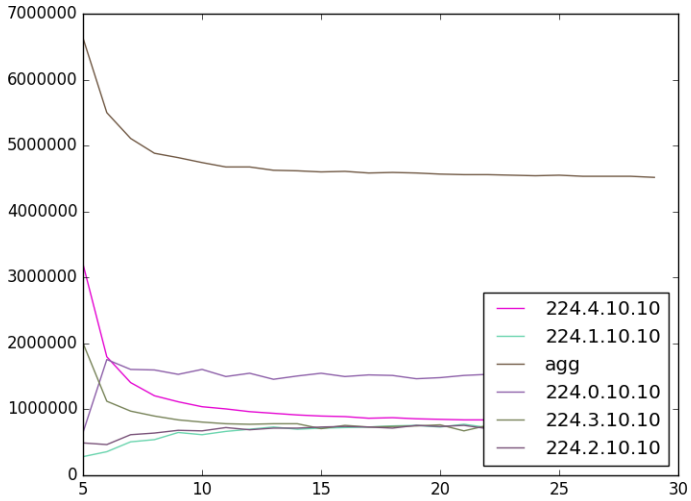


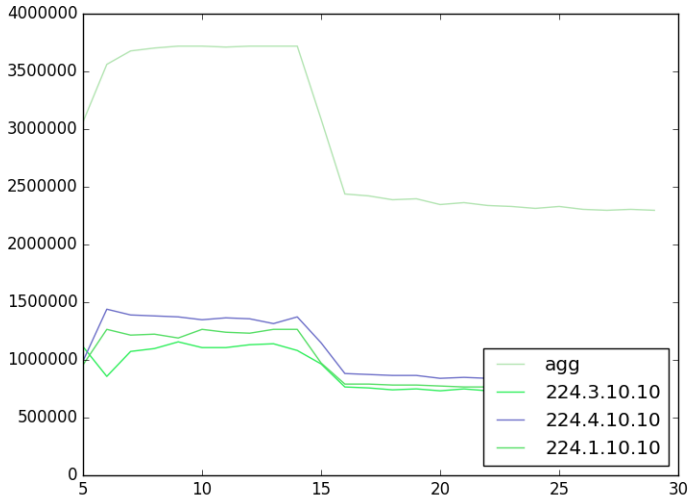


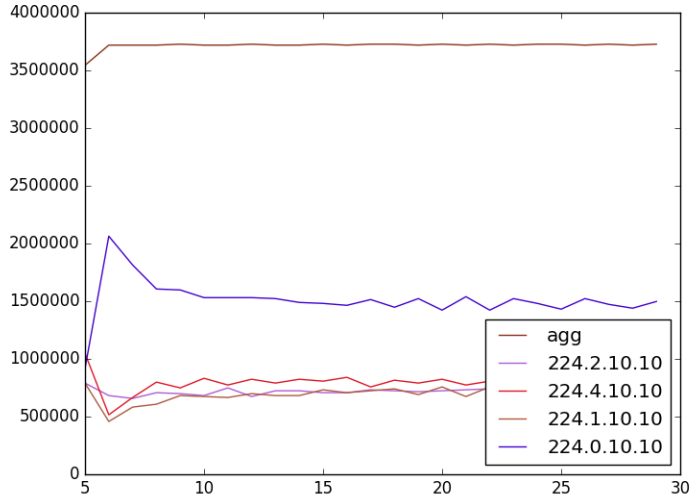

agg
224.2.10.10
224.4.10.10
224.1.10.10
224.0.10.10
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
5
10
15
20
25
30

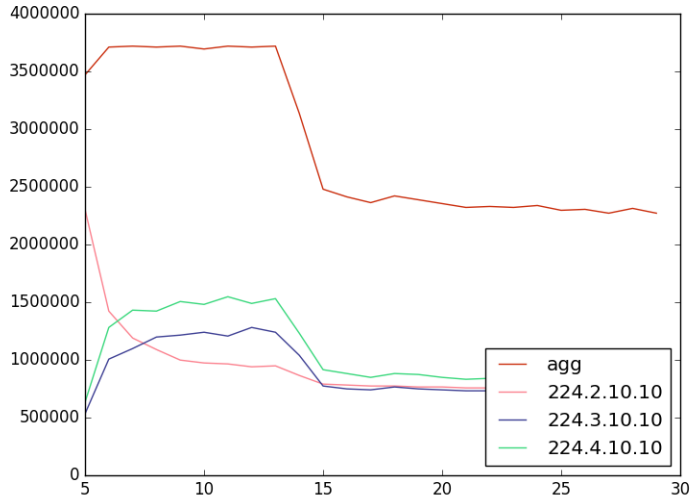

agg
224.3.10.10
224.0.10.10
224.1.10.10
224.2.10.10
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
5
10
15
20
25
30

In the following experiments we set the priority of the multicast group 224.0.10.10 to p=2, while keeping the other priorities p=1. The results for the optimal algorithm (network coding with per-group per-bin information) is shown below.



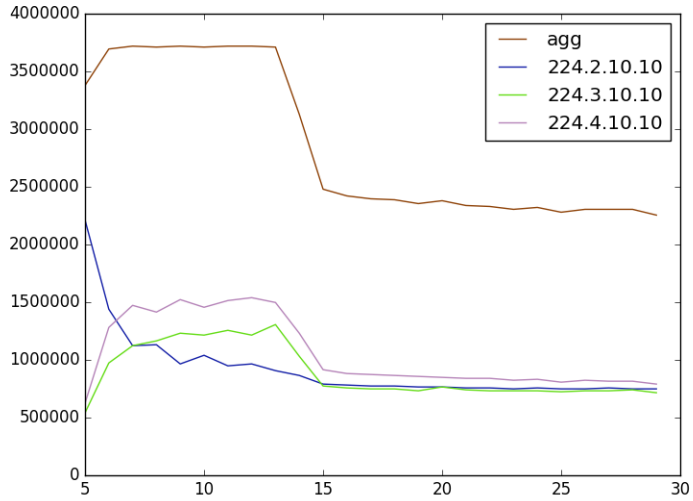


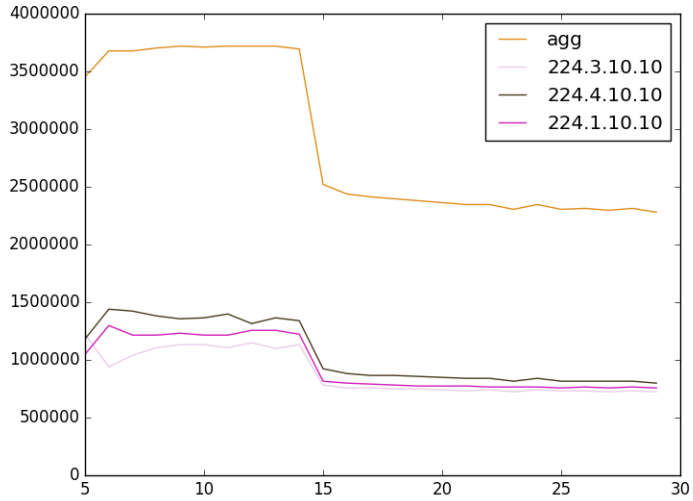


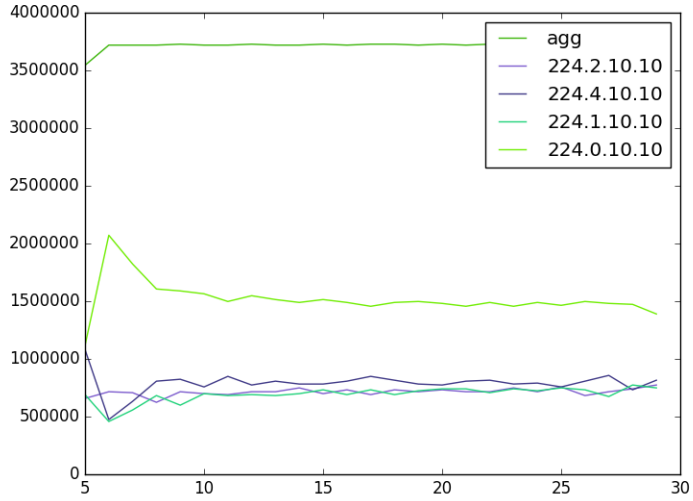


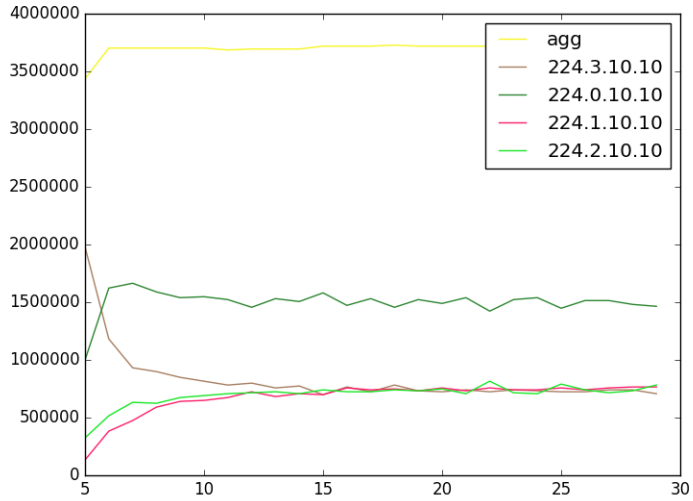
We see a 2:1 ratio between the priority 2 flow and the priority 1 flows.

The next set of plots shows similar results when using per-group per-bin information but without network coding.

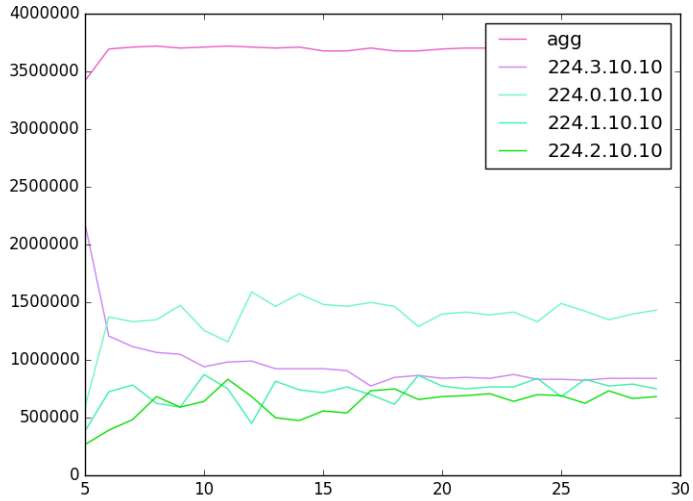


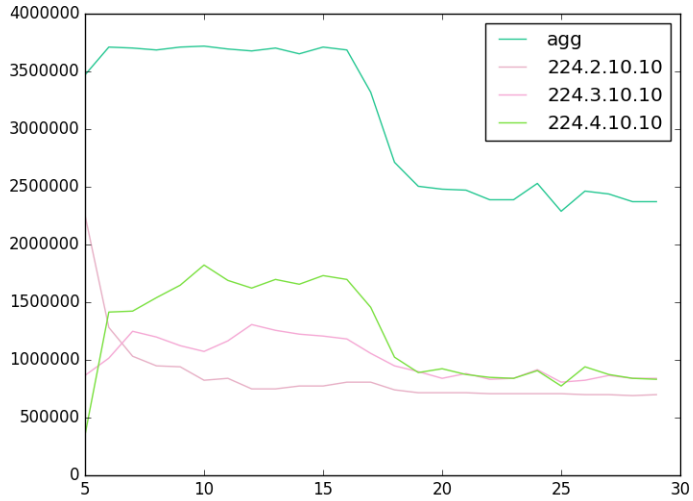


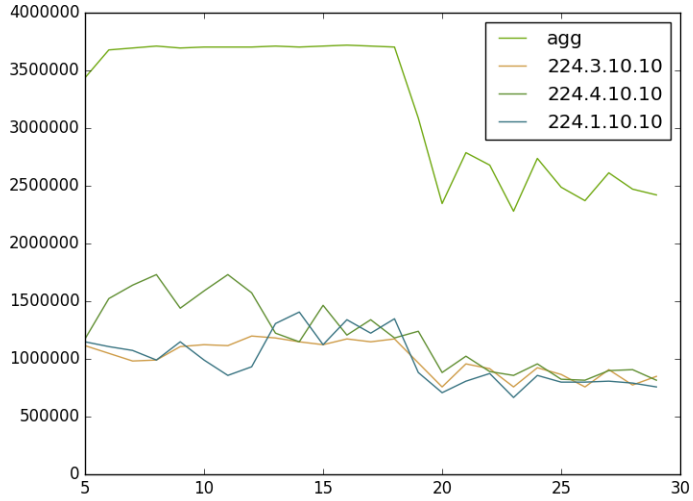

4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
5
10
15
20
25
30
agg
224.2.10.10
224.4.10.10
224.1.10.10
224.0.10.10

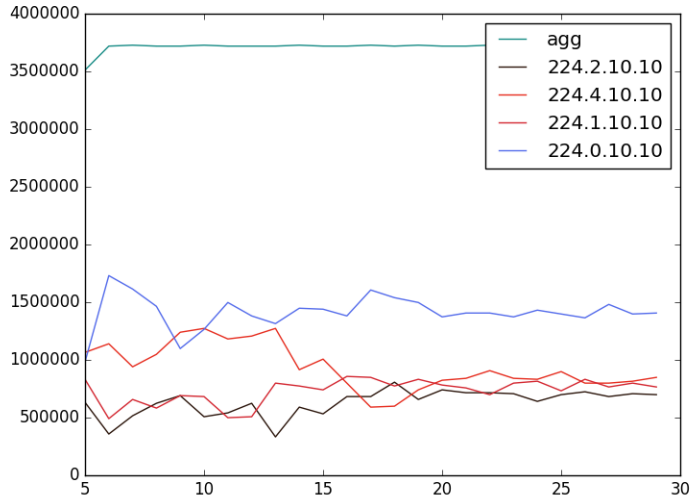

4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
5
10
15
20
25
30
agg
224.3.10.10
224.0.10.10
224.1.10.10
224.2.10.10

The next set of results show the goodput when the approximation is used (total-per-group and total-per-bin):









We see the expected 2:1 ratio, but there is a little more noise.

3 Measurement-based Component Performance Assessments

3.1 SLIQ Congestion Control Algorithm Characterization

This section provides the results of a set of characterization tests performed on the SLIQ (Simple Lightweight IPv4 QUIC) protocol when using the CUBIC and Copa congestion control algorithms. The CUBIC algorithm is the same as that in TCP, but adapted to work with the SLIQ protocol. The Copa algorithm was designed by MIT and was also adapted for use with SLIQ.

3.1.1 Test Setup

3.1.1.1 Congestion Control Algorithms (Single Algorithm Configurations)

SLIQ currently supports two primary congestion control algorithms: CUBIC and Copa.

The CUBIC congestion control algorithm implemented in SLIQ is derived from the CUBIC congestion control algorithm used in TCP. The SLIQ version of CUBIC adds support for hybrid slow start, proportional rate reduction, and send pacing, and all of these features were enabled during this testing. The CUBIC algorithm is loss-based, which means that it uses packet losses to estimate the amount of congestion at the bottleneck link between the two endpoints. It does this by assuming that all packet losses are due to switch buffer overflow.

The Copa congestion control algorithm was designed by MIT and implemented by the IRON team within SLIQ. This algorithm is delay-based, which means that it uses packet RTT calculations to estimate the amount of congestion at the bottleneck link between the two endpoints. This is fundamentally different from CUBIC, and allows Copa to operate with very little packet loss. The Copa algorithm can be summarized as follows:

- Computes a target packet send rate based on the estimated switch delay at the bottleneck link and an aggressiveness parameter called "delta".
- Maintains a congestion window to control how many packets are outstanding (unacknowledged) at any point in time.
- Computes a current packet send rate based on the current congestion window and the average RTT.
- Estimates the current switch delay from a small sample of RTT measurements to eliminate any high frequency jitter.
- Adjusts the congestion window based on a comparison of the current packet send rate (equal to the congestion window size divided by the average RTT) and the current target packet send rate (equal to one over delta times the estimated switch delay). If the current congestion window send rate is lower, then the congestion window size is increased linearly. Otherwise, the congestion window size is decreased linearly.
- Uses a fixed delta value of 0.5.
- Does not randomize the packet inter-send times. The packet inter-send time is equal to one over the current packet send rate.
- Tracks the minimum RTT over an interval to adapt to network routing changes. All tests were performed with this mechanism enabled.
- Includes a TCP-compatibility mode. However, this feature is not implemented in the source code and was not used in these tests.
- Packet transmissions and retransmissions are paced using the current packet inter-send time, while obeying the current congestion window.

Note that for the Copa algorithm, the number of packets that the algorithm attempts to keep enqueued at the bottleneck switch is equal to (1 / delta). In order to handle variable-sized packets, the Copa computations use a nominal packet size of 1000 bytes. Thus, since delta is set to 0.5, the Copa algorithm attempts to maintain 2 packets (or 2,000 bytes) enqueued at the bottleneck switch. CUBIC, on the other hand, fills the bottleneck switch to the point of overflowing, then attempts to stabilize at the point where the switch's input queue remains nearly full.

The default congestion control used in the IRON system is currently "Cubic,Copa". This uses the CUBIC and Copa algorithms in parallel, with preference for the CUBIC algorithm when possible. The following test results were run using the CUBIC and Copa algorithms by themselves in order to characterize their individual behaviors and resulting performance.

3.1.1.2 Software and Equipment

All of the following test results were produced using the SLIQ *test_sliq* program compiled in optimized mode. During each of the tests, a single stream of data was sent from client to server, and the client application was configured to send 1000 byte packet payloads.

For the maximum performance tests, the BBN IRON Testbed 8X was used. The server-side program was run on iron80 (dual-core Intel Xeon 1.87 GHz processor with 4 Gbytes of main memory), and the client-side program was run on iron83 (dual-core Intel Xeon 1.87 GHz processor with 4 Gbytes of main memory). The IRON link emulation program *LinkEm* was not used during the performance tests. The link between iron80 and iron83 is Gigabit Ethernet, which limits the results to 1 Gbps minus the rate required for the packet overhead (1 Gbps minus ~70 Mbps, which equals ~930 Mbps).

For all other tests, the BBN IRON Testbed 9X was used. The server-side program was run on iron94 (dual-core Intel Xeon 1.87 GHz processor with 8 Gbytes of main memory), and the client-side program was run on iron95 (dual-core Intel Xeon 1.87 GHz processor with 6 Gbytes of main memory). The IRON link emulation program *LinkEm* was run on iron97 to control the link characteristics (delay, throttling, packet error rate (PER), and bottleneck switch queue size) between iron94 and iron95.

3.1.2 Maximum Performance

This test measures the maximum goodput (application data throughput) of a single stream of data sent from the client to the server. It is a test of the efficiency of the protocol software implementation as well as a test of the ability of the congestion control algorithm to stabilize at a high data rate with very small RTT measurements.

As mentioned above, the use of a Gigabit Ethernet link and the SLIQ packet overhead limits the results to ~930 Mbps. Each individual test was configured to run for 60 seconds in order to minimize the effect of startup transients. The listed results are taken at the server (receiver-side), and are computed using the total number of application bytes received and the amount of time elapsed from the receipt of the first byte until the last byte. Five consecutive runs were performed for each congestion control algorithm, and the five results were averaged to generate the listed values.

The results are as follows.

SLIQ CUBIC	SLIQ Copa
472 Mbps	458 Mbps

The CUBIC congestion control algorithm achieved a slightly higher result than Copa, with Copa being only 3% lower than CUBIC. In each of these tests, the send-side CPU core running *test_sliq* was at 100%

usage, while the receive-side CPU core running *test_sliq* was at 30% usage. This indicates that these results were limited by the processing power of the computers running the tests, not by the congestion control algorithms. A previous set of these tests on the same hardware using an older implementation of Copa only achieved 195 Mbps, so the Copa algorithm has improved significantly since that time.

Previous maximum performance tests run on the similar hardware with earlier versions of SLIQ posted results of more than 770 Mbps. The performance drop since that time is most likely due to the current SLIQ software implementation being more complex in order to add necessary features. These features require storing more per-packet information and performing more per-packet processing on the send-side, leading to the CPU core running *test_sliq* to hit 100% utilization at a lower packet send rate. It is likely that these results would be much better if run on faster computers.

3.1.3 Distributed Capacity Sharing

This test measures how well two independent SLIQ flows using identical congestion control algorithms are able to fairly share a bottleneck link that has its capacity change over time. The ideal result is having the two independent flows share the link equally at all times, regardless of the current link capacity.

The following tests were run with two separate pairs of *test_sliq* instances operating in parallel, each sending a single data flow from the client host on iron95 to the server host on iron94. The *LinkEm* program was run on iron97 with the following configuration parameters:

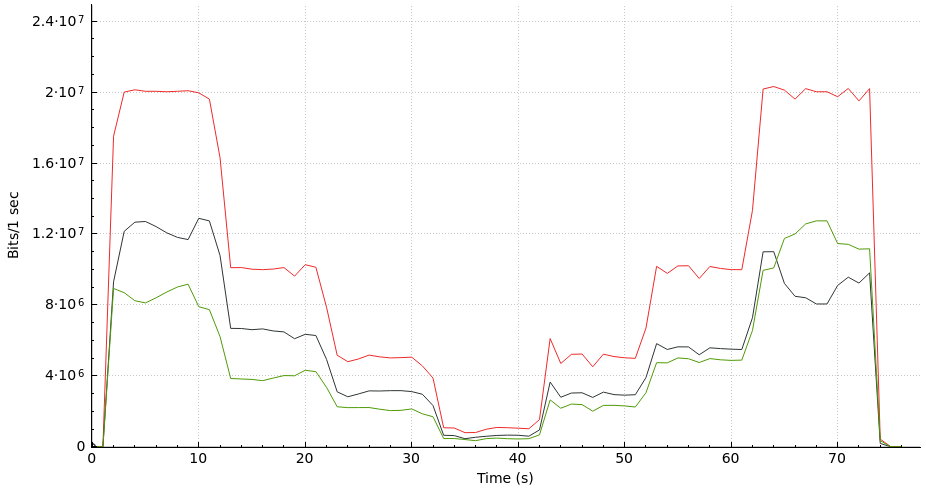
- 20 ms one-way delta (40 ms RTT)
- 200,000 byte maximum conformal buffer size (which is two times the maximum bandwidth-delay product)
- PER set to 0
- Throttling set to the following rates on 10 second intervals: 20 Mbps, 10 Mbps, 5 Mbps, 1 Mbps, 5 Mbps, 10 Mbps, 20 Mbps

During each test, packet traces were captured on the client outbound and server inbound interfaces using *gulp*. These packet traces were then plotted using *wireshark* to show each application's client-to-server traffic and the combined client-to-server traffic. In each of the plots, the first application traffic is plotted in black, the second application traffic is plotted in green, and the combined traffic is plotted in red.

3.1.3.1 SLIQ CUBIC

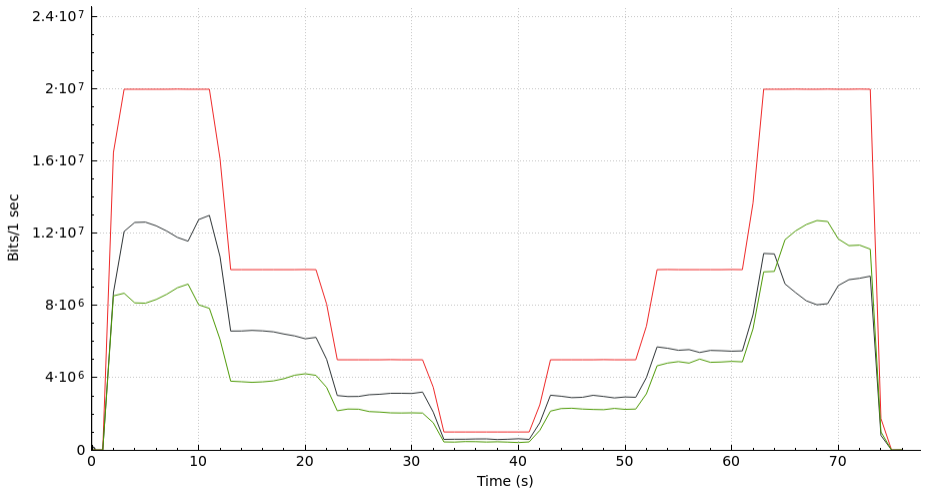
Below is a plot of the network traffic leaving the client node (the data source) when running SLIQ with the CUBIC congestion control algorithm.

Wireshark IO Graphs: char_sharing_cubic_client



Below is a plot of the network traffic arriving at the server node (the data sink) when running SLIQ with the CUBIC congestion control algorithm. Note that this traffic appears after the link emulation has been applied, so the combined traffic line is limited by the *LinkEm* throttling values.

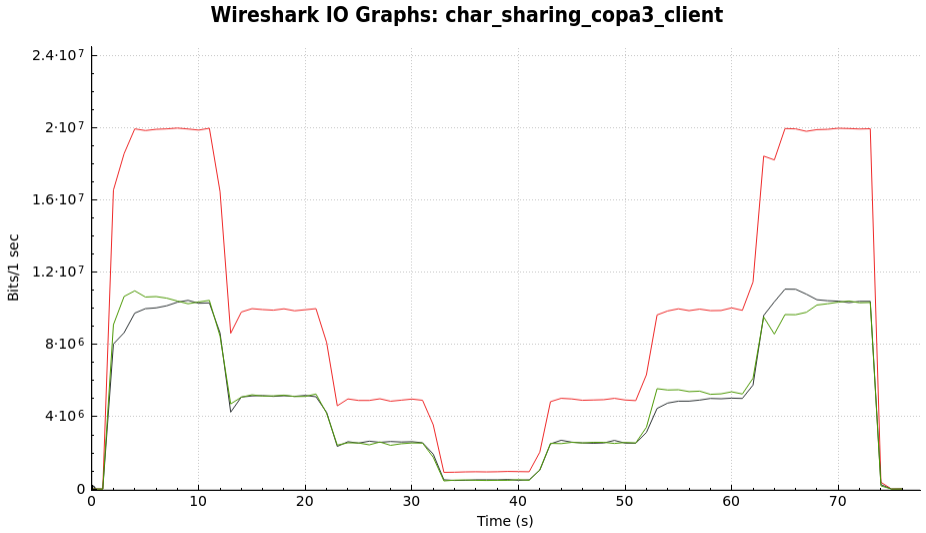
Wireshark IO Graphs: char_sharing_cubic_server



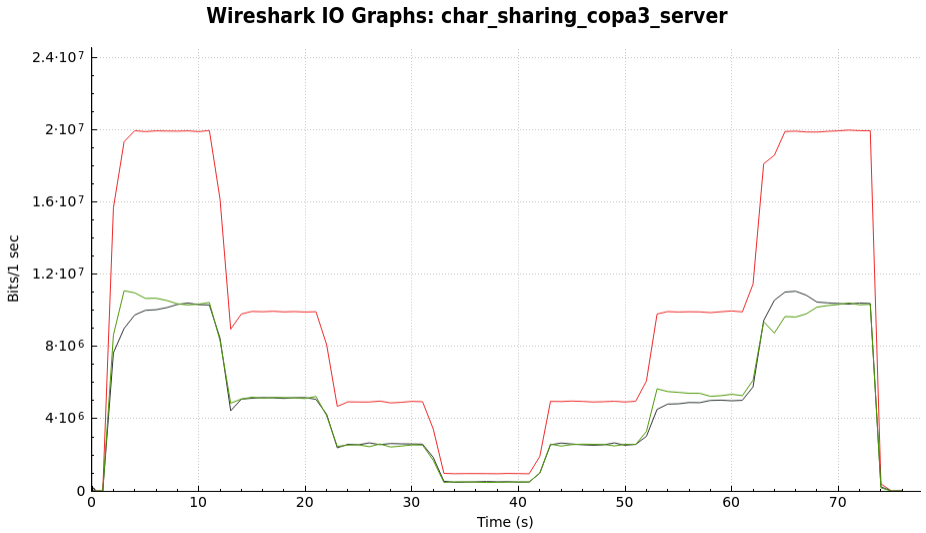
From the first plot, CUBIC causes SLIQ to transmit packets slightly faster than the channel allows in order to induce packet losses, which it uses to determine when the channel capacity has been reached. In both plots, note how the two flows start at roughly the same time, but do not share very equally until around 33 seconds into the test. After this point, sharing is better, but still not perfect: there is a lot of unevenness in the two send rates, and toward the end of the test, the first application (in green) starts getting more than its share of the available channel capacity again. These results are reasonably good, as the two applications keep the channel full, but there is room for improvement with how well the channel is being shared.

3.1.3.2 SLIQ Copa

Below is a plot of the network traffic leaving the client node (the data source) when running SLIQ with the Copa congestion control algorithm.



Below is a plot of the network traffic arriving at the server node (the data sink) when running SLIQ with the Copa congestion control algorithm. Note that this traffic appears after the link emulation has been applied, so the combined traffic line is limited to the *LinkEm* throttling values.



From the first plot, note how Copa sends at just about the exact channel capacity, since it is attempting to fill the channel without inducing any packet loss. During the first 20 Mbps period, the combined traffic adapts to the full channel capacity just about as fast as with CUBIC. In both plots, note how the sharing between the two applications is disrupted during the throttling changes less than with CUBIC, and improves faster than with CUBIC. The three periods where the sharing is not ideal are the two 20 Mbps periods and the second 10 Mbps period, but Copa still does better than CUBIC at these points.

3.1.4 Packet Loss Response

This test measures how well a single SLIQ flow handles non-congestion packet losses in the network. The goal is to have a congestion control algorithm that can continue to move reasonable amounts of data even though some portion of the packets are being dropped for reasons other than network congestion (e.g., network attacks). Here, non-congestion-based packet losses are modeled by a random drop process within *LinkEm,* with a dynamically controllable average packet error rate (PER). The PER applies to all packets flowing in either direction, and is independent of the packet sizes.

The following tests were run with one flow originating at the client host on iron95 and terminating at the server host on iron94. Each test was 60 seconds long to minimize the impact of the protocol starting up. The *LinkEm* software, running on iron97, was configured with the following parameters:

- 20 ms one-way delay (40 ms RTT)
- 10 Mbps throttling
- 100,000 byte maximum conform buffer size (which is two times the maximum bandwidth-delay product)
- PER set to the following values: 0, 0.003, 0.01, 0.03, 0.1, 0.3, 0.4, 0.5

The following table contains the measured SLIQ application goodput, in megabits per second (Mbps), as a function of the PER. Included are the theoretical maximum goodput values based on the SLIQ data packet size (1058 bytes total, which includes 14 bytes of Ethernet frame, 20 bytes of IP header, 8 bytes of UDP header, 16 bytes of SLIQ data header, and 1000 bytes of payload) and results for the Linux kernel's version of TCP with CUBIC.

PER	Theoretical Maximum	Linux TCP CUBIC	SLIQ CUBIC	SLIQ Copa
0.0	9.452	9.280	9.412	9.398
0.003	9.423	3.180	5.067	9.366
0.01	9.357	1.600	2.517	9.289
0.03	9.168	0.914	1.210	9.094
0.1	8.507	0.351	0.490	8.392
0.3	6.606	0.000	0.105	6.284
0.4	5.671	0.000	0.066	5.338
0.5	4.726	0.000	0.024	4.024

Note how SLIQ CUBIC performs slightly better than the Linux kernel's TCP CUBIC. This is most likely due to the differences in the TCP and SLIQ protocols. However, SLIQ Copa achieves much better results than CUBIC, even when only 0.3% of the packets are lost. When 30% of the packets are lost, the goodputs with CUBIC are at or very close to zero, while Copa is still delivering about 6.3 Mbps of application data. The excellent results for Copa are not surprising, as this is a delay-based algorithm and so is relatively insensitive to packet losses, whereas CUBIC being a loss-based algorithm interprets losses as congestion and so reduces its transmit rates accordingly

Converting the above results into a percentage of the theoretical maximum goodput is shown below. Note how long the Copa results are above 90% of the theoretical maximum goodput, while the CUBIC results drop off very quickly.

PER	Linux TCP CUBIC	SLIQ CUBIC	SLIQ Copa
0.0	98.2%	99.6%	99.4%
0.003	33.7%	53.8%	99.4%
0.01	17.1%	26.9%	99.3%
0.03	10.0%	13.2%	99.2%
0.1	4.1%	5.8%	98.6%
0.3	0.0%	1.6%	95.1%
0.4	0.0%	1.2%	94.1%
0.5	0.0%	0.5%	85.1%

These results are shown in the following PER vs. application goodput plot. Note the wide performance difference between the CUBIC results and the Copa results. The results for Copa are very close to the theoretical maximum result.

3.1.5 Link Speed Response

These tests measure how SLIQ responds to different link speeds with latency held constant. Each test was run with a constant link speed (selected from the list of speeds tested), link latency, and switch input buffer size. Ideally, the protocol should be able to fill the available channel in each of these tests.

The following tests were run with one flow originating at the client host on iron95 and terminating at the server host on iron94. Each test was run for a single, constant link speed and was run for 60 seconds to make sure that the results reflect the protocol's steady state behavior. The *LinkEm* software, running on iron97, was configured with the following parameters:

- 1 ms one-way delay (2 ms RTT)
- Maximum conforming buffer size set to two times the maximum bandwidth-delay product for each test
- PER was set to 0.0
- The throttling was set to the following rates: 0.1 Mbps, 0.3 Mbps, 1 Mbps, 3 Mbps, 10 Mbps, 30 Mbps, 100 Mbps, 300 Mbps

The *LinkEm* buffer sizes used in this test, as well as the other tests below, are shown in the table below. Note that if the calculated buffer size was less than 20,000 bytes, then 20,000 bytes was used instead. This is because Copa needs a bottleneck switch buffer that increases the one-way delay when the bottleneck link is full, with room for it to occasionally use additional buffer space beyond the 2,000 bytes that it attempts to keep buffered.

Two Times Bandwidth-Delay Product in Bytes							
Rate in Mbps		One-Way Delay in Milliseconds					
		1	3	10	30	100	300
	0.1	20,000	20,000	20,000	20,000	20,000	20,000
	0.3	20,000	20,000	20,000	20,000	20,000	45,000
	1	20,000	20,000	20,000	20,000	50,000	150,000
	3	20,000	20,000	20,000	45,000	150,000	450,000
	10	20,000	20,000	50,000	150,000	500,000	1,500,000
	30	20,000	45,000	150,000	450,000	1,500,000	4,500,000
	100	50,000	150,000	500,000	1,500,000	5,000,000	15,000,000
	300	150,000	450,000	1,500,000	4,500,000	15,000,000	45,000,000

The following table contains the measured SLIQ application goodput, in megabits per second (Mbps), with the *LinkEm* one-way delay set to 1 ms. The *LinkEm* buffer size and throttling rate were set as listed in the table. The Linux kernel using TCP CUBIC was also tested using the *iperf* application for comparison purposes. Ideally, the application goodput should be approximately 95% of the throttling rate.

LinkEm Throttling (Mbps)	LinkEm Buffering (Bytes)	Linux TCP CUBIC Goodput (Mbps)	SLIQ CUBIC Goodput (Mbps)	SLIQ Copa Goodput (Mbps)
0.1	20,000	0.093	0.094	0.094
0.3	20,000	0.28	0.28	0.28
1	20,000	0.94	0.94	0.94
3	20,000	2.8	2.8	2.8
10	20,000	9.1	9.4	9.4
30	20,000	27.9	28.3	28.1
100	50,000	92	94	94
300	150,000	266	272	271

The SLIQ CUBIC performance matches that of the Linux kernel's TCP CUBIC very closely, with SLIQ CUBIC being a little bit faster at high throttling rates. The SLIQ Copa results are very close to the SLIQ CUBIC results. The overall performance of SLIQ in this test is very good.

3.1.6 Dynamic Link Speed Response

In order to determine how these congestion control algorithms behave when the link speed is changed on the fly, another test was run for each of the algorithms. In each of these tests, the link speeds were changed while the traffic was being transferred.

Again, each test involved one flow originating at the client host on iron95 and terminating at the server host on iron94. The *LinkEm* software, running on iron97, was configured with the following parameters:

- 1 ms one-way delay (2 ms RTT)
- Maximum conforming buffer size set to 50,000 bytes
- PER was set to 0.0
- The throttling was set to the following rates on 10 second boundaries: 0.1 Mbps, 1 Mbps, 10 Mbps, 100 Mbps, 10 Mbps, 1 Mbps, 0.1 Mbps

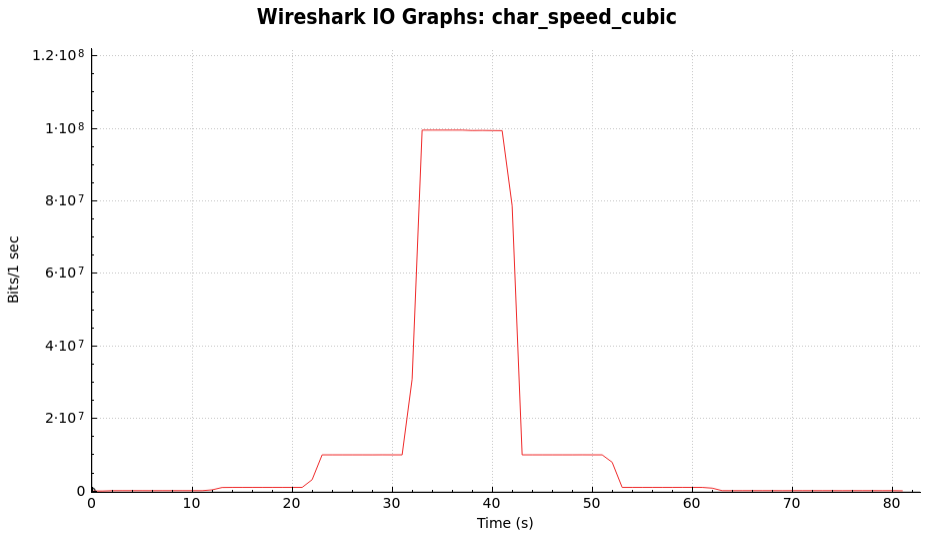
The 300 Mbps rate was not included as the hosts had trouble running the SLIQ test application while doing packet captures at this high of a link rate.

The results are displayed as a wireshark plot of the packets received by the server host, and as a the total number of data packets successfully sent to the server host. Note that the rate changes occur at approximately the following times in the wireshark plots:

- 0.1 Mbps to 1 Mbps at 11 seconds
- 1 Mbps to 10 Mbps at 21 seconds
- 10 Mbps to 100 Mbps at 31 seconds
- 100 Mbps to 10 Mbps at 41 seconds
- 10 Mbps to 1 Mbps at 51 seconds
- 1 Mbps to 0.1 Mbps at 61 seconds

3.1.6.1 SLIQ CUBIC

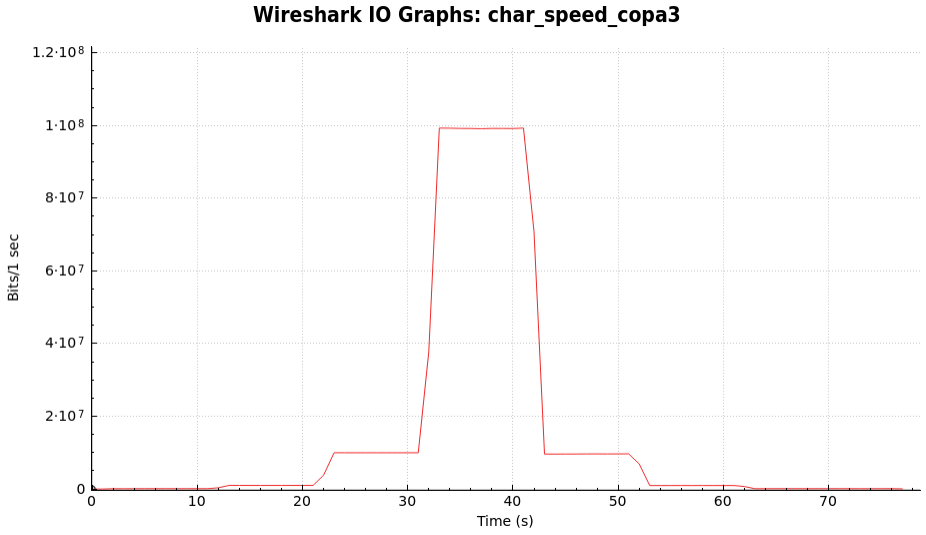
The SLIQ CUBIC results are shown below.



The CUBIC algorithm is able to fill the link at each of the different link speeds, including the 0.1 and 1 Mbps periods which are hard to see in the plot. The total number of data packets delivered to the server host was 143,822 packets.

3.1.6.2 SLIQ Copa

The SLIQ Copa results are shown below.



The Copa algorithm is able to fill the link at each of the different link speeds, including the 0.1 and 1 Mbps periods which are hard to see in the plot. During the 100 Mbps period, Copa appears to be sending slightly slower than CUBIC. The total number of data packets delivered to the server host was 142,925 packets, which is 99.4% of the packets delivered using CUBIC. These results are very close to the CUBIC results.

3.1.7 Link Delay Response

These tests measure how SLIQ responds to different link latencies with speed held constant. Each test was run with a constant link speed, link latency (selected from the list of latencies tested), and switch input buffer size. Ideally, the protocol should be able to fill the available channel in each of these tests.

The following tests were run with one flow originating at the client host on iron95 and terminating at the server host on iron94. Each test was run for 60 seconds to make sure that the results reflect the protocol's steady state behavior. The *LinkEm* software, running on iron97, was configured with the following parameters:

- The one-way delay was set to the following values: 1 ms, 3 ms, 10 ms, 30 ms, 100 ms, 300 ms
- Maximum conform buffer size set to two times the maximum bandwidth-delay product for each test
- PER was set to 0.0
- 10 Mbps throttling

The buffer sizes used in the tests are shown in the "Two Times Bandwidth-Delay Product in Bytes" table above. Note that if the calculated buffer size was less than 20,000 bytes, then 20,000 bytes was used instead. This is because Copa needs a bottleneck switch buffer that increases the one-way delay when the bottleneck link is full, with room for it to occasionally use additional buffer space beyond the 2,000 bytes that it attempts to keep buffered.

The following table contains the measured SLIQ application goodput, in megabits per second (Mbps), with the *LinkEm* throttling set to 10 Mbps. The *LinkEm* buffer size and one-way delay were set as listed in the table. The Linux kernel using TCP CUBIC was also tested using the *iperf* application for comparison purposes. Ideally, the application goodput should be approximately 95% of the throttling rate, or 9.5 Mbps for each test.

LinkEm One-Way Delay (ms)	LinkEm Buffering (Bytes)	Linux TCP CUBIC Goodput (Mbps)	SLIQ CUBIC Goodput (Mbps)	SLIQ Copa Goodput (Mbps)
1	20,000	9.06	9.43	9.43
3	20,000	9.37	9.43	9.36
10	50,000	9.36	9.43	9.42
30	150,000	9.35	9.39	9.43
100	500,000	9.26	9.25	9.43
300	1,500,000	8.71	7.32	9.43

The SLIQ CUBIC performance matches that of the Linux kernel's TCP CUBIC protocol for delays up to 100 ms, then falls off somewhat. The SLIQ Copa performance is outstanding, maintaining nearly maximum goodput in each test.

3.1.8 Dynamic Link Delay Response

In order to determine how these congestion control algorithms behave when the link delay is changed on the fly, another test was run for each of the algorithms. In each of these tests, the link delays were changed while the traffic was being transferred.

Again, each test involved one flow originating at the client host on iron95 and terminating at the server host on iron94. The *LinkEm* software, running on iron97, was configured with the following parameters:

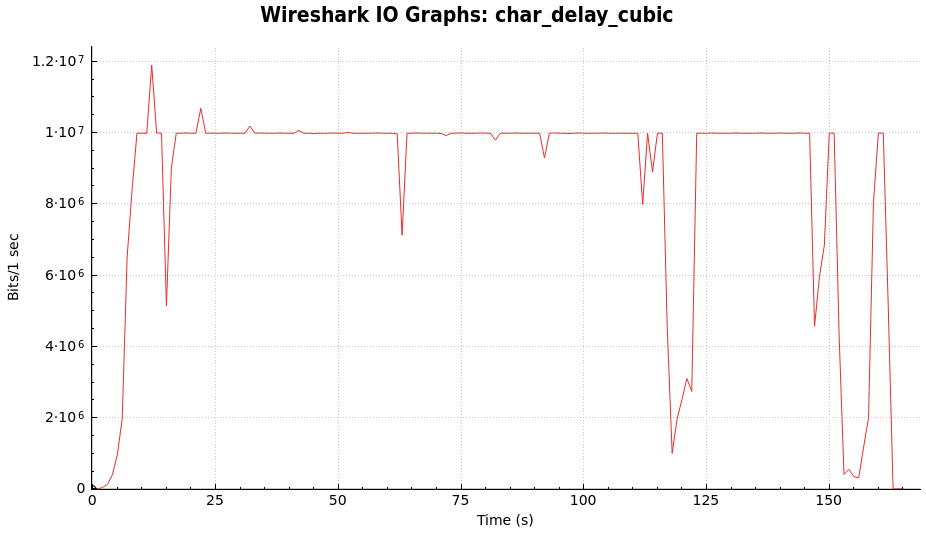
- The one-way delay was set to the following values for the specified durations: 300 ms for 10 seconds, 100 ms for 10 seconds, 30 ms for 10 seconds, 10 ms for 10 seconds, 3 ms for 10 seconds, 1 ms for 10 seconds, 3 ms for 10 seconds, 10 ms for 10 seconds, 30 ms for 10 seconds, 100 ms for 20 seconds, 300 ms for 50 seconds
- Maximum conforming buffer size set to 1,500,000 bytes
- PER was set to 0.0
- 10 Mbps throttling

The results are displayed as a wireshark plot of the packets received by the server host, and as a the total number of data packets successfully sent to the server host. Note that the delay changes occur at approximately the following times in the wireshark plots:

- 300 ms to 100 ms at 12 seconds
- 100 ms to 30 ms at 22 seconds
- 30 ms to 10 ms at 32 seconds
- 10 ms to 3 ms at 42 seconds
- 3 ms to 1 ms at 52 seconds
- 1 ms to 3 ms at 62 seconds
- 3 ms to 10 ms at 72 seconds
- 10 ms to 30 ms at 82 seconds
- 30 ms to 100 ms at 92 seconds
- 100 ms to 300 ms at 112 seconds

3.1.8.1 SLIQ CUBIC

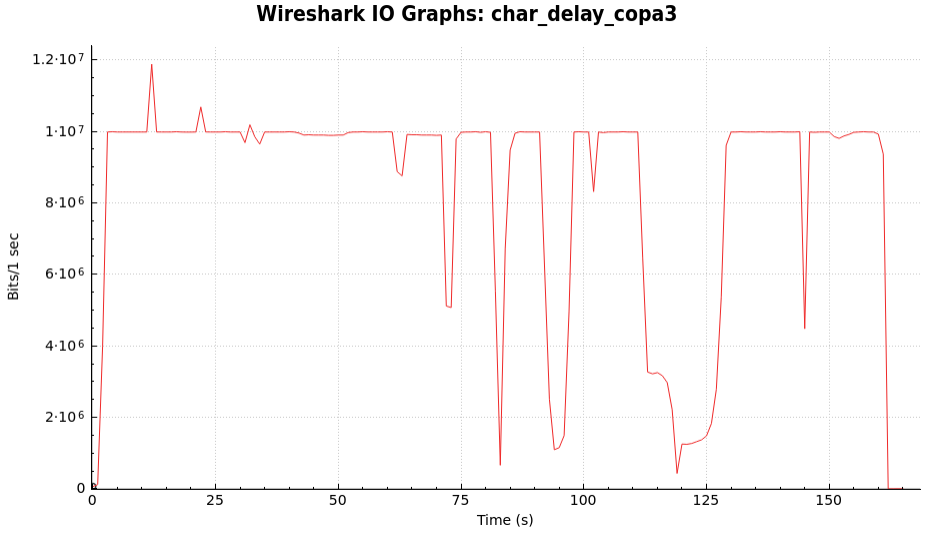
The SLIQ CUBIC results are shown below.



The CUBIC algorithm starts off slowly due to the extremely large 600 ms RTT. During most of the middle transients, CUBIC is able to keep the link full, although there are some minor spikes up and down at the transition times. In the last 300 ms link delay period, it has significant trouble keeping the link full. The total number of data packets delivered to the server host was 167,644 packets. Overall, this is a fairly good result.

3.1.8.2 SLIQ Copa

The SLIQ Copa results are shown below.



The Copa algorithm starts off very quickly, reaching 10 Mbps within the first couple of seconds. It is then able to keep the link full through all of the link delay decreases. However, there is a noticeable drop in send rate during each of the link delay increases which is due to the length of the time window used in detecting link delay increases. This is most pronounced during the 100 ms to 300 ms transition, which takes Copa approximately 20 seconds to begin sending at the proper rate again.

The total number of data packets delivered to the server host was 163,431 packets, which is 97.5% of the packets delivered in the CUBIC test. This result is very comparable to the CUBIC result.

3.1.9 Capacity Estimation

This test measures how well the protocol estimates the current channel capacity, which is needed by IRON for several uses, such as sending QLAM packets at an appropriate rate. Note that the capacity estimate is not the current send rate for the flow, but the amount of capacity currently available in the channel for the flow. With a single SLIQ flow sending packets as fast as possible, the protocol should be able to closely estimate the channel capacity as set by *LinkEm*.

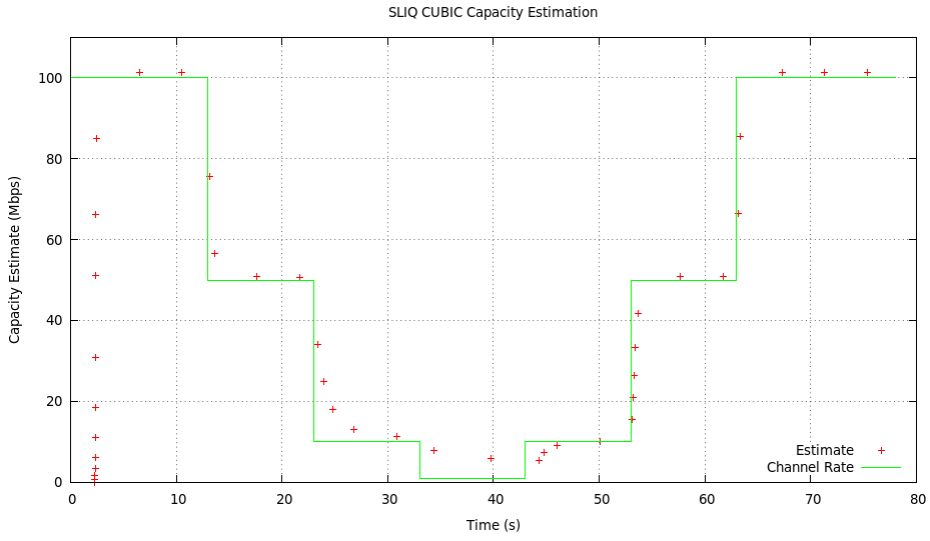
The following tests were run with one flow originating at the client host on iron95 and terminating at the server host on iron94. The *LinkEm* software, running on iron97, was configured with the following parameters:

- 5 ms one-way delay (10 ms RTT)
- 250,000 byte maximum conform buffer size (which is two times the maximum bandwidth-delay product)
- PER was set to 0.0
- The throttling was set to the following rates on 10 second intervals: 100 Mbps, 50 Mbps, 10 Mbps, 1 Mbps, 10 Mbps, 50 Mbps, 100 Mbps

During this test, the *LinkEm* channel rates and SLIQ capacity estimates were logged, and were used to generate the following plots. The green line in the plots is the *LinkEm* channel capacity in Mbps, while the red crosses are the SLIQ capacity estimates in Mbps. Since the SLIQ capacity estimates (red crosses) take into account the packet overhead due to headers and trailers, the channel capacity (green line) can be directly compared with the capacity estimates (red crosses). Note that in each of the following plots, the horizontal position of the channel capacity (green line) is only accurate to the previous whole second, so the actual rate change position might be shifted by up to one second to the right of the green line transients.

3.1.9.1 SLIQ CUBIC

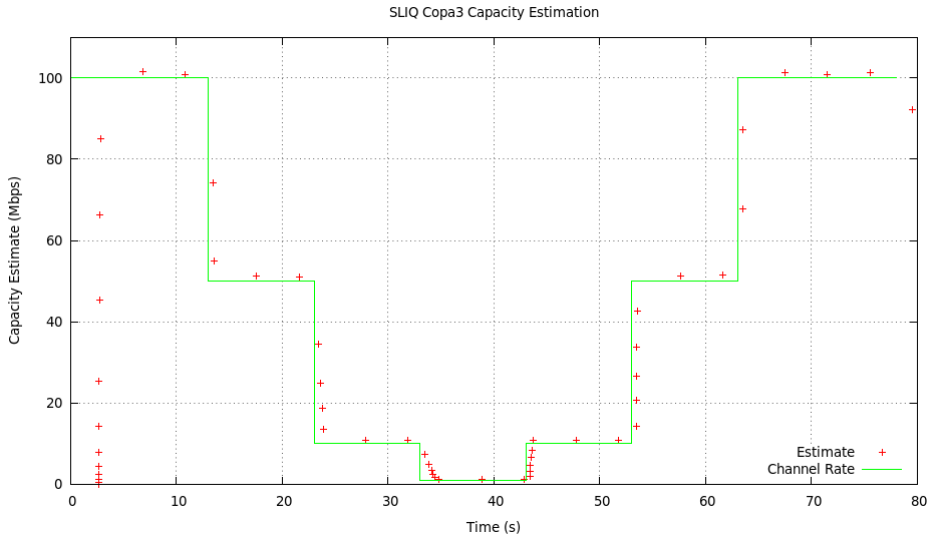
The SLIQ CUBIC capacity estimate results are plotted below. Since the CUBIC algorithm includes a congestion window, the capacity estimates are derived from the current congestion window size and the smoothed RTT estimate.



The capacity estimate points closely follow the channel capacity line through all of the different channel rates except for the 1 Mbps period. During the 1 Mbps period, the estimates only drop to 5.4 Mbps, which is not very close to the actual 1 Mbps rate. The capacity estimates react faster when the channel capacity is higher, which makes sense as there is more frequent feedback to adjust the estimate when the send rate is high. There are no capacity estimate overshooting in the CUBIC results.

3.1.9.2 SLIQ Copa

The SLIQ Copa capacity estimate results are plotted below. Since the Copa algorithm includes a congestion window, the capacity estimates are derived from the current congestion window size and the smoothed RTT estimate.



The capacity estimate points closely follow the channel capacity line through all of the different channel rates. The capacity estimates for Copa react at least as fast, but sometimes faster, than those for CUBIC, regardless of the channel capacity. During the transition from 10 Mbps to 1 Mbps, there is only a slight delay before the capacity estimate starts to drop quickly toward 1 Mbps. There are no capacity estimate overshooting in the Copa results. Overall, the Copa results are better than the CUBIC results for this test.

3.2 SLIQ Congestion Control Algorithm Characterization (Dual Algorithm Configurations. i.e., Nuisances)

This section provides the results of a set of characterization tests performed on SLIQ when using two congestion control algorithms concurrently. The two congestion control algorithms used in these tests are the SLIQ versions of CUBIC and Copa. These results may be compared with previous performance results of SLIQ using a single congestion control algorithm found above.

Keep in mind that the goal of combining two congestion control algorithms in parallel is to take advantage of the capabilities of one algorithm when the other algorithm is unable to perform well. It is not important that the two algorithms always handle one half of the network traffic, only that the two algorithms complement each other in terms of their strengths and weaknesses. Thus, some of the following tests will show one algorithm handling a large portion of the network traffic, and this condition is acceptable so long at the overall results of the test are acceptable.

All of the following test results were produced using the SLIQ *test_sliq* program compiled in optimized mode. During each of the one-way tests, a single stream of data was sent from client to server. Each test was configured to send packets with 1000 byte payloads.

For the maximum performance tests, the BBN IRON Testbed 8X was used. The server-side program was run on iron80 (dual-core Intel Xeon 1.87 GHz processor with 4 Gbytes of main memory), and the client-side program was run on iron83 (dual-core Intel Xeon 1.87 GHz processor with 4 Gbytes of main memory). The IRON link emulation program *LinkEm* was not used during the performance tests. The link between iron80 and iron83 is Gigabit Ethernet, which limits the results to 1 Gbps minus the rate required for the packet overhead (1 Gbps minus ~70 Mbps, which equals ~930 Mbps).

For all other tests, the BBN IRON Testbed 9X was used. The server-side program was run on iron94 (dual-core Intel Xeon 1.87 GHz processor with 8 Gbytes of main memory), and the client-side program was run on iron95 (dual-core Intel Xeon 1.87 GHz processor with 6 Gbytes of main memory). The IRON link emulation program *LinkEm* was run on iron97 to control the link characteristics (delay, throttling, packet error rate (PER), and bottleneck switch queue size) between iron94 and iron95.

In each of the wireshark plots that follow, note that the x-axis resolution is changed from 1 second to 0.1 seconds in wireshark in order to show additional detail. This forces the y-axis to be in units of "Bits/100ms". Thus, 10 Mbps appears as "$1 \cdot 10^6$ Bits/100 ms" in the plots.

3.2.1 Maximum Performance Results

This test measures the maximum goodput (application data throughput) of both a single stream of data sent from the client to the server and a bi-directional stream of data sent both ways. It is a test of the efficiency of the protocol software implementation as well as a test of the ability of the congestion control algorithm to stabilize at a high data rate. Data was sent from iron83 to iron80 for the one-way tests.

As mentioned above, the use of a Gigabit Ethernet link and the SLIQ packet overhead limits the results to ~930 Mbps. Each individual test was configured to run for 60 seconds in order to minimize the effect of startup transients. The listed results are computed using the total number of application bytes received and the amount of time elapsed from the receipt of the first byte until the last byte. For the one-way tests, five consecutive runs were performed for each congestion control algorithm, and the five results were averaged to generate the listed values. For the two-way tests, the two goodput values are shown along with the average of the two values.

The test results are shown in the table below.

Congestion Control Algorithm(s)	Maximum One-Way Goodput	Maximum Two-Way Goodput (Mbps)
CUBIC	472 Mbps	270/340 Mbps, 305 Mbps Average
Copa	458 Mbps	280/338 Mbps, 309 Mbps Average
CUBIC, Copa	446 Mbps	280/321 Mbps, 301 Mbps Average
Copa, CUBIC	441 Mbps	285/304 Mbps, 295 Mbps Average

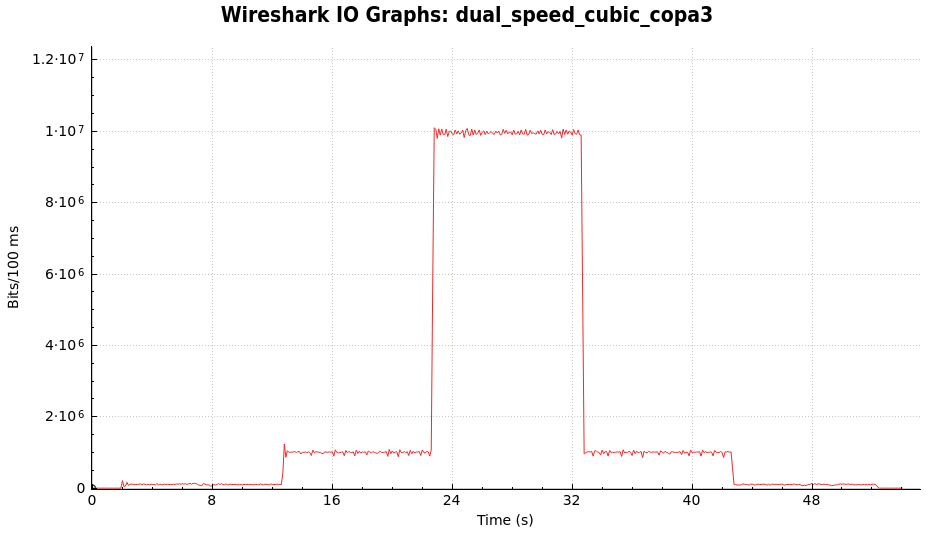
It is interesting to note that with both congestion control algorithms in use and CUBIC specified first (which means that SLIQ will always try to use CUBIC before Copa), CUBIC ends up sending almost all of the packets in both the one-way and two-way tests. If the algorithm order is reversed, the goodput results are slightly lower due to Copa sending more of the packets. This is due to the amount of jitter in these small RTT measurements at these high speeds, which is treated as a sign of congestion by Copa. Both the one-way and two-way results for "CUBIC,Copa" are better than those for "Copa,CUBIC", so this order of congestion control algorithms is used throughout the rest of the tests.

3.2.2 Dynamic Link Speed Response

In order to determine how the congestion control algorithms operating in parallel behave when the link speed is changed on the fly, another test was run in which the link speed was changed while the traffic was being transferred. The test involved data being sent from the client on iron95 to the server on iron94. The *LinkEm* software running on iron97 was configured with the following parameters:

- 1 ms one-way delay (2 ms RTT)
- Maximum conforming buffer size set to 50,000 bytes
- PER set to 0.0
- The throttling was set to the following rates on 10 second boundaries: 1 Mbps, 10 Mbps, 100 Mbps, 10 Mbps, 1 Mbps

The wireshark plot of the SLIQ traffic sent by the client while running with congestion control set to "CUBIC,Copa" is shown below. Individually, the two congestion control algorithms are able to adapt to changes in the link speed. Once combined, this result shows that they are able to quickly and accurately track the link speed as it changes without significant instabilities.

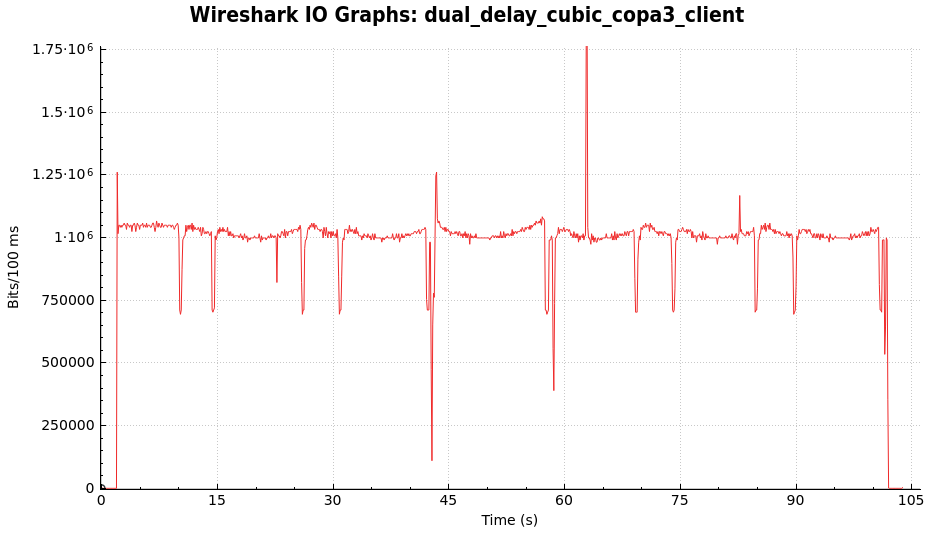


3.2.3 Dynamic Link Delay Response

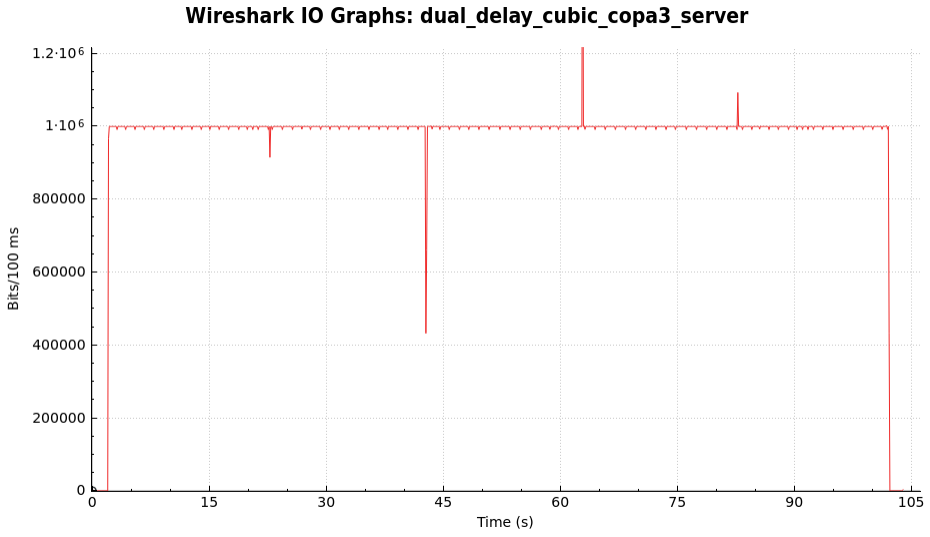
In order to determine how the congestion control algorithms operating in parallel behave when the link delay is changed on the fly, another test was run in which the link delay was changed while the traffic was being transferred. A change in link delay emulates a route change between the two endpoints. The test involved data being sent from the client on iron95 to the server on iron94. The *LinkEm* software running on iron97 was configured with the following parameters:

- 10 Mbps throttling
- Maximum conforming buffer size set to 500,000 bytes
- PER set to 0.0
- The one-way delay was set to the following values on 20 second boundaries: 1 ms, 10 ms, 100 ms, 10 ms, 1 ms

The wireshark plot of the SLIQ traffic sent by the client while running with congestion control set to "CUBIC,Copa" is shown below. Individually, CUBIC is able to adapt to changes in link delay faster than Copa. Once combined, the results show that CUBIC is able to take over from Copa when needed. The send rate is not completely stable during the link delay changes, but the variations are acceptable.



In order to determine if the sender is completely filling the available network capacity during the link delay changes, the wireshark plot of the SLIQ traffic received by the server during the test is shown below. Notice how the receive rate stays at 10 Mbps with very little variation. The plot shows that the client is successful in utilizing all of the network capacity during the link delay changes.



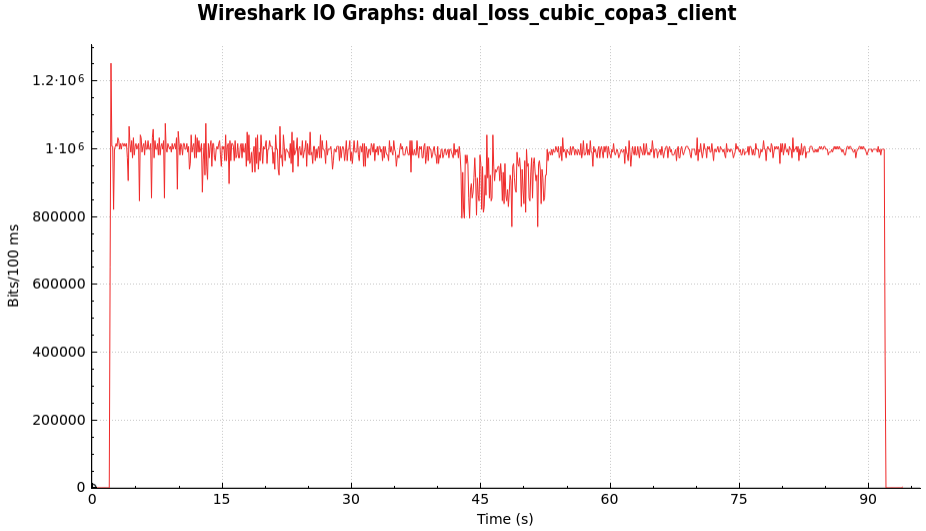
3.2.4 Dynamic Link Packet Loss Response

In order to determine how the congestion control algorithms operating in parallel behave when the link suffers non-congestion packet losses, another test was run in which the link packet error rate (PER) was changed while the traffic was being transferred. The test involved data being sent from the client on iron95 to the server on iron94. The *LinkEm* software running on iron97 was configured with the following parameters:

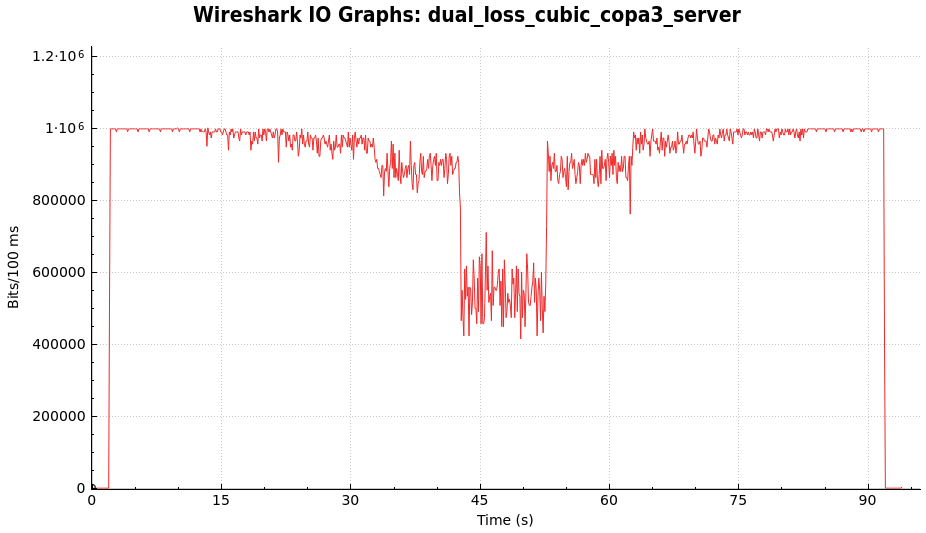
- 10 Mbps throttling
- 10 ms one-way delay (20 ms RTT)
- Maximum conforming buffer size set to 50,000 bytes

- The PER was set to the following values on 10 second boundaries: 0.0, 0.01, 0.03, 0.1, 0.4, 0.1, 0.03, 0.01, 0.0

The wireshark plot of the SLIQ traffic sent by the client while running with congestion control set to "CUBIC,Copa" is shown below. Individually, Copa is able to continue to operate when non-congestion packet losses occur while CUBIC has trouble. Once combined, the results show that Copa is able to take over from CUBIC when needed. When the PER is set to 0.4, even Copa starts having some trouble sending at the correct rate due to the extremely large percentage of packets that are lost. While the send rate is not completely stable during the PER=0.4 period, the results are still acceptable.



In order to determine if the sender is filling the available network capacity during the packet loss events, the wireshark plot of the SLIQ traffic received by the server during the test is shown below. Notice how the maximum possible receive rate is equal to the amount of network traffic that is not lost by the link, which is 10 Mbps times (1 - PER): 10 Mbps, 9.9 Mbps, 9.7 Mbps, 9 Mbps, 6 Mbps, 9 Mbps, 9.7 Mbps, 9.9 Mbps, and 10 Mbps. The plot shows that the client is fairly good at utilizing all of the network capacity during the link loss periods, and is very successful in quickly recovering to nearly the maximum rate possible when the loss periods are over.



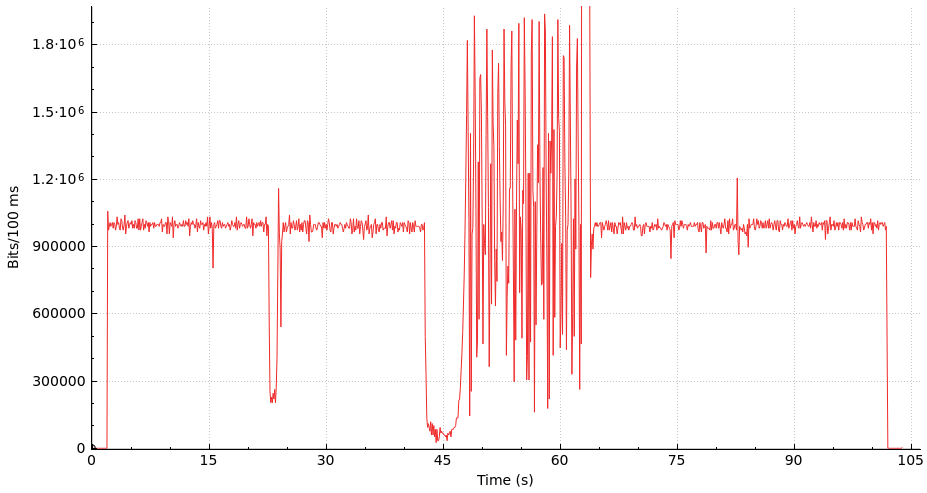
3.2.5 Dynamic Link Delay and Packet Loss Response

In order to determine how the congestion control algorithms operating in parallel behave when the link suffers both link delay changes and non-congestion packet losses at the same time, another test was run in which the link packet error rate (PER) was set to a non-zero value and the link delay was changed while the traffic was being transferred. This is a much more extreme test than either of the last two tests as it stresses both congestion control algorithms in some of their weakest areas at the same time. The test involved data being sent from the client on iron95 to the server on iron94. The *LinkEm* software running on iron97 was configured with the following parameters:

- 10 Mbps throttling
- Maximum conforming buffer size set to 50,000 bytes
- The PER set to 0.1
- The one-way delay was set to the following values on 20 second boundaries: 1 ms, 10 ms, 100 ms, 10 ms, 1 ms

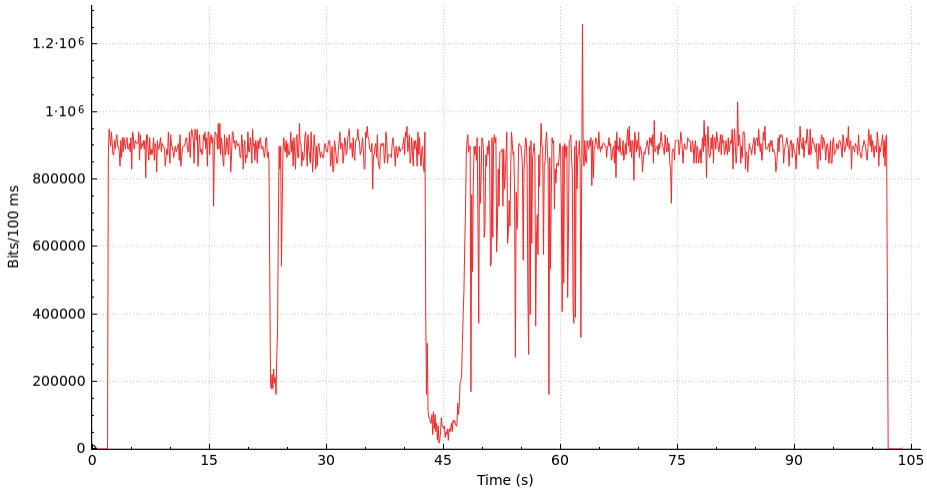
The wireshark plot of the SLIQ traffic sent by the client while running with congestion control set to "CUBIC,Copa" is shown below. Because the PER is set to 0.1 throughout this test, CUBIC ends up sending very little traffic. The bulk of the sending then falls to Copa, which can handle the random packet loss. As the link delay goes from 1 ms to 10 ms at time 22 seconds, the send rate dips down momentarily to about 2 Mbps while Copa adapts its minimum RTT estimate upward. As the link delay goes from 10 ms to 100 ms at time 42 seconds, there is a 6 second period when the send rate is approximately 1 Mbps while Copa again adapts its minimum RTT estimate upward. Since the RTT is larger at the start of the second transition, it takes the algorithm additional time to detect the change. Once adapted to the 100 ms delay, Copa then oscillates its send rate more dramatically due to the large RTT. As the link delay goes from 100 ms to 10 ms at time 62 seconds, and 10 ms to 1 ms at time 82 seconds, Copa is able to adapt its minimum RTT estimate immediately, and there are no big drops in the observed send rate as a result. While there are periods when the send rate is too low and there is a period of large send rate oscillations, the results are still acceptable.

Wireshark IO Graphs: dual_loss_delay_cubic_copa3_client



In order to determine if the sender is filling the available network capacity during the packet loss events, the wireshark plot of the SLIQ traffic received by the server during the test is shown below. Notice how the receive rate is roughly equal to the amount of network traffic that is not lost by the link, which should be 10 Mbps times (1 - PER), or 9 Mbps. Except for the two very low send rate periods when the link delay is increased, this shows that the client is utilizing either all or nearly all of the network capacity throughout the test.

Wireshark IO Graphs: dual_loss_delay_cubic_copa3_server



3.2.6 Network Buffering Response

In order to determine how the congestion control algorithms operating in parallel behave when there are varying levels of network buffering, another set of tests were run, each with a different *LinkEm* buffer size. The tests involved data being sent from the client on iron95 to the server on iron94 while running with congestion control set to "CUBIC,Copa". The *LinkEm* software running on iron97 was configured with the following parameters:

- 10 Mbps throttling
- 10 ms one-way delay (20 ms RTT)
- PER set to 0.0

The LinkEm maximum conforming buffer size was set to varying multiples of the link's bandwidth-delay product (BDP), which in this case is (10,000,000 * 0.020 / 8) = 25,000 bytes. Typically, the optimal amount of buffering for a network link is 2-4 times the BDP, but it is possible for network links to be either under-buffered or over-buffered in practice.

The test results are shown in the table below.

LinkEm Buffer Size	Goodput	CUBIC Traffic	Copa Traffic
6,250 Bytes (0.25 x BDP)	9.34 Mbps	4.2%	95.8%
12,500 Bytes (0.5 x BDP)	9.40 Mbps	65.2%	34.8%
25,000 Bytes (1 x BDP)	9.43 Mbps	78.1%	21.9%
50,000 Bytes (2 x BDP)	9.43 Mbps	82.0%	18.0%
100,000 Bytes (4 x BDP)	9.43 Mbps	88.9%	11.1%
200,000 Bytes (8 x BDP)	9.43 Mbps	94.0%	6.0%
400,000 Bytes (16 x BDP)	9.43 Mbps	96.6%	3.4%

The goodput results from the tests show that the link is being completely used regardless of the network buffering size. Only the 0.25 and 0.5 times BDP results are just slightly lower than the maximum goodput rate of 9.43 Mbps. Having this high of a result for both severely under-buffered and over-buffered conditions is exceptional.

Looking at the percentage of the traffic controlled by CUBIC and Copa, the results are as expected. CUBIC attempts to fill all of the network buffer space available, while Copa attempts to utilize only 2 packets (2,000 Bytes) worth of the network buffer space. In practice, Copa's send rate is designed to oscillate, which causes the amount of network buffer space that it uses to also oscillate. Thus, the actual average network buffer space used by Copa is usually slightly larger than 2 packets, and can vary depending on the exact send rate and RTT values. Given that, at steady state, the percentage of network buffer space used by one of the congestion control algorithms is equal to the percentage of packets being sent by that algorithm, the trend should be that Copa sends most of the packets when the network buffer space is small, while CUBIC sends most of the packets when the network buffer space is large. This trend is clearly evident in the results above. The first three rows in the table correspond to Copa using approximately 5 packets of network buffer space. The other rows correspond to Copa using more than 5 packets worth of buffer space, but this occurs because Copa has a very small congestion control window during these tests, and granularity issues when updating the congestion window causes a slightly more aggressive behavior.

3.2.7 Network Outage Response

In order to make sure that the congestion control algorithms operating in parallel are able to recover properly when there is a network outage event, another set of tests were run with *LinkEm* simulating an outage by setting the PER to 1.0 at the start of the outage and returning PER to 0.0 after the outage is over. Both complete outages and one-way outages are tested, as well as short (one second) and long (10 seconds) outages. In these tests, data is sent in both directions between the client on iron95 and the server on iron94, since this is the normal operating mode for CATs. The congestion control is set to "CUBIC,Copa" in each test. The LinkEm software running on iron97 was configured with the following parameters:

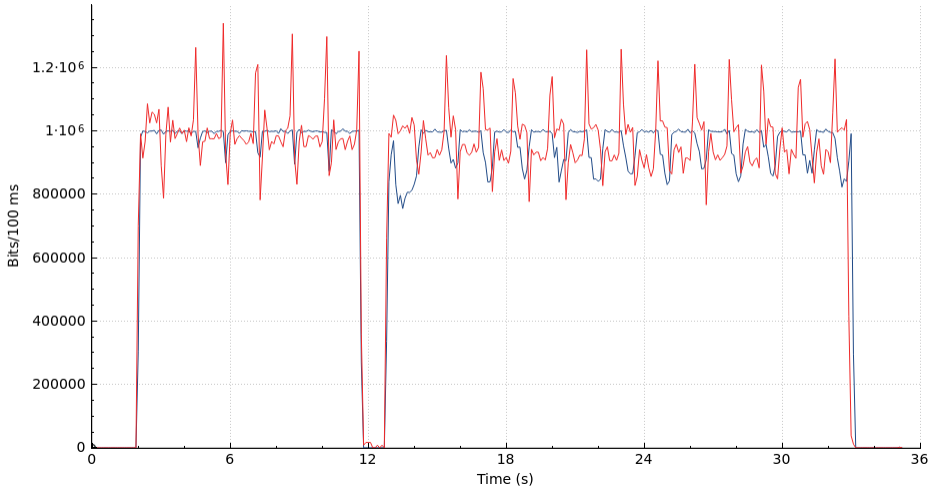
- 10 Mbps throttling
- 10 ms one-way delay (20 ms RTT)
- Maximum conforming buffer size set to 50,000 bytes
- PER set to 0.0 and 1.0 as necessary

The results of these tests are wireshark plots of the client and the server. In these plots, traffic sent by the host is plotted in red, and traffic received by the host is plotted in dark blue. Note that the x-axis resolution of these plots is changed from 1 second to 0.1 seconds in wireshark in order to show additional detail, and this forces the y-axis to be in units of "Bits/100ms". Thus, 10 Mbps appears as "$1 \cdot 10^6$ Bits/100 ms" in the plots.

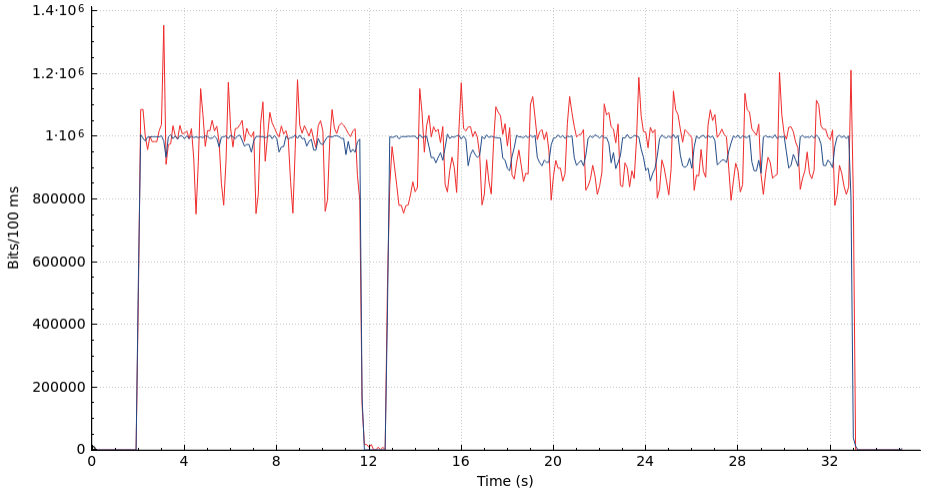
3.2.7.1 Short Two-Way Outage

The client and server plots show that both ends stop receiving all traffic for about 1 second, where the blue lines go to zero. Both ends slow down their send rates rapidly to near zero, where the red lines go toward zero. Once the outage is over, both ends quickly rebound to their normal throughput.

Wireshark IO Graphs: dual_short_twoway_cubic_copa3_client



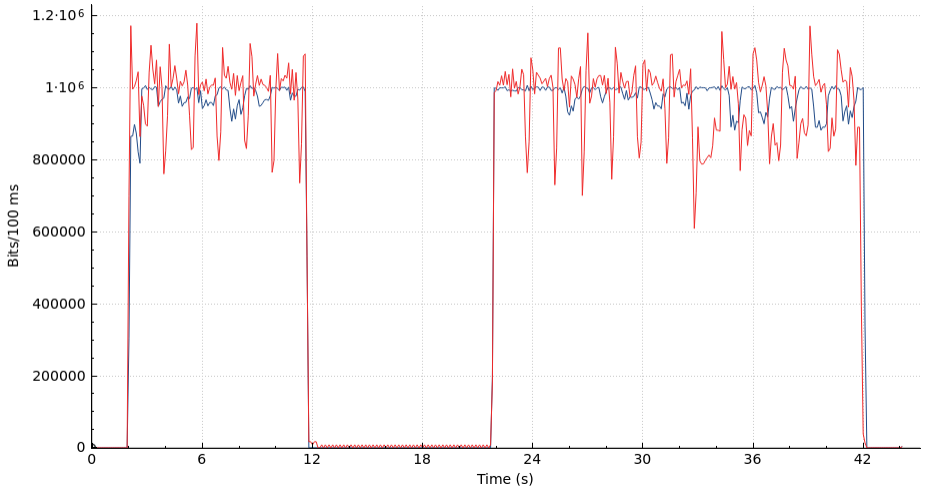
Wireshark IO Graphs: dual_short_twoway_cubic_copa3_server



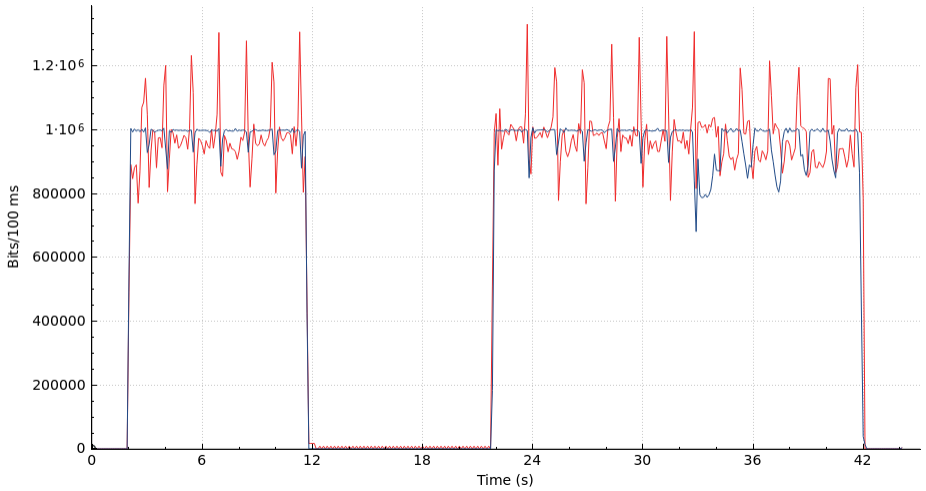
3.2.8 Long Two-Way Outage

The client and server plots show that both ends stop receiving all traffic for about 10 seconds, where the blue lines go to zero. Both ends slow down their send rates rapidly to near zero, where the red lines go toward zero. Once the outage is over, both ends quickly rebound to their normal throughput.

Wireshark IO Graphs: dual_long_twoway_cubic_copa3_client

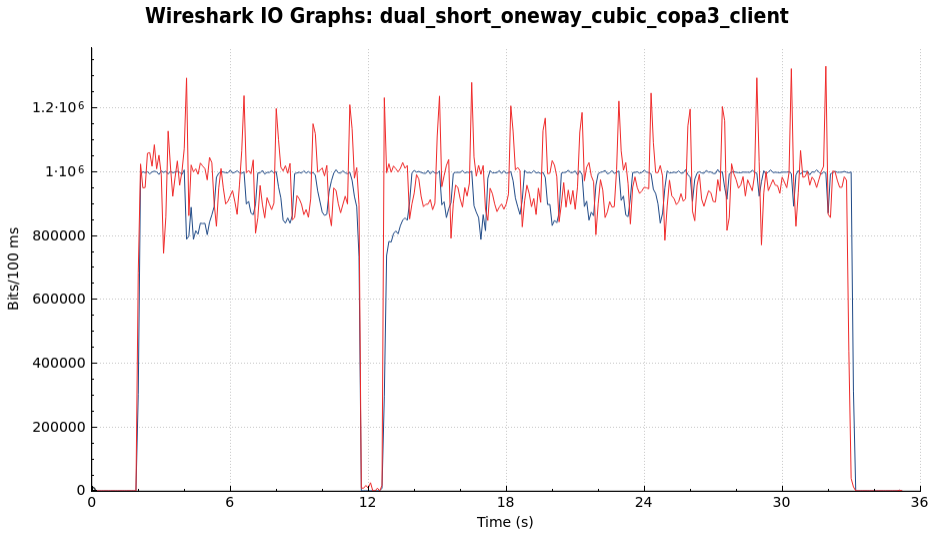


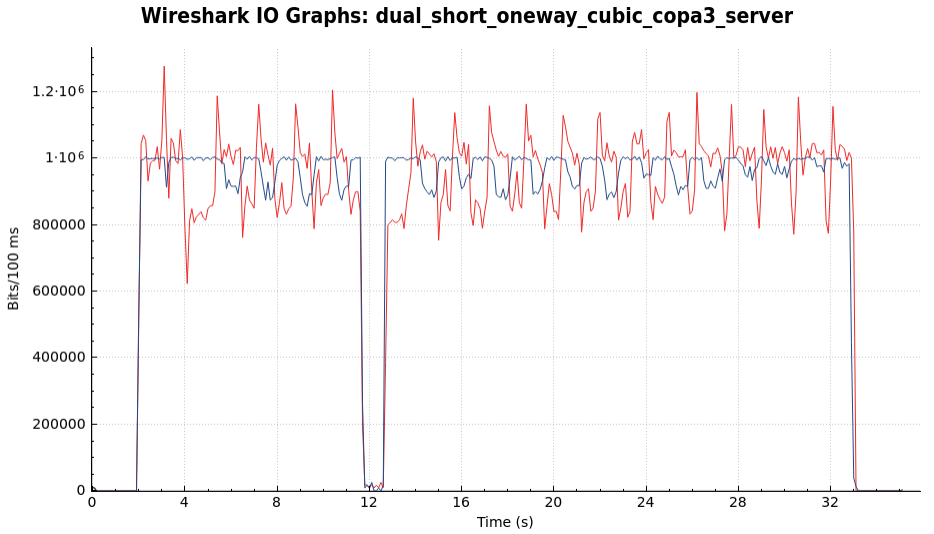
Wireshark IO Graphs: dual_long_twoway_cubic_copa3_server



3.2.9 Short One-Way Outage

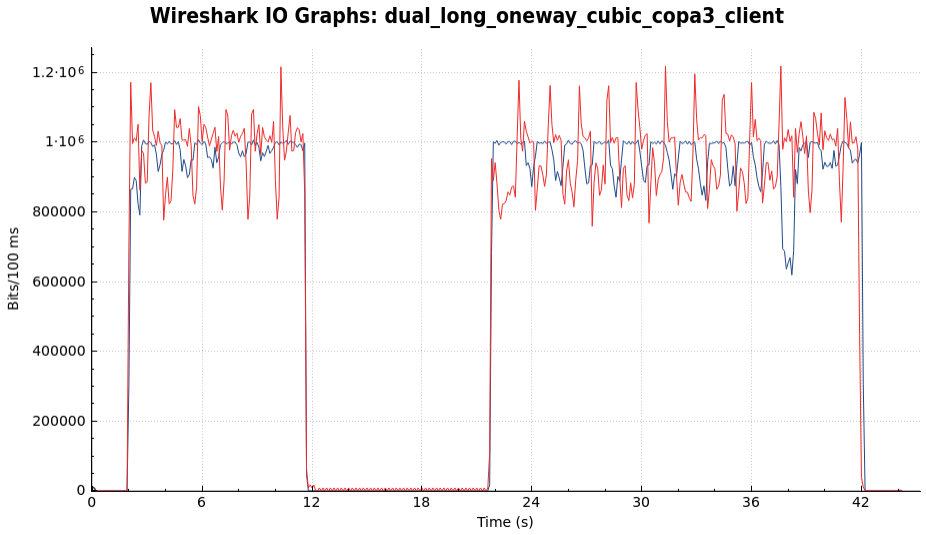
In this test, the one-way outage occurs from the server to the client. The client plot shows that it stops receiving all traffic for about 1 second, where the blue line goes to zero. Since the client stops receiving any packets from the server, it enters SLIQ's outage mode. However, since the server continues to receive some packets from the client, it slows down its sending but does not enter SLIQ's outage mode. Both ends slow down their send rates rapidly, where the red lines go toward zero. Once the outage is over, both ends quickly rebound to their normal throughput, with the server taking slightly longer to reach the full send rate because it was not in outage mode.

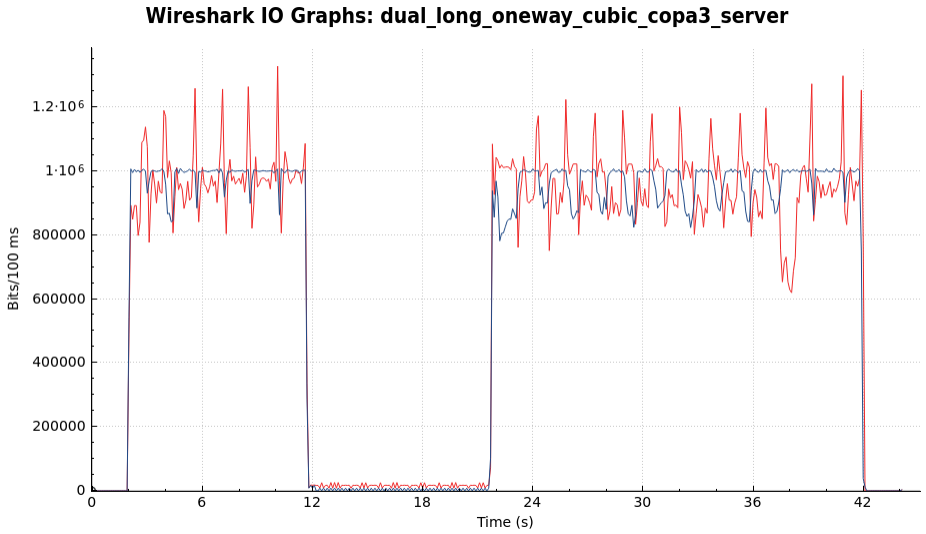




3.2.10 Long One-Way Outage

In this test, the one-way outage occurs from the server to the client. The client plot shows that it stops receiving all traffic for about 10 seconds, where the blue line goes to zero. Since the client stops receiving any packets from the server, it enters SLIQ's outage mode. However, since the server continues to receive some packets from the client, it slows down its sending but does not enter SLIQ's outage mode. Both ends slow down their send rates rapidly, where the red lines go toward zero. Once the outage is over, both ends quickly rebound to their normal throughput, with the server taking slightly longer to reach the full send rate because it was not in outage mode.





3.3 SLIQ Latency-Constrained Adaptive Error Control (AEC) Characterization

This section provides the results of a set of characterization tests performed on the SLIQ (Simple Lightweight IPv4 QUIC) protocol when using the Latency-Constrained Adaptive Error Control (AEC) algorithm. This algorithm is used by a SLIQ stream when configured to use the semi-reliable ARQ+FEC mode, which attempts to deliver a portion of the datagrams within a time limit. This algorithm is adaptive in that it utilizes the current SLIQ measurements of the packet error rate (PER), local-to-remote one-way delay, and round-trip time (RTT) to decide when automatic repeat request (ARQ) and/or forward error correction (FEC) should be used to meet the delivery constraints.

3.3.1 Test Setup

For these tests, the BBN IRON Testbed 8X was used. The server-side program was run on iron84 (dual-core Intel Xeon 1.87 GHz processor with 4 Gbytes of main memory), and the client-side program was run on iron83 (dual-core Intel Xeon 1.87 GHz processor with 4 Gbytes of main memory). The IRON link emulation program *LinkEm* was run on iron86 to control the link characteristics (delay, throttling, packet error rate (PER), and bottleneck switch queue size) between iron83 and iron84.

All of the following test results were produced using the SLIQ *test_sliq* program compiled in optimized mode. During each of the tests, a single stream of data was sent from client to server, and the client application was configured to send 1250 byte packets by specifying the use of 1184 byte packet payloads. This 1250 byte packet size results in 10,000 bit packets, which makes the packet transmission delay math easier for the reader (e.g., sending 1250 bytes packets at 10 Mbps causes a transmission delay of 1 millisecond). The SLIQ connection was configured to use the dual congestion control algorithm configuration "Cubic,Copa". The SLIQ stream was configured to use semi-reliable ARQ+FEC mode, with a target packet delivery probability of 0.995 and a target packet delivery deadline of between 30 milliseconds to 170 milliseconds in 20 millisecond increments. Note that the deadline is the maximum delay allowed from the time the application data is passed to the SLIQ sender until the data is delivered to the receiving application. This means that even though 100% of the application data will be delivered to the receiver, 99.5% of the application data will be delivered within the specified deadline.

All of the channel properties (delay, bit rate, packet loss rate, and queue sizes) are unknown to both the SLIQ sender and receiver during each of the tests. Thus, SLIQ must compute estimates for each of these channel properties in order to use AEC to select the best error control mode to use that will maximize efficiency while meeting the delivery constraints. The three error control modes that AEC can choose from when attempting to meet the delivery constraints are:

- **ARQ Mode**: Also called **Pure ARQ Mode**. Lost data packets are repaired using only retransmissions. No FEC is used. It may take several rounds of retransmissions in order to get a data packet to the receiver in time.
- **Coded ARQ Mode**: Lost data packets are repaired using a combination of FEC and retransmissions. It may take several rounds of retransmissions in order to get a data packet to the receiver in time.
- **FEC Mode**: Also called **Pure FEC Mode**. Lost data packets are repaired using only FEC. Only a single round is used to get a data packet to the receiver in time.

Note that after the delivery constraints can no longer be met for a particular data packet, ARQ is used to still deliver the data packet even though it is late.

3.3.2 AEC Goodput Performance

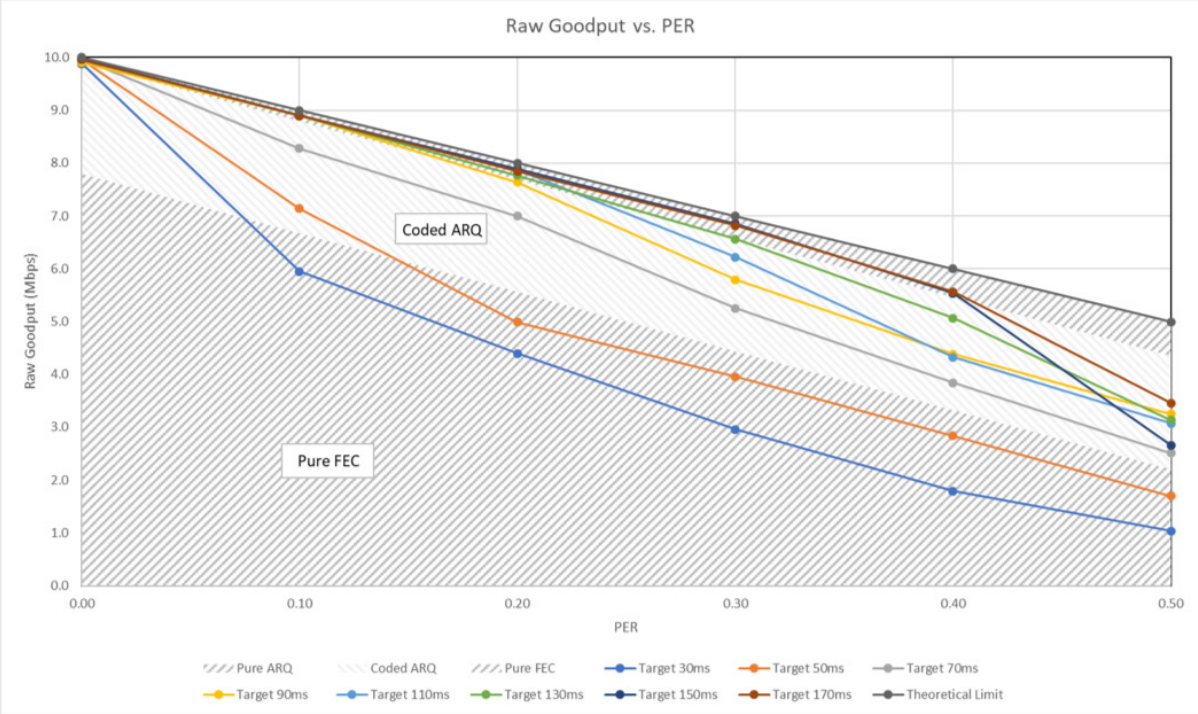
This test measures the raw goodput of SLIQ's AEC algorithm over a range of PER and target packet delivery deadlines. Raw goodput is the rate in which all *usable* data packets are received, including all data packet headers on those packets. A series of individual tests were completed to create a plot of how the raw goodput changes as PER is changed, for a set of target packet delivery deadline values.

The following tests were run with one flow originating at the client host on iron83 and terminating at the server host on iron84. Each test was 120 seconds long to minimize the impact of the SLIQ protocol starting up and calculating the first channel property estimates. The *LinkEm* software, running on iron86, was configured with the following parameters:

- 10 ms one-way delay (20 ms RTT)
- 10 Mbps throttling
- 50,000 byte maximum conform buffer size
- Bi-directional PER set to the following values: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5

The SLIQ stream was configured to use semi-reliable ARQ+FEC mode (which utilizes AEC), with a target packet delivery probability of 0.995 and a target packet delivery deadline of between 30 ms to 170 ms in 20 ms increments.

The results of these tests are shown in the following Raw Goodput vs. PER plot.



Note that all operating points achieved the target packet delivery probability of 0.995 except for the 30 ms target packet delivery deadline test at PER=0.0. In this test, the CUBIC congestion control algorithm dominates since there is no non-congestion packet loss, resulting in the *LinkEm* buffers being full most of the time. Given this situation, the minimum packet delivery delay achievable is 1 ms (for the data packet transmission delay) plus 10 ms (for the one-way network delay) plus 40 ms (for the 50,000 bytes

of LinkEm buffer holding 40 data packets), which is 51 ms. Thus, no data packets will be able to meet the 30 ms deadline for this test.

The error control mode areas that have been shaded in this plot are only there to indicate which error control mode is in use at each test data point. The transition points between the error control modes may not actually occur as indicated in the plot.

The resulting plot shows that the tests with largest target packet delivery delay deadlines (e.g., 170 ms) can meet the delivery constraints by using pure ARQ mode (the top-most shaded area just below the theoretical limit line). Since pure ARQ mode is very efficient, the resulting raw goodput is high, which places the line close to the theoretical limit. The tests with the smallest target packet delivery delay deadlines (e.g., 30 ms) can only meet the delivery constraints by using pure FEC mode, as there is no time for retransmissions. Since pure FEC mode is the least efficient of the three modes, the resulting raw goodput is low, which places the line the furthest from the theoretical limit. The tests with the mid-range target packet delivery delay deadlines (e.g., 70 ms) can't meet the constraints by using pure ARQ mode, but can by using some FEC along with retransmissions, resulting in AEC choosing the coded ARQ mode. Since coded ARQ is not as efficient as pure ARQ, but is more efficient than pure FEC, the resulting raw goodput is in the middle.

The test result points at PER=0.5 for the larger target packet delivery deadlines do not necessary lie where one would expect them to given all of the other points on those lines. This is due to the very high data packet and ACK packet losses causing retransmission timeouts (RTO events) at the sender, leading to periods of no data packet transmissions while waiting for an ACK packet to be received.

3.3.3 AEC Efficiency Characteristics

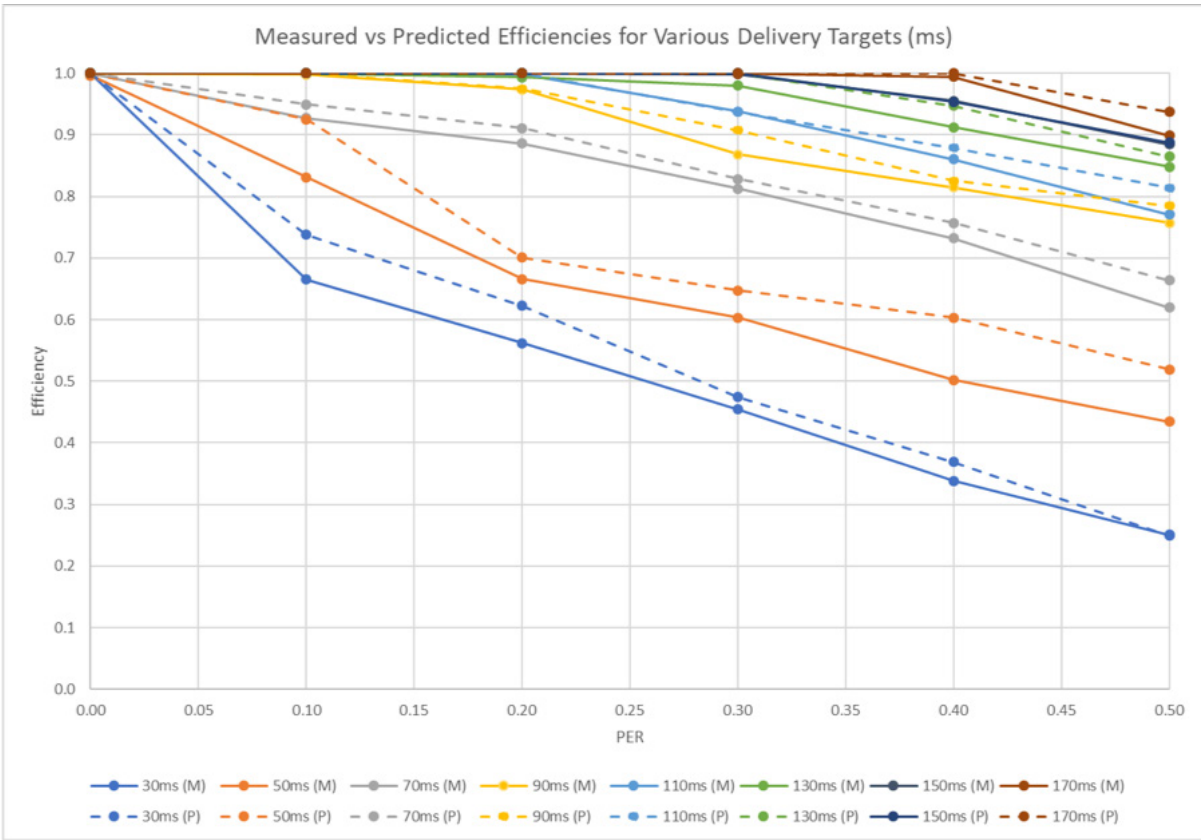
This test measures the actual efficiency of SLIQ's AEC algorithm compared to the algorithm's predicted efficiency over a range of PER and target packet delivery deadlines. For the purposes of this section, efficiency is the number of *usable* data packets received divided by the *total* number of data packets received. The predicted efficiencies take into account the exact channel properties as well as the AEC decision logic and lookup tables implemented in SLIQ. A series of individual tests were completed to create a plot of how the efficiency changes as PER is changed, for a set of target packet delivery deadline values.

The following tests were run with one flow originating at the client host on iron83 and terminating at the server host on iron84. Each test was 120 seconds long to minimize the impact of the SLIQ protocol starting up and calculating the first channel property estimates. The *LinkEm* software, running on iron86, was configured with the following parameters:

- 10 ms one-way delay (20 ms RTT)
- 10 Mbps throttling
- 50,000 byte maximum conform buffer size
- Bi-directional PER set to the following values: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5

The SLIQ stream was configured to use semi-reliable ARQ+FEC mode (which utilizes AEC), with a target packet delivery probability of 0.995 and a target packet delivery deadline of between 30 ms to 170 ms in 20 ms increments.

The results of these tests are shown in the following Efficiency vs. PER plot.



These results show that actual efficiencies are reasonably close to the predicted efficiencies. This implies that SLIQ is estimating the channel parameters reasonable well, and that the SLIQ AEC implementation is selecting the appropriate error control mode to use.

The largest differences between the measured and predicted values occur on the lines with a target packet delivery deadline of 50 ms (the orange line) and 30 ms (the blue line). These differences are likely caused by variations in the channel parameter estimates combined with quantization effects when using the AEC lookup tables.

3.3.4 AEC Delivery Delays

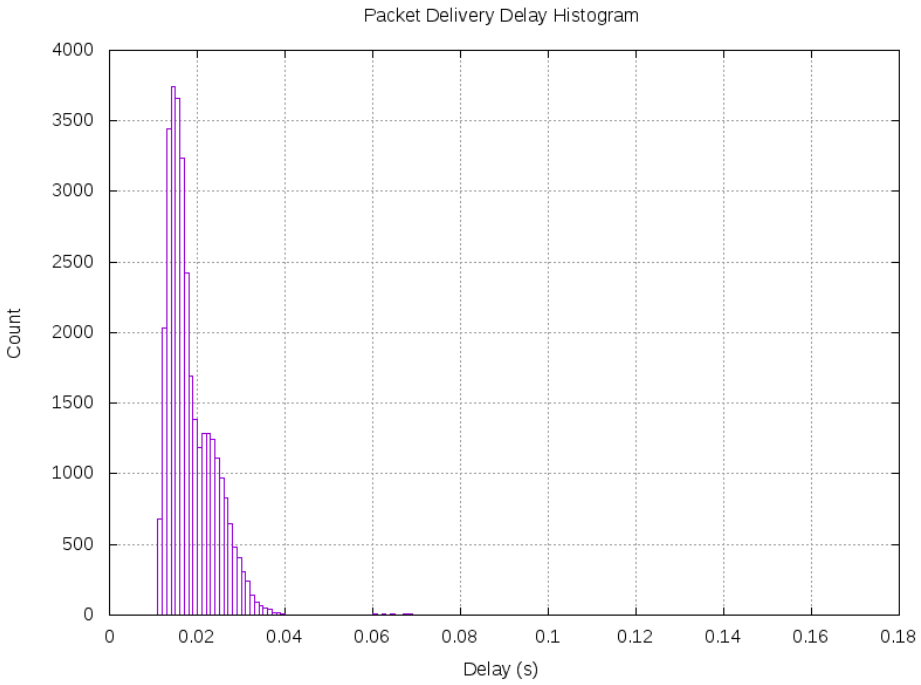
This test measures the actual application data delivery delays when using SLIQ's AEC algorithm for a number of target packet delivery deadlines. The resulting delivery delays are plotted in histograms to show the distribution of the delays.

The following tests were run with one flow originating at the client host on iron83 and terminating at the server host on iron84. Each test was 60 seconds long. The first and last few seconds of delivery delay data were discard so that only the steady-state data is plotted. The *LinkEm* software, running on iron86, was configured with the following parameters:

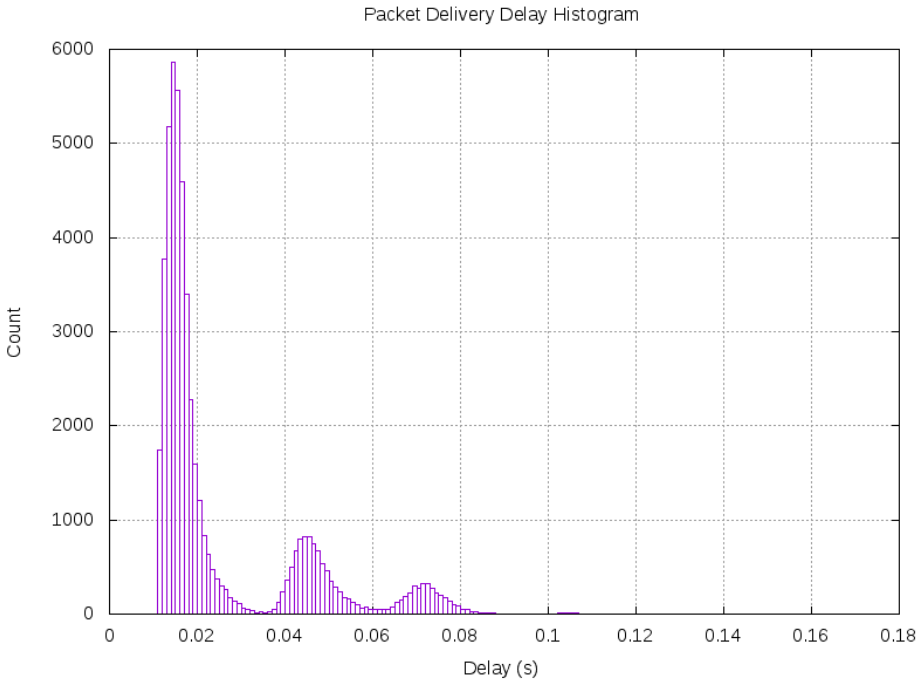
- 10 ms one-way delay (20 ms RTT)
- 10 Mbps throttling
- 50,000 byte maximum conform buffer size
- Bi-directional PER set to 0.4

The SLIQ stream was configured to use semi-reliable ARQ+FEC mode (which utilizes AEC), with a target packet delivery probability of 0.995 and a target packet delivery deadline of 50 ms, 110 ms, and 170 ms.

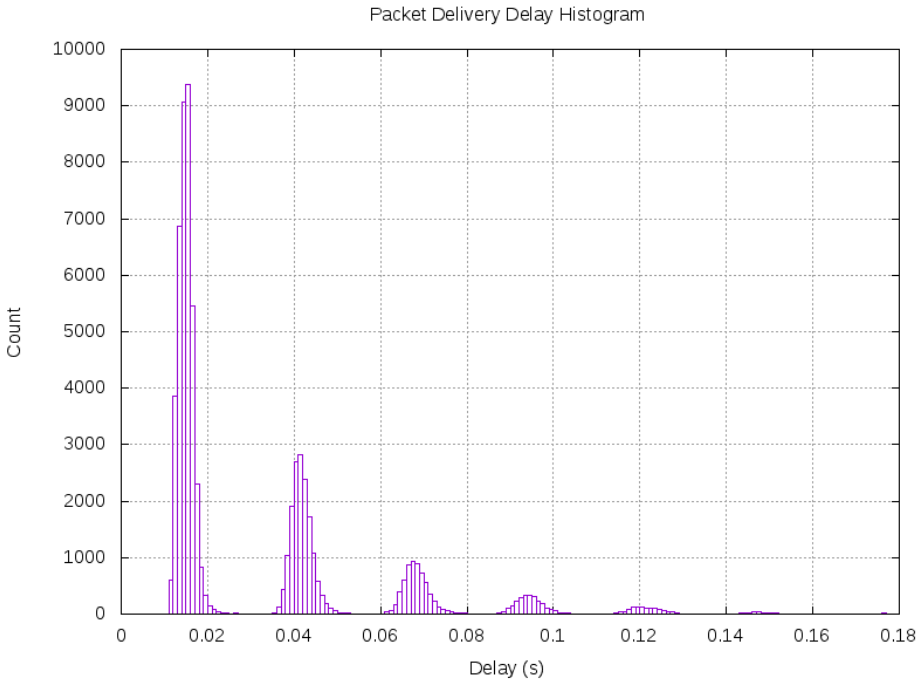
The delivery delay histogram for the test with a target packet delivery deadline of 50 ms is shown in the following plot. Note that in this test, AEC used pure FEC mode in order to meet the short delivery deadline. Thus, almost all of the delivery delays are grouped together as a result of the original FEC source and encoded data packet transmissions, and this grouping occurs within the 50 ms deadline. There are a small number of delivery delays just beyond 60 ms that are due to the very small number of data packets (less than 0.5%) that are still missing after the single FEC round. The achieved efficiency (the number of *usable* data packets received divided by the *total* number of data packets received) for this test is 52%, which is low due to the use of pure FEC mode.



The delivery delay histogram for the test with a target packet delivery deadline of 110 ms is shown in the following plot. Note that in this test, AEC used coded ARQ mode in order to meet the delivery deadline using only three possible retransmissions. The leftmost grouping is from the original FEC source and data packet transmissions, the next smaller grouping to its right is from the first retransmission of missing FEC packets, the next smaller grouping to its right is from the second retransmission of missing FEC packets, and the final tiny grouping to its right is from the third retransmission of missing FEC packets. These groupings all occur within the 110 ms deadline. The achieved efficiency for this test is 83%, which is higher than for the 50 ms test above due to the use of ARQ with FEC.



The delivery delay histogram for the test with a target packet delivery deadline of 170 ms is shown in the following plot. Note that in this test, AEC used pure ARQ mode in order to meet the delivery deadline using five possible retransmissions. The leftmost grouping is from the original data packet transmissions, the next smaller grouping to its right is from the first retransmission of missing data packets, and on up to the fifth retransmission of missing data packets. These groupings all occur within the 170 ms deadline. There are a small number of delivery delays just beyond 170 ms that are due to the very small number of data packets (less than 0.5%) that are still missing after the fifth retransmission. The achieved efficiency for this test is 99%, which is higher than for the 50 ms and 110 ms tests above due to the use of ARQ with no FEC.



It is important to note the tradeoffs that are occurring in these three tests. Pure ARQ mode achieves nearly perfect efficiency (99%), but takes the longest to deliver 99.5% of the application data. Coded ARQ mode achieves a lower efficiency (83%), but can deliver 99.5% of the application data sooner. Finally, pure FEC mode achieves that worst efficiency (52%), but can deliver 99.5% of the application data in the shortest possible time (a single FEC group round). Thus, efficiency increases with longer deadlines due to the use of more efficient coding schemes and the use of receiver feedback to pinpoint the missing data packets.

3.3.5 AEC Response to Path Changes

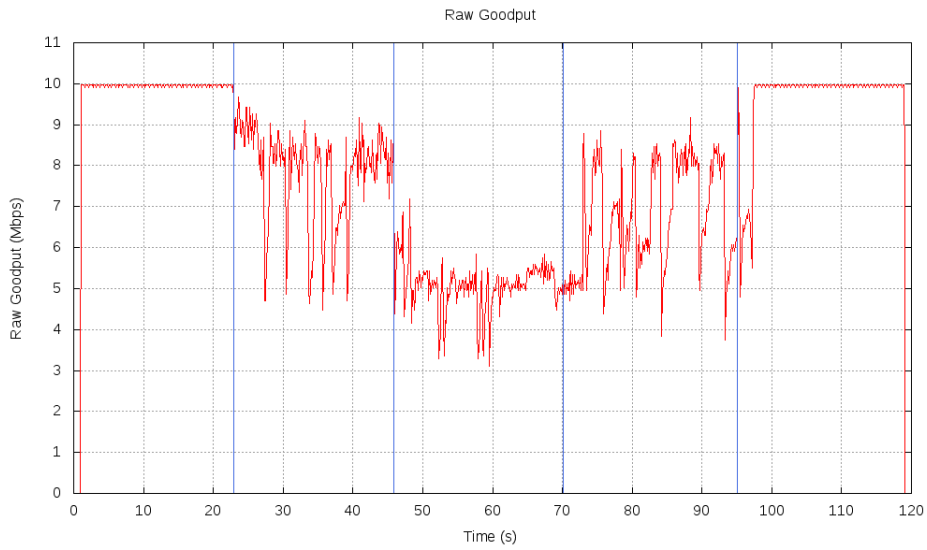
This test measures how well a single SLIQ flow using the AEC algorithm handles changes to the PER. This exercises AEC's ability to dynamically adapt to the underlying network conditions while still attempting to meet delivery constraints.

The following test was run with one flow originating at the client host on iron83 and terminating at the server host on iron84. The test was 120 seconds long with the PER changed approximately every 24 seconds. The *LinkEm* software, running on iron86, was configured with the following parameters:

- 10 ms one-way delay (20 ms RTT)
- 10 Mbps throttling
- 50,000 byte maximum conform buffer size
- Bi-directional PER set to the following values on 24 second intervals: 0.0, 0.1, 0.2, 0.1, 0.0

The SLIQ stream was configured to use semi-reliable ARQ+FEC mode (which utilizes AEC), with a target packet delivery probability of 0.995 and a target packet delivery deadline of 50 ms.

The results of these tests are shown in the following raw goodput plot.



Note that in this test, the PER changes at the times marked with blue vertical lines. The following table captures the detailed results of this test.

Time	PER	Predicted Raw Goodput	Actual Raw Goodput	AEC Mode
0-23 seconds	0.0	9.97 Mbps	9.97 Mbps	Pure ARQ
23-46 seconds	0.1	7.20 Mbps	7.51 Mbps	Coded ARQ
46-70 seconds	0.2	5.13 Mbps	5.11 Mbps	Pure FEC
70-94 seconds	0.1	7.20 Mbps	7.03 Mbps	Coded ARQ
94-120 seconds	0.0	9.97 Mbps	9.97 Mbps	Pure ARQ

It is clear from the plot that the AEC algorithm adapts the error control mode fairly quickly as the PER changes. There are some peaks and valleys in the plot, which are due to some packets arriving earlier than other packets at the receiver as well as fluctuations in the send rate at the sender due to switching between the CUBIC and Copa congestion control algorithms. The actual raw goodput matches the predicted raw goodput fairly well throughout this test. The one actual raw goodput that exceeds the predicted raw goodput is most likely due to SLIQ PER estimation errors and timing inaccuracies when the test was running.

3.4 TCP Proxy Performance Scaling with the Number of Flows

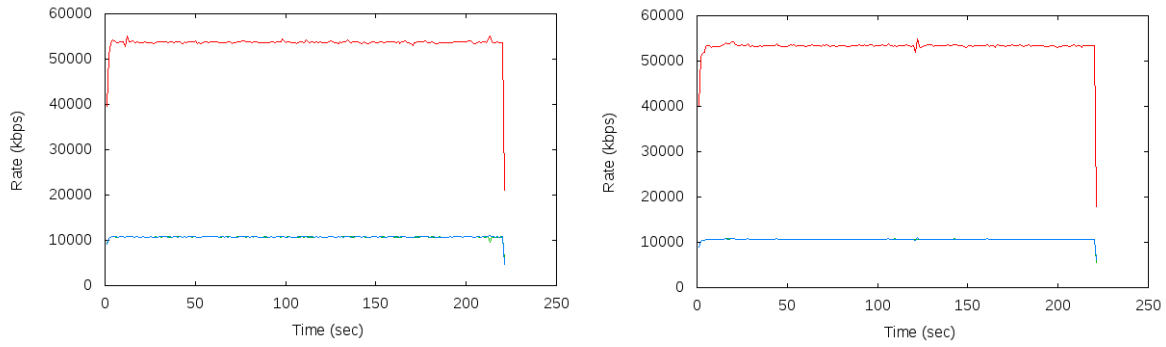
A number of experiments were run to determine the performance impacts on the TCP Proxy as the number of flows increased.

Experiment characteristics:

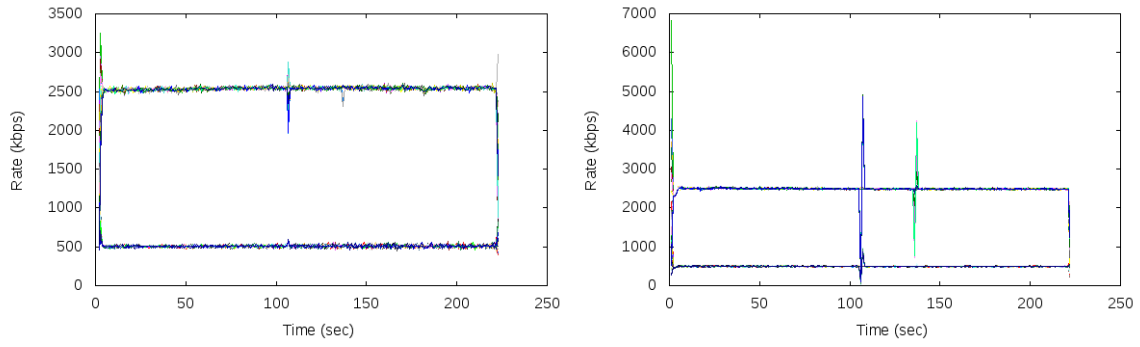
- Triangle topology, traffic from node0 to node1 and from node1 to node0
- LinkEm limits each link to 50 Mbps with 10 ms delay
- MGEN limits each flow to 100 Mbps
- Separate MGEN process for each flow
- Duration of experiment is 4 minutes
- No network impairments

All experiments were run on the BBN 90 testbed. Following are the TRPR plots for each of the experiments that were run. Analysis of these results demonstrates the TCP Proxy's ability to scale sub-linearly with the number of flows.

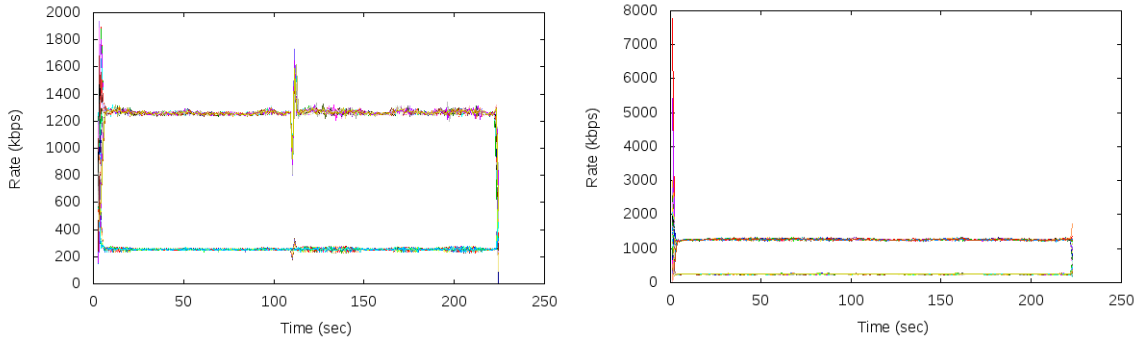
3.4.1 Experiment 1: High Priority flow and 2 Low Priority flows from each source node (total of 6 flows)



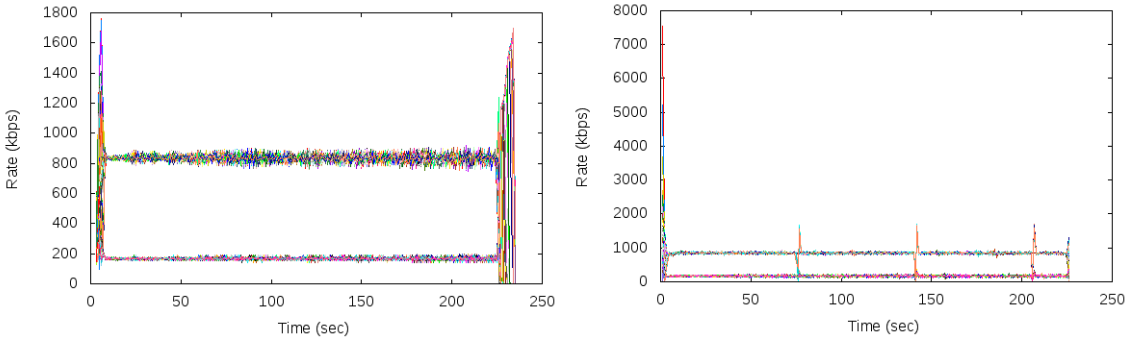
3.4.2 Experiment 2: High Priority flows and 25 Low Priority flows from each source node (total of 100 flows)



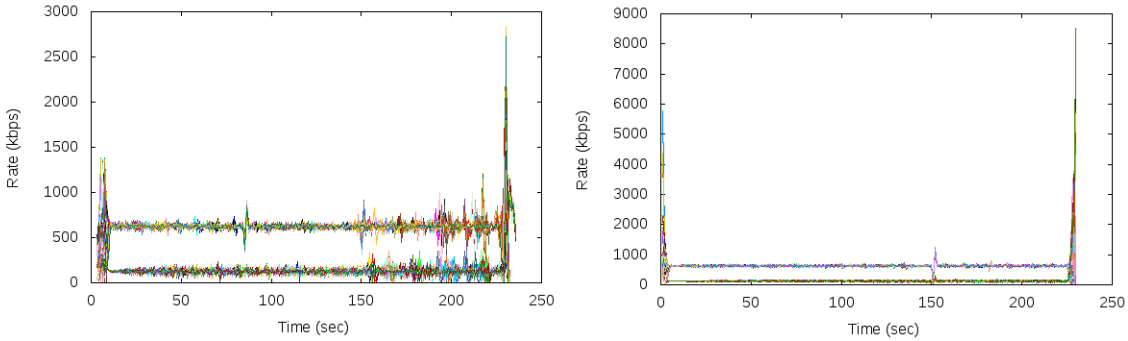
3.4.3 Experiment 3: 50 High Priority flows and 50 Low Priority flows from each source node (total of 200 flows)



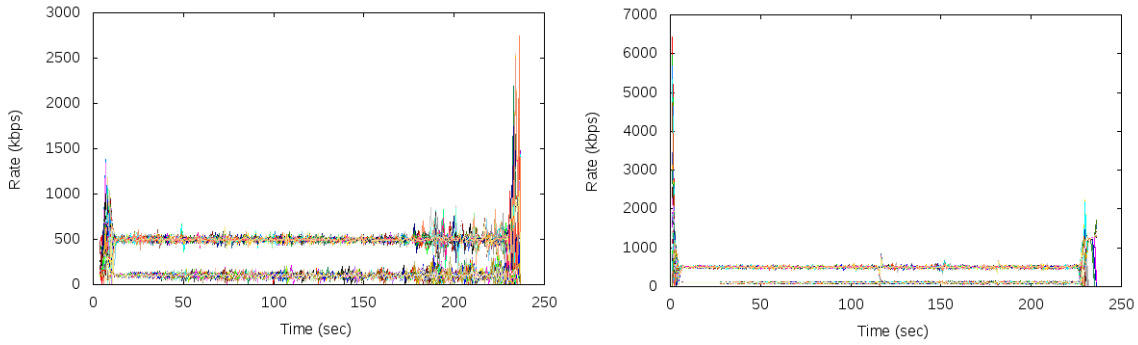
3.4.4 Experiment 4: 75 High Priority flows and 75 Low Priority flows from each source node (total of 300 flows))



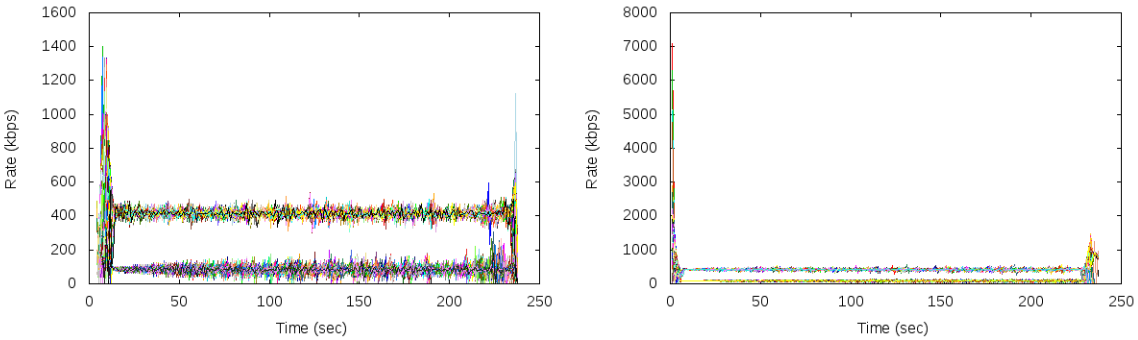
3.4.5 Experiment 5: 100 High Priority flows and 100 Low Priority flows from each source node (total of 400 flows)



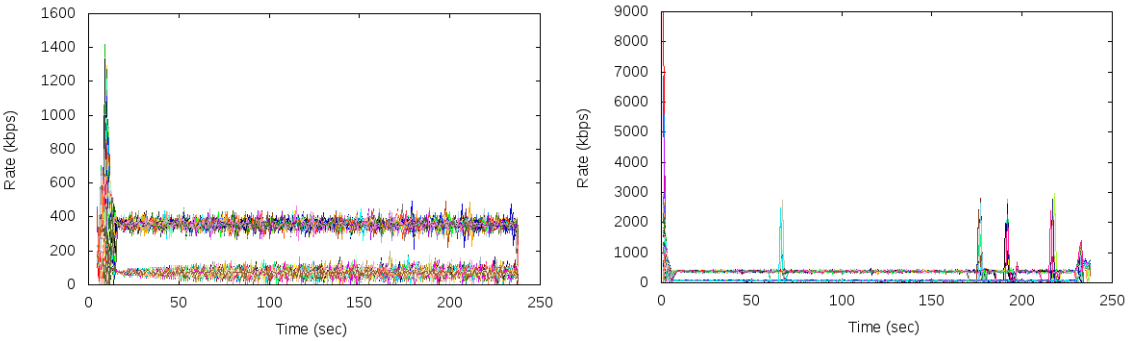
3.4.6 Experiment 6: 125 High Priority flows and 125 Low Priority flows from each source node (total of 500 flows)



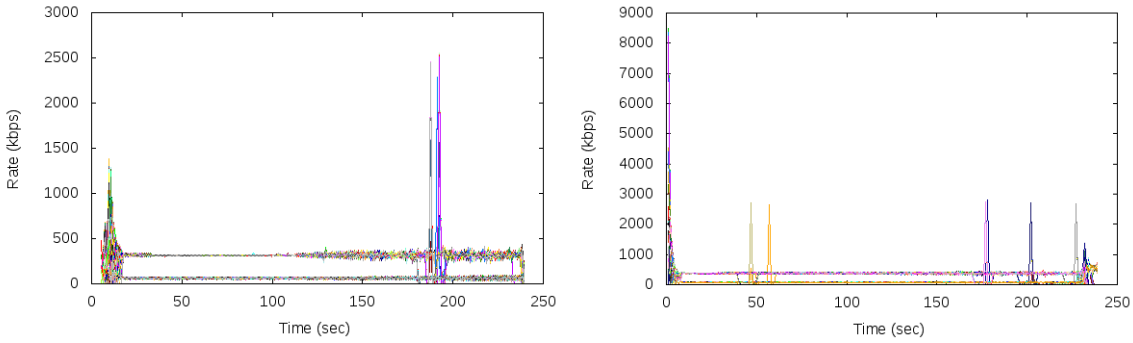
3.4.7 Experiment 7: 150 High Priority flows and 150 Low Priority flows from each source node (total of 600 flows)



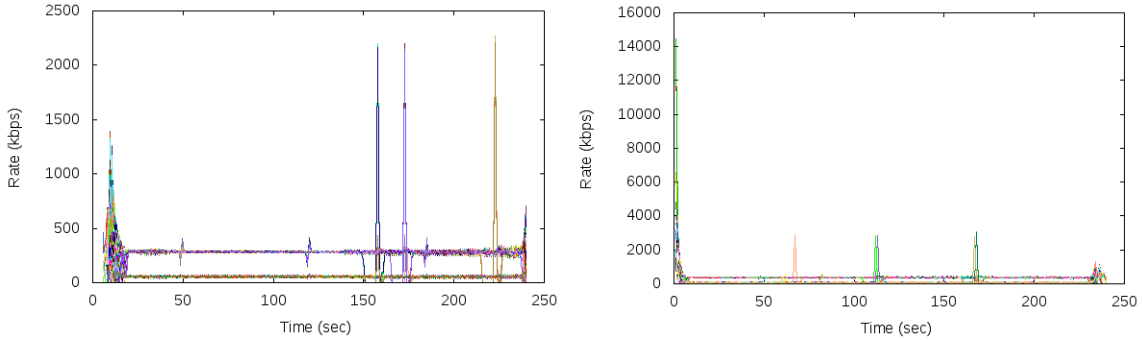
3.4.8 Experiment 8: 175 High Priority flows and 175 Low Priority flows from each source node (total of 700 flows)



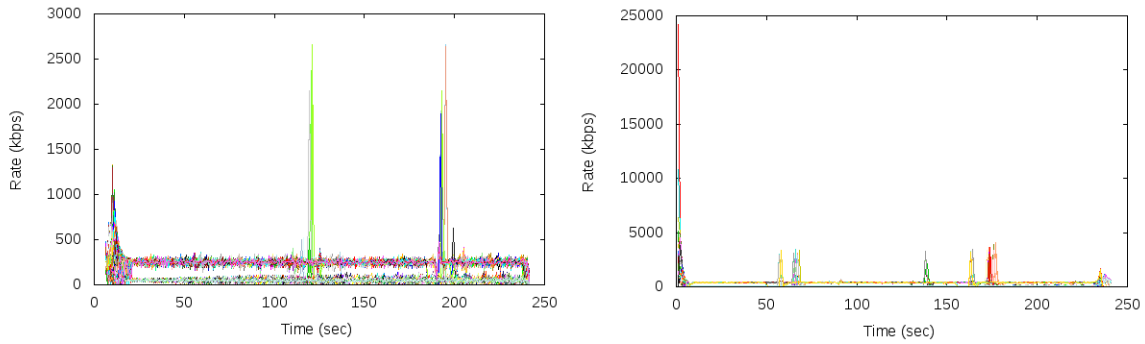
3.4.9 Experiment 9: 200 High Priority flows and 200 Low Priority flows from each source node (total of 800 flows)



3.4.10 Experiment 10: 225 High Priority flows and 225 Low Priority flows from each source node (total of 900 flows)



3.4.11 Experiment 11: 250 High Priority flows and 250 Low Priority flows from each source node (hence each proxy must handle a total of 1000 flows)



3.4.12 CPU Utilization

The following CPU utilization results were obtained by setting the following variable

MONITOR_PERF=true

in the experiment configuration file (exp.cfg) for each of the above runs. The table shows the average CPU utilization (maximum CPU utilization) for the TCP Proxy and the BPF.

Since these experiments were run on older hardware with very limited resources (2 processing cores, 4 MBytes physical memory) the absolute numbers here are not representative of GNAT performance, but rather are used to indicate relative scaling.

Total Number of Flows	TCP Proxy	BPF
6	50.24% (58.00%)	61.75% (84.00%)
100	53.27% (58.00%)	61.88% (85.00%)
200	60.03% (72.00%)	59.62% (83.00%)
300	57.84% (65.00%)	63.60% (86.00%)
400	60.62% (68.00%)	61.92% (85.00%)
500	63.09% (71.00%)	60.95% (84.00%)
600	65.49% (76.00%)	61.27% (85.00%)
700	69.06% (80.00%)	59.73% (83.00%)
800	69.20% (85.00%)	60.96% (86.00%)
900	71.43% (82.00%)	60.43% (86.00%)
1000	71.21% (84.00%)	59.21% (82.00%)

4 Measurement-based System Performance Assessments

A wide variety of system performance tests and analysis of test results is provided in this section. Testing is organized into four major categories as shown below. These categories emphasize different aspects GNAT behavior as a system. Short summaries of each test are also provided below.

Achieving DCOMP Program Metrics	
jms-diro	A two path experiment with competing traffic and link degradation demonstrating the ability to achieve 50X or greater improvement in throughput over baseline with response times less than 100ms
Multicast Performance	
6enclave_mcast_1grp	This experiment tests the ability of GNAT to maximally leverage the available capacity of a network with multiple diverse paths to multiple multicast receivers.
10enclave_mcast_5grp	This experiment tests the operation of GNAT with multiple competing multicast flows on more complex/difficult topologies
9enclave_mcast_asap	This experiment tests GNAT's support for low-volume multicast flows using ASAP.
34_node_dist_sim	This experiment demonstrates GNAT's ability to support publish-subscribe architectures using GNAT's ability to specify individual receivers on a per-packet basis at the sender.
118_node_dist_sim	This experiment demonstrates GNAT's ability to support sender-directed multicast in support of publish-subscribe architectures. This experiment features a comparison of GNAT behavior with a baseline configuration where GNAT is not used.
Overall System Performance	
3-node-system	Test basic GNAT operation with a triangle topology, including prioritization between TCP and UDP elastic flows and operation during impairments.
3-node-udp-perf	Performance test of a few UDP flows running over a slightly longer time period. Highlights GNAT's reduction of the queueing delay inherent in backpressure schemes.
3-node-tcp-perf	Performance test of a few TCP flows running over a slightly longer time period.
3-node-perf	Performance test of a moderate number of flows (both UDP and TCP) running for a moderate time period over relatively high-capacity links.
3-node-tcp-max	Performance test of high rate TCP traffic from few flows.
3-node-short-tcp	Tests that GNAT can support short-lived TCP flows with low start-up time and total transmission time.
3-node-tcp-large-num_flows	Tests that GNAT can support a large number of simultaneous TCP flows.
3-node-dynamics	Tests GNAT performance in the face of a dynamic network or dynamic traffic patterns.
y3_edge	Tests GNAT's ability to support fair prioritization and quick reactions even when the bottleneck is remote from the source.
12enclave_concurrent_mixed	Tests GNAT support of 12 enclaves simultaneously sending traffic.

Latency-sensitive Flows and BPF Features	
3-node-system-lat	Tests GNAT handling of latency-sensitive traffic, including operation during impairments.
4-node-system	Tests basic GNAT operation, including latency-aware routing
4-node-latency-routing	Tests a fully loaded network where latency-insensitive traffic is displaced by latency-sensitive traffic.
3-node-smallhighprio	Tests that GNAT does not starve small flows.
Prioritization and Admission Control	
3-node-util-vs-thruput	Tests GNAT prioritization when network utility maximization does not match goodput maximization.
4-node-strap	Demonstrates GNAT handling of inelastic flows.
3_node_960_flows	Performance test GNAT supporting many flows, both elastic and inelastic, including performance with link impairments.
4enclave_edge_big_flows	Performance test of GNAT supporting many flows, including elastic and inelastic, in a 4 enclave network.
3-node-flog	Tests triage with the Floored Log Utility function.
Admission Planner and Supervisory Control	
3-node-loss-triage	Tests that the UDP Proxy correctly triage latency-sensitive flows that are not meeting their deadlines and that AMP minimizes the number of flows that can probe.
3-node-latency-thrash	Tests that supervisory control will correctly triage latency-sensitive flows if the GNAT node can immediately detect that the deadlines cannot be met, including changing conditions due to impairments.
3-node-thrash	Tests that supervisory control will limit the number of thrashing flows, and that the use of inertia in the STRAP utility ensures that ongoing flows are given preference over new flows of equal priority.
6-node-thrash	Tests that supervisory control will limit the number of thrashing flows in a scenario where there is a remote bottleneck, shared by traffic from distinct source-destination pairs.
4-node-dist-triage	Tests that supervisory control and loss-based triage will limit the number of thrashing flows and the system will converge to a solution where nodes share bottleneck resources, without explicit coordination.

4.1 jms-diro

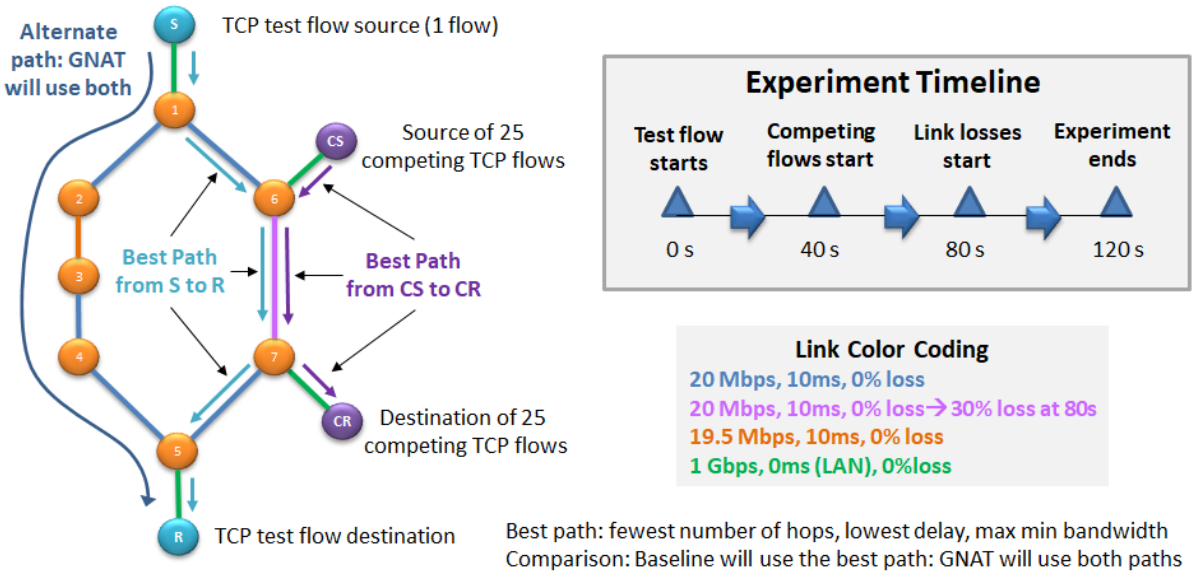
4.1.1 DCOMP Program Metrics

The program metrics for the networking portion (TA2) of the DCOMP Program specified 1) a 50X gain in utility, and 2) a response time of 100ms or less.

This experiment Illustrates how the combined use of multipath forwarding and hop-by-hop error and congestion control can yield 50X gains in goodput or greater on a simple two path topology with competing traffic with response times measurable in the 10s of ms.

This is a two-path experiment to demonstrate GNAT gains against a baseline. As shown below, there is a single TCP-based test flow from a sender at host node S to a receiver at host node R, both colored cyan. Network nodes, shown in orange, act either as standard routers (in the baseline case) or GNAT nodes. Two additional host nodes, shown in purple, provide competing TCP traffic, originating at node CS and terminating at node CR. In this set of experiments, competing traffic starts approximately 40 seconds after the test flow starts, and persists for the remainder of the experiment. The duration of the experiments is 120s.

All links from host nodes to network nodes run at full LAN speeds of 1 Gbps with negligible delay, and are lossless. These are shown in green. Most of the links between network nodes run at 20 Mbps with 10ms one-way delay, and are lossless. These are shown in blue. One of the links, shown in dark orange, is set at a slightly reduced rate of 19.4 Mbps, with one-way delay of 10ms and is lossless. The last link, shown in purple, has characteristics that vary over time for the purposes of the experiment. Initially it is set to 20 Mbps, with 10ms delay, and is lossless. Approximately 80s into the run the link loss is increased to 30% packet loss rate where it remains for the duration of the experiment.



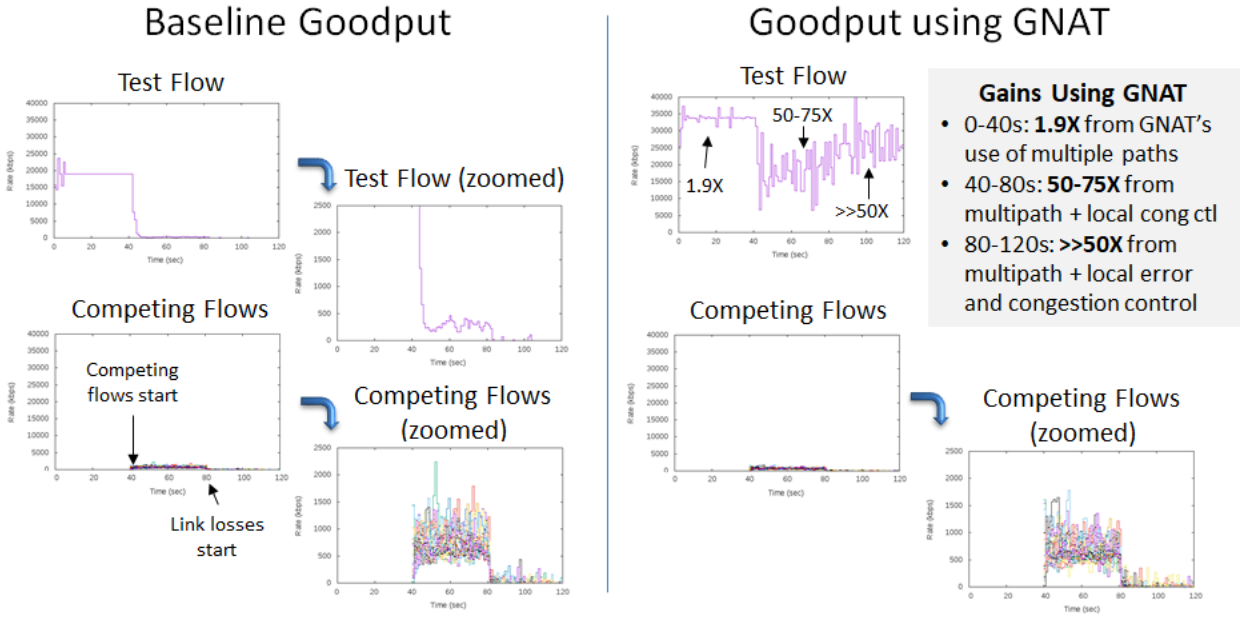
Using traditional, metrics-based single best path routing, the single best path from sender S to receiver R is along the sequence of network nodes 1 --> 6 --> 7 --> 5, as this has the fewest number of hops from S to R, the lowest end-to-end latency from S to R, and the greatest minimum capacity from S to R. In the baseline case, TCP traffic from S to R uses this single path. For the case with GNAT enabled forwarding, GNAT will use both the 1 --> 6 --> 7 --> 5 path *and* the 1 --> 2 --> 3 --> 4 --> 5 path concurrently.

The single best path from CS to CR is via the sequence of network nodes 6 --> 7. Since the competing traffic is not managed by GNAT, competing traffic will use this route for both the baseline case and the case where GNAT forwarding is enabled.

4.1.2 Expected results:

- For the first 40 seconds, the goodput for the single high rate tcp flow, which will use both paths since it is using GNAT, will be 20 + 19.5 ~ 40 Mbps (less tunneling overhead and TCP/IP header overhead).
- For the second 40 seconds, the goodput for the single high rate tcp flow will be reduced to 19.5 + 20Mbps/26 ~= 20 Mbps (less tunneling and TCP/IP overhead). Note that this estimate assumes that the GNAT-supported tcp flow and the other 25 tcp flows over the link between GNAT nodes 6 and 7 share capacity equally. Actually, since the CUBIC-based error control loop between GNAT nodes 6 and 7 can be closed more quickly, it will get a increasingly greater portion of the link capacity
- For the final 40 seconds, the goodput for the single high rate tcp flow will recover towards 19.5 + 0.7 * 20 Mbps ~= 34Mbps (less tunneling and TCP/IP overhead) as the 25 competing tcp flows will not be able to achieve significant flow rates over the high loss link.

The goodput results for the test TCP flow from the baseline run (without GNAT) and the GNAT-enabled run are shown in the figure below. As can be seen from the goodput charts, GNAT achieves about 2X increased goodput over baseline due to GNAT's concurrent use of multiple paths during the first 40s interval when there is not competing traffic; GNAT achieves 50X to 75X increased goodput over baseline due to both multipath and local congestion control in the second 40s interval where there is competing traffic; GNAT achieves >>50X increased throughput in the final 40s interval when there is both competing traffic and path 30% path loss. This is due to GNATs ability to achieve near capacity delivery in high loss conditions, in contrast to the competing traffic's use of CUBIC congestion control which cannot tolerate high loss rates.



Finally, GNAT responds very quickly to changes in link conditions, typically on the order of a round trip times (here a round trip time is on the order of 50ms). This can be seen, for example, in GNAT's shift from relying primarily on CUBIC during the first and second 40s intervals, to relying primarily on COPA in the third 40s second interval. This shift occurs fast enough that the throughput is not reduced during the transition (compared to that of the competing traffic), and continues to climb over the next 10-20 seconds as the control system dynamics settle into a new equilibrium near 30 Mbps average.

Note: These goodput curves appear somewhat jagged as there is the familiar head-of-the-line blocking common to TCP for both the test flow and the competing flows.

Experimental detail 1: This is a specialized experiment that includes competing TCP traffic from CS to CR below that is ***not*** managed by IRON/GNAT. This is enabled by making GNAT nodes 6 and 7 purely interior nodes in the bin_map, and by not having tcp or udp proxies. Without the proxies, there is no way for the competing traffic to get into IRON/GNAT. Hence there are no amp instances, TCP proxies, or UDP proxies configured for GNAT nodes 6 and 7.

Experimental detail 2: Setting up this topology requires making sure that the appropriate subnets are added to the link emulators (LinkEms) associated with GNAT nodes 6 and 7.

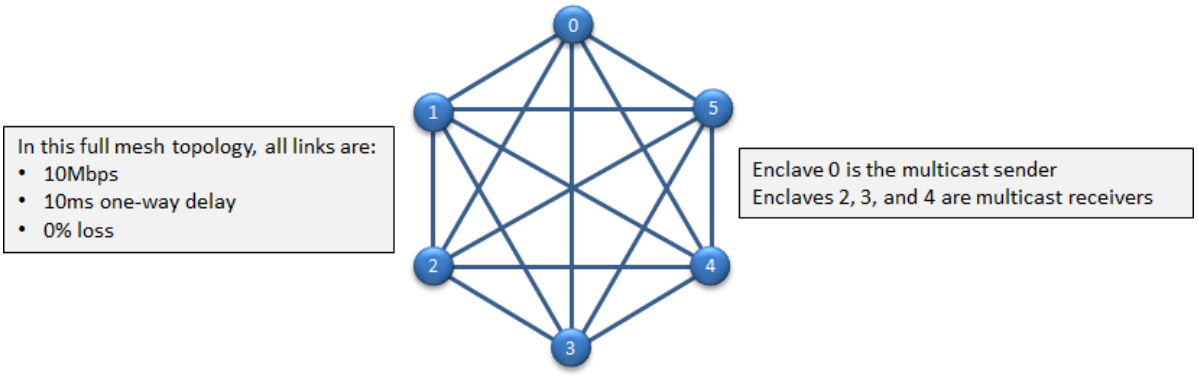
4.2 6enclave_mcast_1grp

4.2.1 Goal

This experiment tests the ability of GNAT to maximally leverage the available capacity of a network with multiple diverse paths to multiple multicast receivers. This experiment approximates the first stage of a Jupiter experiment where the Directed Acyclic Graph (DAG) specifies sending the same file to each of three independent receivers in a 6 node network. This experiment is only a capacity test to assess basic forwarding properties, as it does not use our *nftp* application to move an actual file using reliable multicast (NORM)

4.2.2 Topology and Conditions

The experiment runs over a 6-node full mesh topology as shown in the figure below. All links are set to 10 Mbps, with 10ms one-way delay, and no loss. There is a single multicast (UDP) sender located at node 0, and three multicast receivers located at nodes 2, 3, and 4. The sender is configured to send much faster (100 Mbps) than the aggregate multicast capacity of the network (the min cut set to each receiver is 5 separate paths at 10Mbps for a total of 50 Mbps). The experiment uses a LOG utility function configured (via AMP) to send at a maximum rate of 100 Mbps, such that all rate regulation to keep the network in the feasible region is done by the admission controller in the UDP proxy at node 1. The experiment runs for 60s.

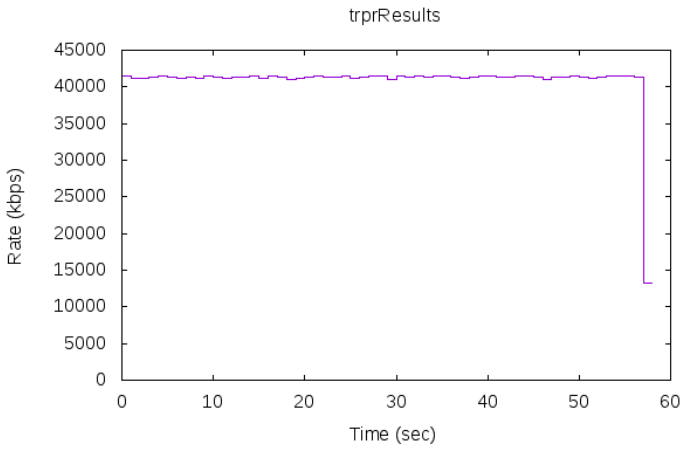


4.2.3 Expected Results

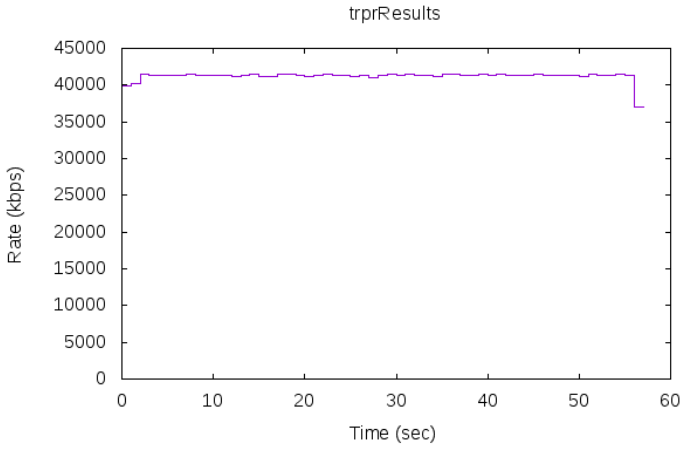
Each of the three receivers will receive approximately 50 Mbps of traffic, less tunneling and packet header overhead. All receivers should receive the same amount of traffic, and traffic flows should be relatively constant for the duration of the experiment.

4.2.4 Observed Results

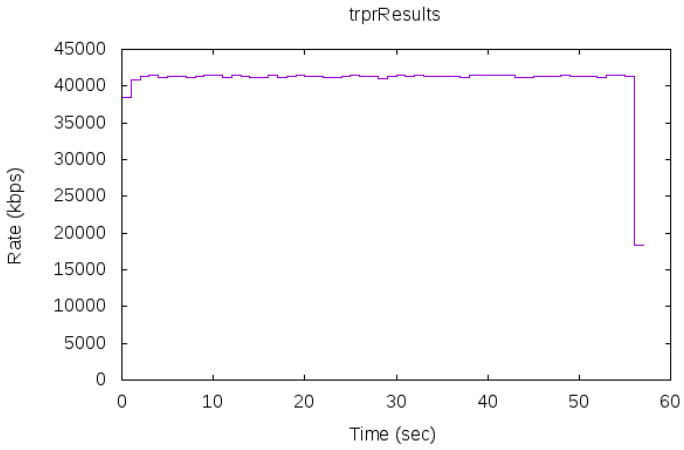
As shown in the goodput graphs below, flows converge rapidly (within about 1s after startup) to a stable goodput of about 42 Mbps, which is 84% of the available network capacity, so an effective 16% reduction due to overhead and potential forwarding inefficiencies. For unicast experiments the observed overhead is about 12.5%; for multicast experiments overhead is slightly greater due to the need to carry multicast destination vectors. Overall this experiment is successful, but also hints that additional performance gains may be possible.



Goodput observed at Node 2



Goodput observed at Node 3



Goodput observed at Node 4

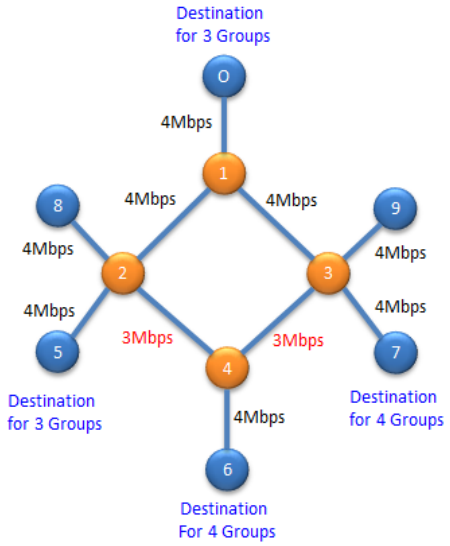
4.3 10enclave_mcast_5grp

4.3.1 Goal

This experiment tests the operation of GNAT with multiple competing multicast flows.

4.3.2 Topology and Conditions

The experiment runs over a 10-node diamond shaped topology as shown in the figure below. Notionally all links in the topology are set to 4 Mbps, although during the course of the experiment link rates are changed to highlight dynamic adaptation to change.



Five of the nodes – 0, 5, 7, 8, and 9 – are the sources for five different multicast flows. The multicast receiver set, and assigned IPv4 multicast address for each flow is

0 -> (6,7) (224.0.10.10)

5 -> (0, 6, 7) (224.1.10.10)

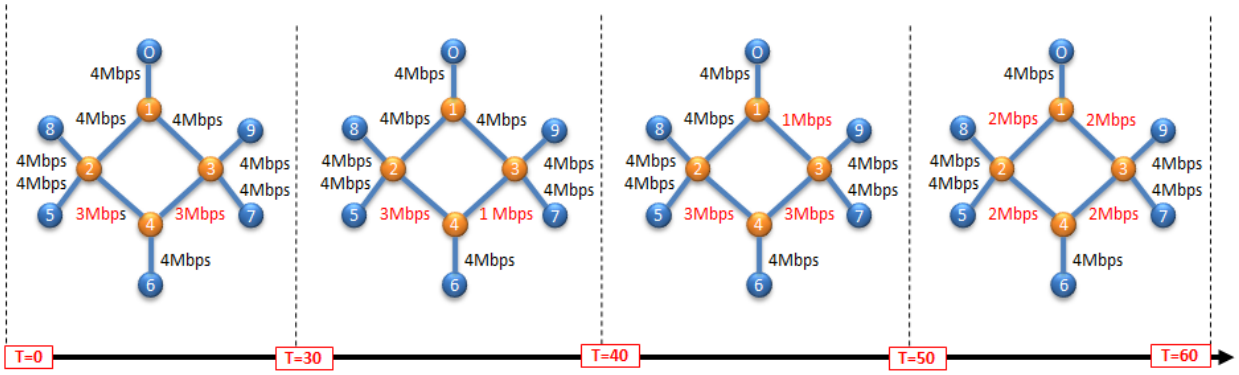
7 -> (0, 5, 6) (224.2.10.10)

8 -> (5, 6, 7) (224.3.10.10)

9 -> (0, 5, 7) (224.4.10.10)

Each multicast (UDP) flow is configured via AMP to use a LOG utility function having a maximum send rate of 25 Mbps. The flows themselves are generated by mgen with an offered rate of 10 Mbps. Hence any single flow could use all available network capacity (min cut set capacity is 8 Mbps). All utility functions use a priority value of 1. Hence this experiment is design to highlight equal load sharing between all flows.

The experiment is consists of 4 intervals. The first interval is 30s long to allow for the network dynamics to reach an equilibrium state. Subsequent intervals are each ten seconds long. During each interval the link speeds for multiple links are changed as highlighted in red in the Figure below.



4.3.3 Expected Results

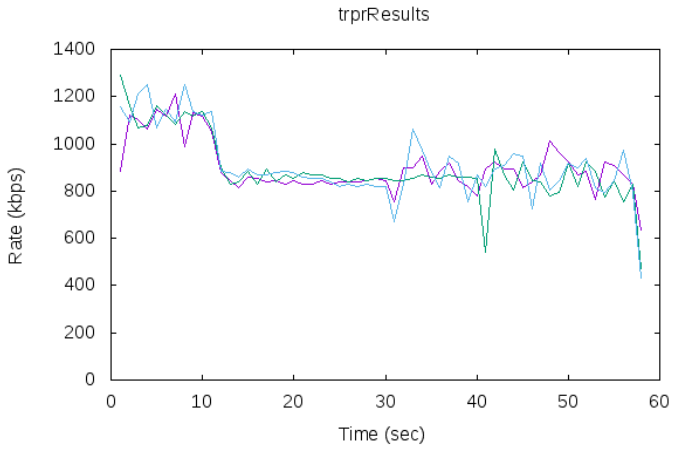
The sequence of link changes are made in such a way that all multicast flows should still be deliverable to their receiver set at a shared rate of 1 Mbps per flow, although adapting to these changes sometimes requires quite different overlapping multicast trees and so is a fairly strenuous test.

4.3.4 Observed Results

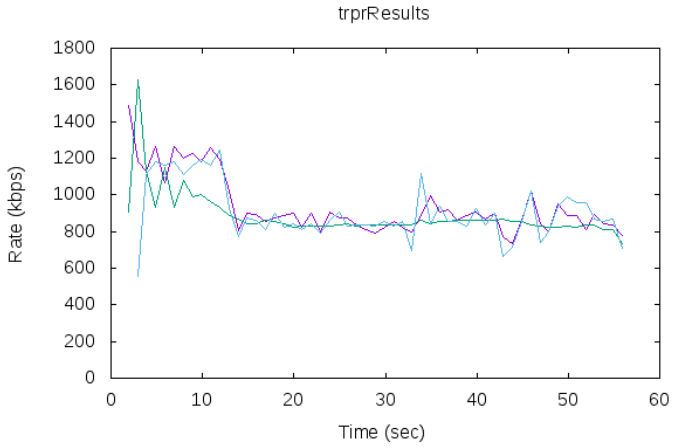
As shown in the goodput graphs below, the inbound rates to all nodes reach a consistent rate of about 850Kbps per flow after about 12s. A rate of 850Kbps well approximates an equally shared throughput rate of 1 Mbps per flow less tunneling and packet overhead.

Since the rates to those nodes receiving four groups (nodes 6 and 7) limits the shared capacity for each flow to 1 Mbps of throughput each, the throughput rates to nodes receiving three groups (nodes 0 and 5) are effectively underutilized. Initially there is a surge in traffic to nodes 0 and 5 where the network tries to fill the 4 Mbps of available link capacity with 3 flows, but eventually the joint constraint of having the same rate for any given flow at each receiver limits the inbound rates to nodes 0 and 5 to 1 Mbps each. The under use of the network does still exhibit some variability for these two nodes as shown in the results below.

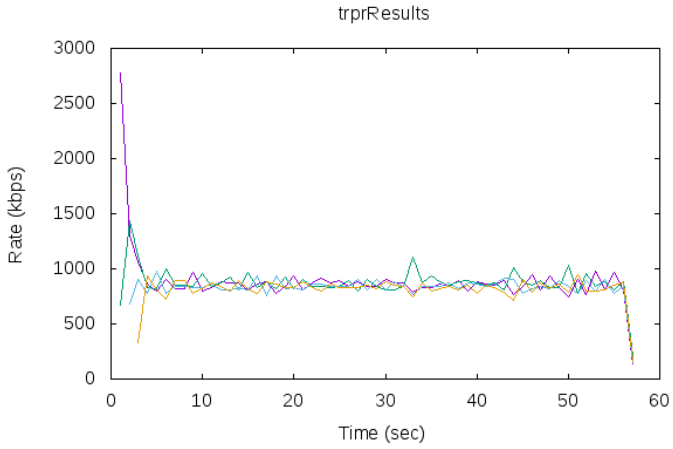
Nonetheless, the inbound rates to the other two nodes (nodes having four flows) fills the access channels and exhibits consistent and stable throughout the experiment.



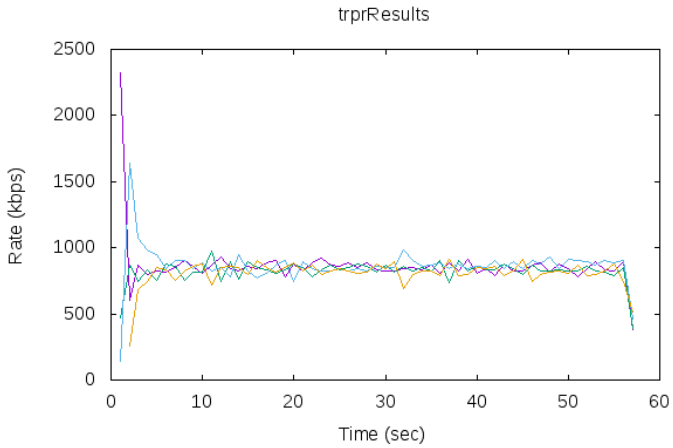
Inbound traffic observed at node 0



Inbound traffic observed at node 5



Inbound traffic observed at node 6



Inbound traffic observed at node 7

4.4 9enclave_mcast_asap

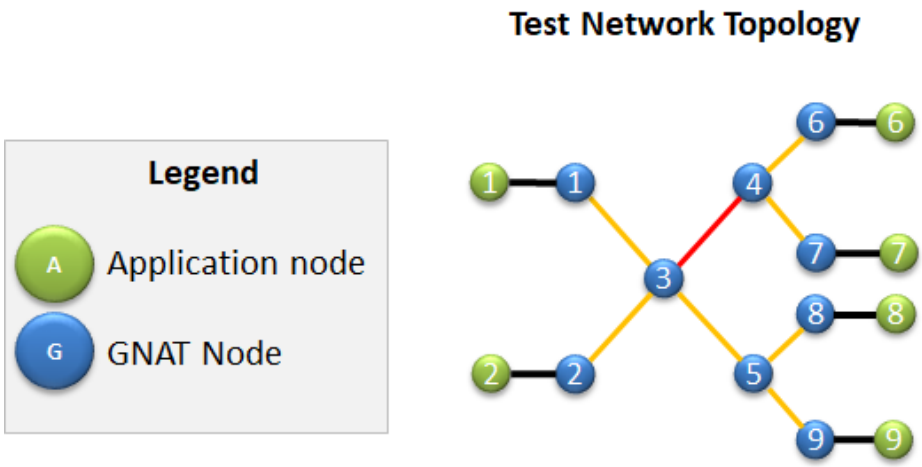
4.4.1 Goal

This experiment tests GNAT's support for low-volume multicast flows using ASAP.

4.4.2 Topology and Conditions

The experiment runs over an X-shaped topology as shown in the figure below. All access links (shown in black between green Application nodes and blue GNAT nodes) are 1 Gbps, with no delays and no loss. Most of the links between GNAT nodes (orange) are set to 10 Mbps with 10ms one-way delay and no loss. The remaining link between GNAT nodes 3 and 4 (red) is set to 8Mbps to create a bottleneck link within the topology. The delay and loss characteristics for this bottleneck link are the same as that of the other links between GNAT nodes.

There are two multicast senders, located at App nodes 1 and 2 respectively. App node 1 is configured to send UDP multicast traffic to receivers at nodes 6 and 7 at an offered load of 20 Mbps. App node 2 is configured to send UDP multicast traffic to receiver nodes 7, 8, and 9 at an offered load of 8 Kbps. The same LOG utility function with a maximum send rate of 25 Mbps and priority value of 5 is used for both the high and the low volume flows.

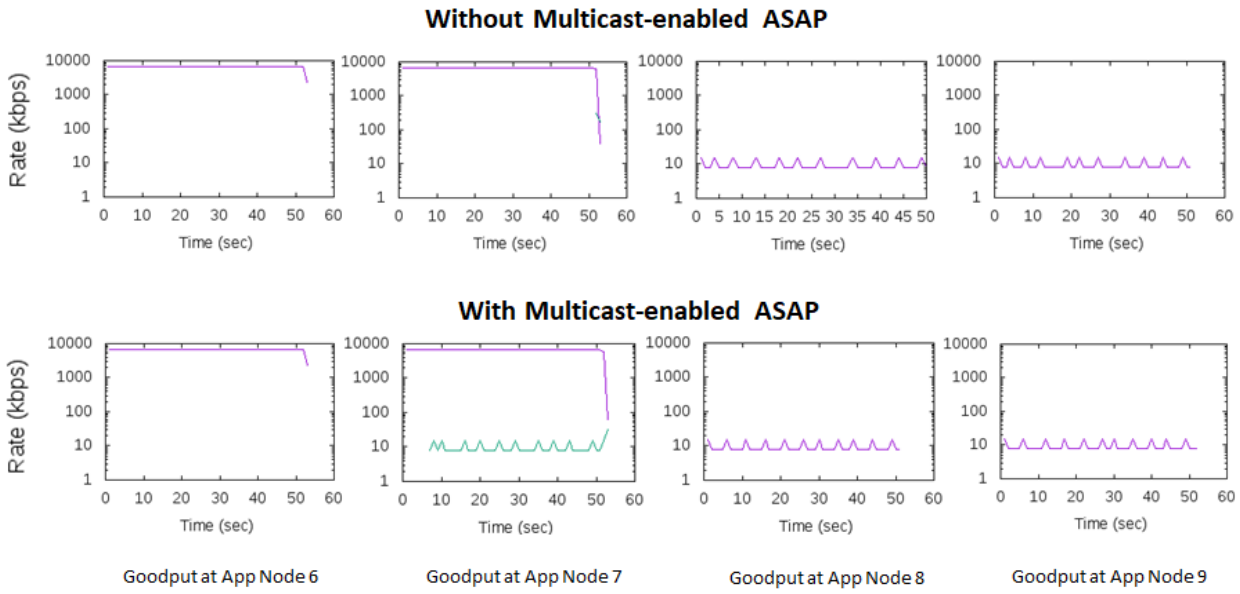


4.4.3 Expected Results

Without ASAP enabled, the high rate flow will squeeze out the low rate flow, so that no low rate traffic is delivered to the low rate receiver at node 7. With ASAP enabled, the low rate flow will (after sufficient ASAP reaction time) will achieve its 8 Kbps rate to node 7.

4.4.4 Observed Results

The goodput plots for all nodes and all flows are provided below. Here the expected behavior for the low volume flow observed at node 7 both without and with ASAP is confirmed.


Without Multicast-enabled ASAP
With Multicast-enabled ASAP
Rate (kbps)
Time (sec)
Goodput at App Node 6
Goodput at App Node 7
Goodput at App Node 8
Goodput at App Node 9

4.5 34_node_dist_sim

4.5.1 Goal

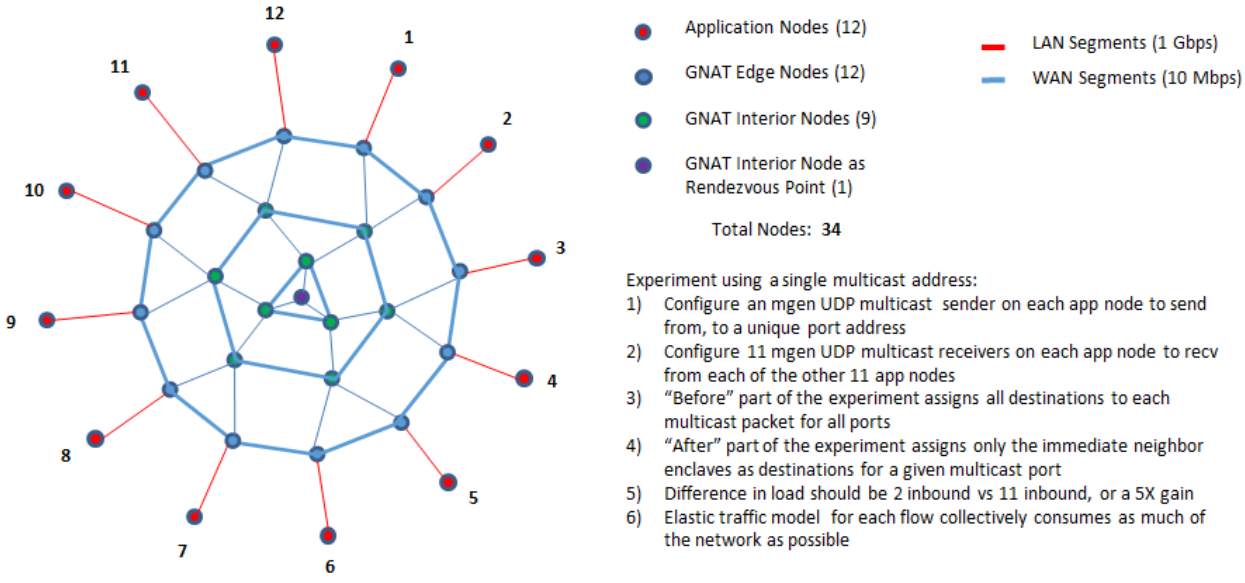
This experiment demonstrates GNAT's ability to support publish-subscribe architectures using GNAT's ability to specify individual receivers on a per-packet basis at the sender.

4.5.2 Topology and Conditions

This experiment uses a web-shaped topology as shown in the figure below. All access links (shown in red between red Application hosts and blue GNAT nodes) are 1 Gbps, with no delays and no loss. All links between GNAT nodes (blue and purple) are set to 10 Mbps with 10ms one-way delay and no loss. All GNAT nodes (blue and purple) are configured as PIM Sparse Mode routers (PIM-SM) with the center-most GNAT node acting as the designated rendezvous point.

All application hosts act as both multicast senders and multicast receivers, with all using the same IPv4 multicast address. Each app node is configured to send UDP multicast traffic at an offered load of 50 Mbps. GNAT nodes are configured to use a LOG utility function with equal priorities and a maximum admission rate of 50 Mbps. GNAT nodes are also configured to specify multicast receiver membership consisting only of the immediate neighbors clockwise and counterclockwise around the outer edge. This mimics a publish-subscribe configuration where only immediate neighbors are subscribed to a given sender's stream, providing what is known in the distributed simulation community as "interest filtering".

In order to be able to track the performance of individual multicast flows, each sender is configured to use a different UDP port, and each receiver is configured to listen to all UDP ports except its own. Protocol ports are generally ignored in routed infrastructures (and are also ignored within our backpressure forwarding network), and are used here only for the purposes of per-flow accounting.



4.5.3 Expected Results

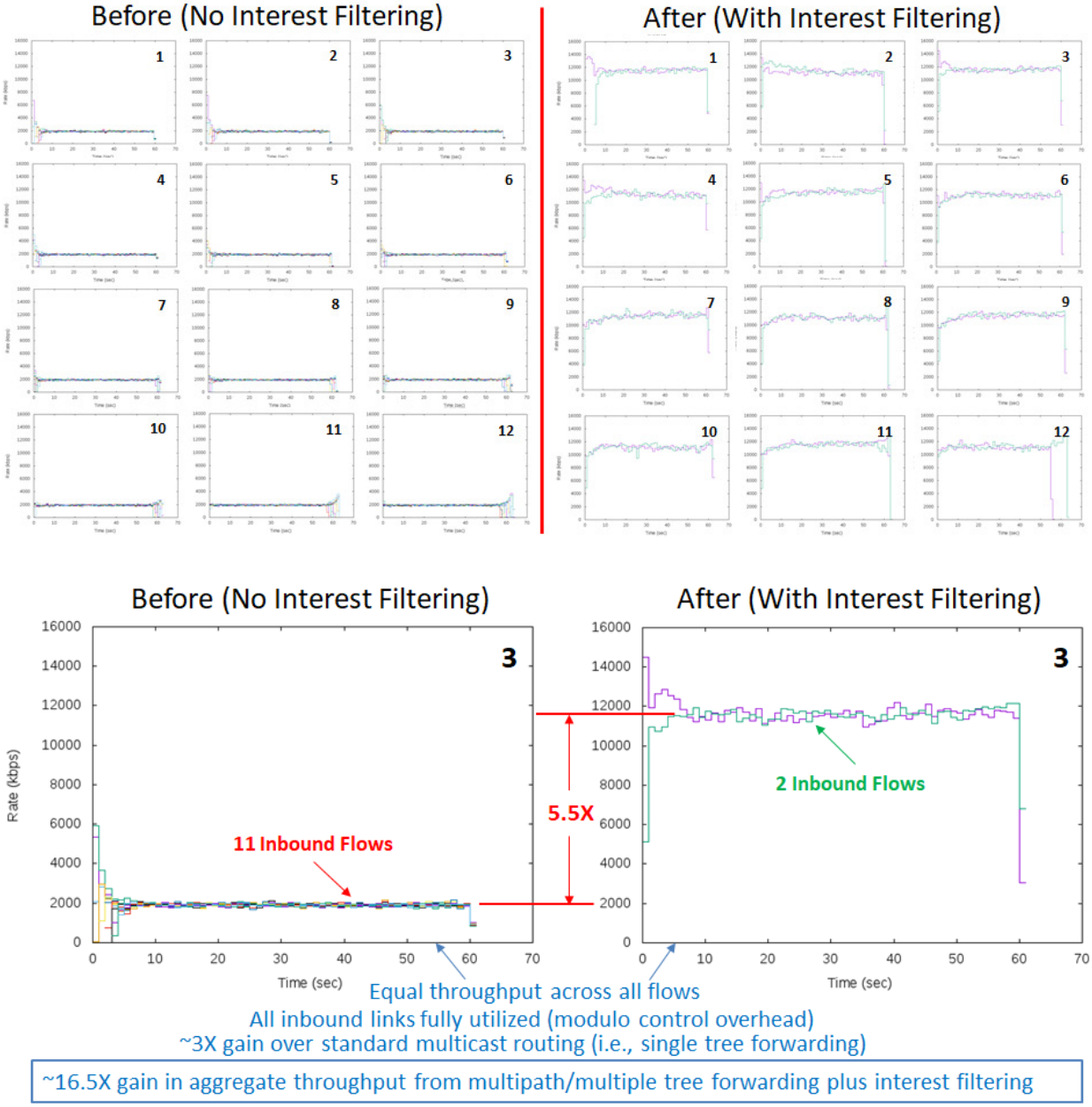
Without leveraging GNAT's ability to specify forwarding of individual multicast packets to individual receivers at the sender, each receiver will receive traffic from all other senders. This translates to total inbound capacity to each host node (3 links at 10 Mbps per link) being split evenly between the 11

senders. Hence the inbound receive rates for each flow are about 30 Mbps/11 flows of raw throughput, or about 2.2 Mbps/flow of goodput after accounting for tunneling and packet overhead.

Using sender-specified receiver membership, only packets from two immediately-adjacent senders will be received, with a corresponding increase in both throughput and goodput of 11/2 = 5.5.

4.5.4 Observed Results

The results observed at each node confirm per-flow goodputs close to the expected rates both without (2.2Mbps/flow) and with interest filtering (12 Mbps). Further below shows a zoom-in on node 3, highlighting the expected 5.5:1 gain using interest filtering.



4.6 118_node_dist_sim

4.6.1 Goal

This experiment demonstrates GNAT's ability to support sender-directed multicast in support of publish-subscribe architectures. This experiment features a comparison of GNAT behavior with a baseline configuration where GNAT is not used.

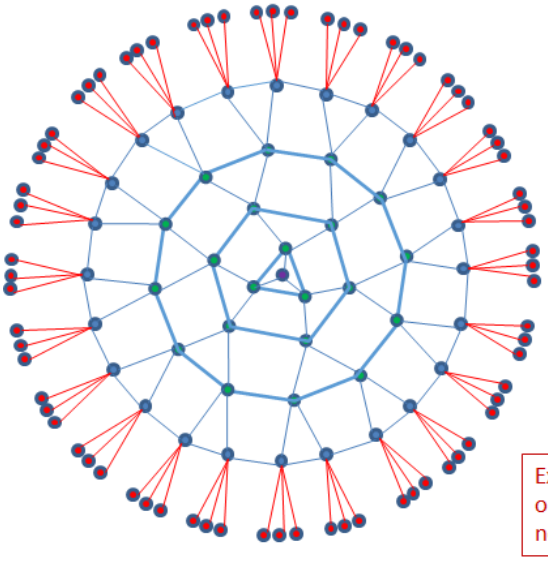
Note: This experiment was run on the DCOMP program's Merge testbed because of the large number of compute nodes involved.

4.6.2 Topology and Conditions

This experiment uses a web-shaped topology as shown in the figure below. All access links (shown in red between red Application hosts and blue/purple GNAT nodes) are 1 Gbps, with no delays and no loss. All links between GNAT nodes (blue) are set to 10 Mbps with 10ms one-way delay and no loss. All blue nodes are configured as PIM sparse mode routers (PIM-SM) with the center-most GNAT node (purple) being the designated rendezvous point.

All application hosts act as both multicast senders and multicast receivers using the same IPv4 multicast address. Each app node is configured to send UDP multicast traffic at an offered load of 25 Mbps. GNAT nodes are configured to use a LOG utility function with equal priorities and a maximum admission rate of 25 Mbps. GNAT nodes are also configured to specify multicast receiver membership consisting only of host nodes in the immediate neighbor enclaves clockwise and counterclockwise around the outer edge. This mimics a publish-subscribe configuration where only immediate neighbors (here, 3 nodes in each neighboring enclave) are subscribed to a given sender's stream, providing what is known in the distributed simulation community as "interest filtering".

In order to be able to track the performance of individual multicast flows, each sender is configured to use a different UDP port, and each receiver is configured to listen to all UDP ports except its own. Protocol ports are generally ignored in routed infrastructures (and are also ignored within our backpressure forwarding network), and are used here only for the purposes of per-flow accounting.



Application Nodes (72)
GNAT Edge Nodes (24)
GNAT Interior Nodes (21)
GNAT Interior Node as Rendezvous Point P (1)
Total Nodes: **118**

LAN Segments (1 Gbps)
WAN Segments (10 Mbps)

Experiment uses a single multicast address for all traffic
1) Configure an mgen UDP multicast sender on each app node to send from, to a unique port address
2) Configure 71 mgen UDP multicast receivers on each app node to recv from each of the other 71 app nodes
3) Baseline experiment: without GNAT, just pure all-to-all multicast
4) With GNAT: receivers subscribe only to traffic from senders in their immediate neighbor enclaves (will also get LAN traffic, which we don't count)
5) Load delta to each app node should be 6 inbound vs 69 inbound, or a ~10X
6) GNAT also leverages multiple paths vs std multicast routing for addtl 3X gain
7) Elastic traffic model for each flow collectively consumes as much of the network as possible

Experiment design goal: model the ~**10:1** traffic reduction obtained under our Active Nets project only with a larger network and demonstrate improved transport characteristics

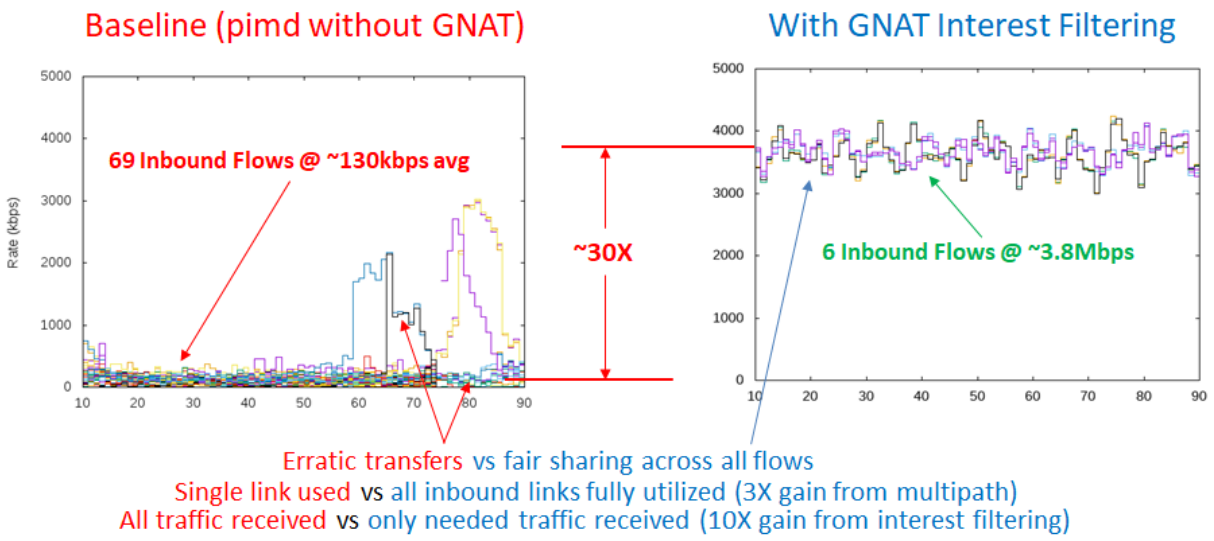
4.6.3 Expected Results

Without GNAT and GNAT's ability to limit forwarding of individual multicast packets to specific receivers, all receivers will get all traffic from all other senders (71 senders in this experiment). In the baseline case, which uses standard "single multicast tree" routing, this implies each of the 69 flows from different enclaves at a given receiver will be comprised of 1/69th of the total capacity of a 10Mbps single link. This translates to 144 Kbps per flow. (Note: although each receiver observes traffic from 71 different senders, 2 of the senders are within its own enclave and hence do not need to traverse the larger network.)

With GNAT, each receiver gets traffic both from its neighbor hosts in the same enclave, and from neighbor hosts in the two immediately adjacent enclaves. Only the traffic from hosts in adjacent enclaves need traverse the larger network, meaning that a total of 6 flows share the 30 Mbps inbound capacity, yielding per-flow throughput rates of 30/6 or 5 Mbps per flow. This represents a net gain of nearly 35X in throughput

4.6.4 Observed Results

Goodput plots are shown below for the baseline case (left, using standard pimd routing, without GNAT) and with GNAT leveraging interest filtering (right). Here we see that the realized gains are only approximately 30X. The reduction in achieved gains is in part due to the additional overhead imposed by tunnel with GNAT, and partly due to suboptimal forwarding performance with GNAT's current implementation.



4.7 3-node-system

4.7.1 Goal

This experiment tests the basic operation of GNAT before, during, and after an outage, particularly whether GNAT will be able to use the full network capacity (both paths to the destination), avoid the unavailable link during an outage, and limit how many packets are admitted based on per-flow utility function specifications. This experiment also tests the ability of GNAT's admission control to deliver fair throughput between flows of different priorities.

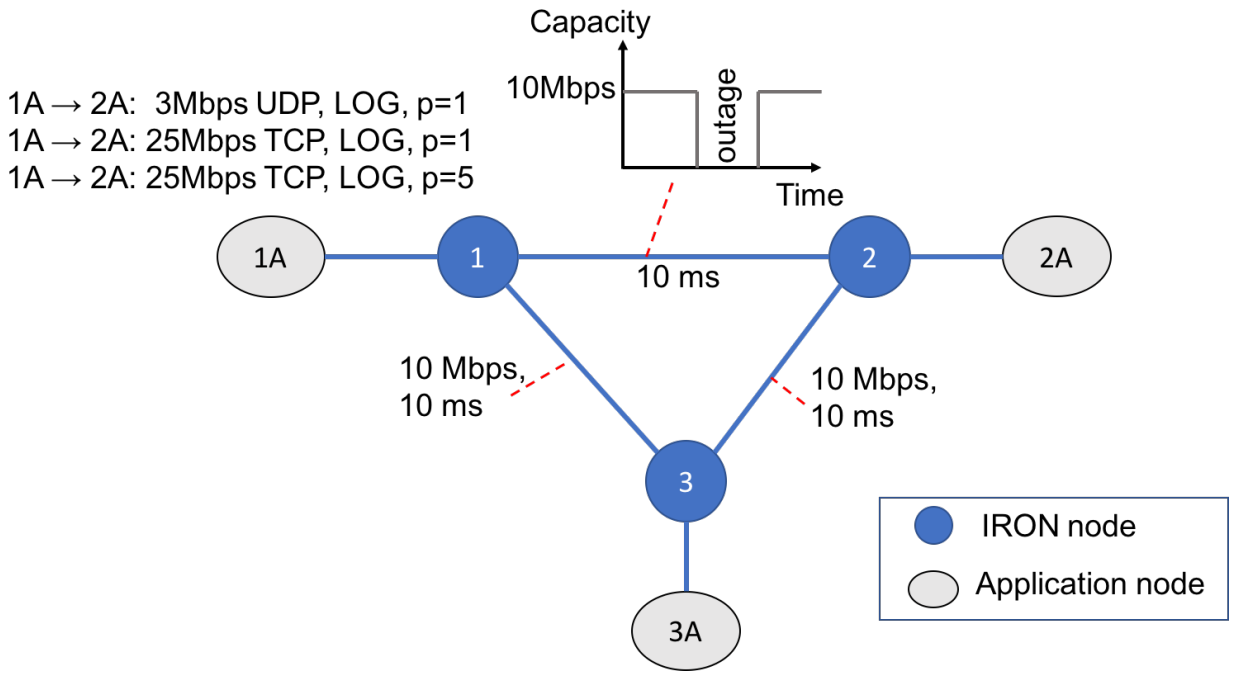
This experiment leverages and tests the following components and features:

- CAT operation,
- UDP and TCP proxies for LOG-utility admission and flow prioritization,
- BPF multi-path forwarding and broken-link avoidance.

4.7.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below, where source enclave 1 sends three flows to destination enclave 2. Each packet may be forwarded over two paths: a direct 10Mbps path (from enclave 1 to enclave 2), and an indirect 10Mbps path (from enclave 1 to enclave 3 to enclave 2). The direct path is intentionally impaired by setting its packet loss rate to 1 around 12 seconds into the experiment, with the impairment subsequently removed around 22 seconds into the experiment.

Source enclave 1 sends one low priority UDP flow with LOG utility and a priority of 1; the UDP flow is capped at a maximum send rate of 3Mbps. It also sends two TCP flows (capped at 25 Mbps each) with a LOG utility function, the first being a low priority flow assigned a priority of 1 and the second being a high priority flow assigned a priority of 5.



4.7.3 Expected Results

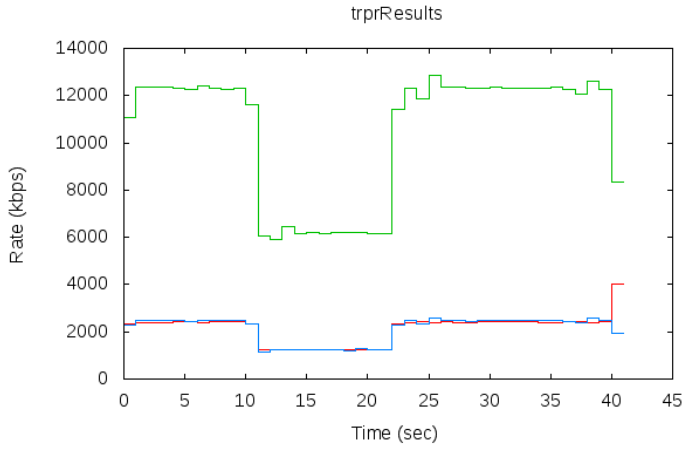
The flows should stabilize to throughput ratios of 5-to-1 between the high and low priority flows, regardless of the transport protocol. With 15% overhead, the aggregate throughput measured at enclave 2 should be about 17Mbps until the direct link is impaired. The 17Mbps goodput should be split ~12Mbps to ~2.5Mbps between flows of priority 5 and 1.

The flow ratios should be maintained throughout the simulation; however the TCP flow can suffer from head-of-the line blocking which means its throughput (as observed by the receiving application) typically varies more than that of the UDP throughput.

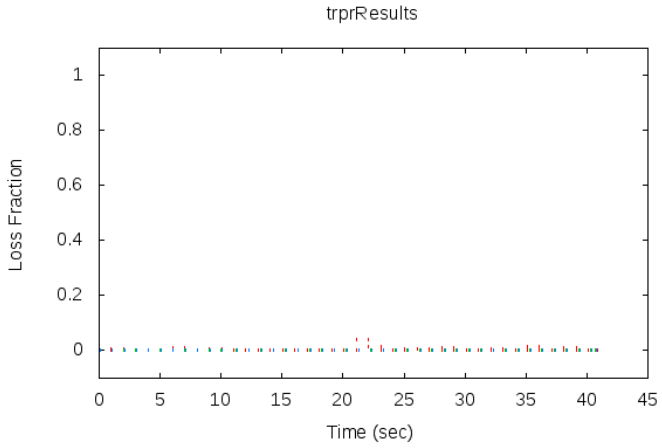
After 20s, the total network capacity is reduced to 10Mbps when the direct link's capacity falls to zero. This will cause some packet loss, requiring more retransmissions and causing more head-of-the-line blocking. This can manifests itself as large changes in instantaneous throughput (i.e., no packets released while waiting for a "hole" to be filled in the TCP flows, followed by a release of a burst of packets once the "hole" is filled). During the impairment, the goodput of the high-priority flow should be around 6Mbps, and 1Mbps for the low-priority flows. Immediately after the start of the outage, the admission controllers in the UDP and TCP proxies should limit admission rates and therefore network loss. Therefore, there should be few packets lost inside the network once admitted.

4.7.4 Observed Results

The observed goodput results are shown below. Flows converge rapidly to a stable throughput: the high-priority flow claims a goodput of a little over 12Mbps and the low-priority UDP and TCP flows are both around 2.5Mbps. During the outage, the rate falls by half to around 6Mbps and 1Mbps respectively. At the start of the outage, we observe dips and spikes as the network adjusts to the missing capacity. GNAT recovers rapidly after the full capacity is restored, back to the expected goodput. Throughout the experiment, the 5-to-1 ratio dictated by flow priorities is preserved.



The observed loss rate for packets admitted onto the network is at most 10% for only the UDP flow and for a short time after the outage begins and while the network deals with the capacity loss.



4.7.5 Conclusions

This experiment demonstrates proper admission onto the network, prioritization, multi-path forwarding, outage avoidance and CAT operation.

4.8 3-node-udp-perf

4.8.1 Goal

This experiment tests the basic operation of GNAT under significant load of multiple UDP flows, particularly whether GNAT will be able to use the full network capacity (all paths to any destination), and limit how many packets are admitted based on per-flow utility function specifications. This experiment also tests the ability of GNAT's admission control to deliver fair throughput between flows of different priorities.

This experiment also highlights the benefits of GNAT's Zombie Latency Reduction to reduce queue delay for all flows.

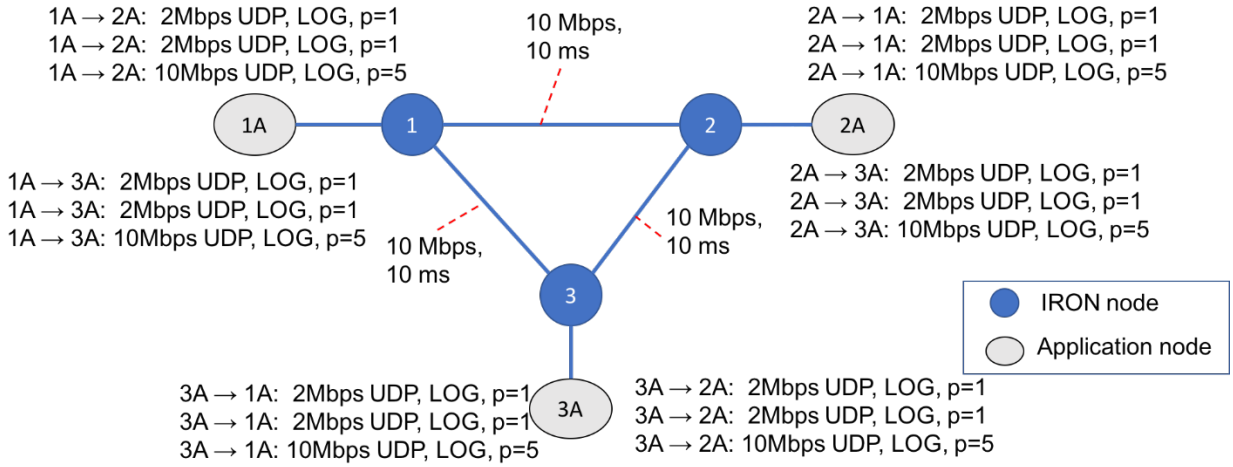
This experiment leverages and tests the following components and features:

- Bidirectional CAT operation,
- UDP proxies for LOG-utility admission and flow prioritization under many flows,
- BPF multi-path forwarding with multiple flow competition, and Zombie Latency Reduction (ZLR).

4.8.2 Topology and Traffic

The experiment runs over a 3-node triangular topology as shown in the figure below, where each enclave sends 3 UDP flows to each other enclave.

The flows between each pair of enclaves include one high priority UDP flow with LOG utility, capped at 10 Mbps, and 2 low priority UDP flows with LOG utility, each capped at 2 Mbps.



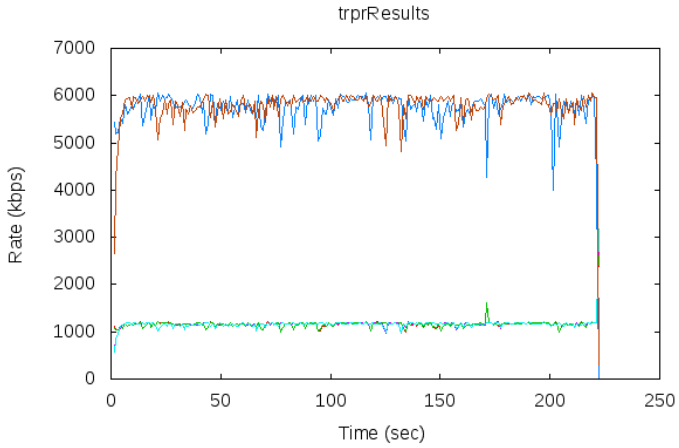
4.8.3 Expected Results

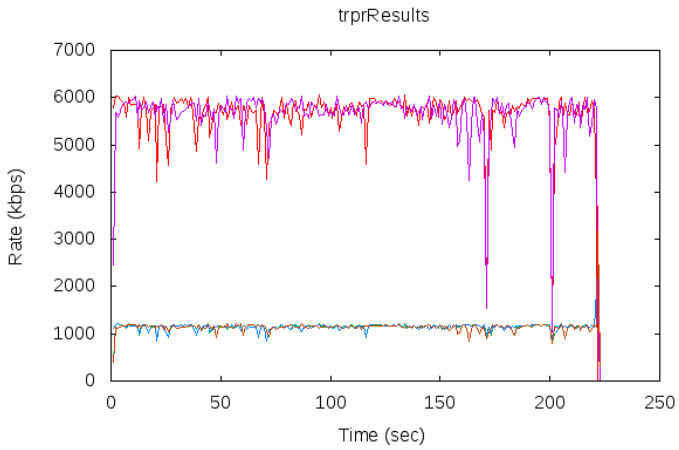
The flows should stabilize to throughput ratios of 5-to-1 between the high and low priority flows. Assuming 20% overhead, each of the high priority flows will have a rate of 5.7Mbps. Each of the low priority flows will have a rate of about 1.1 Mbps. The throughput and flow ratios should be maintained throughout the simulation.

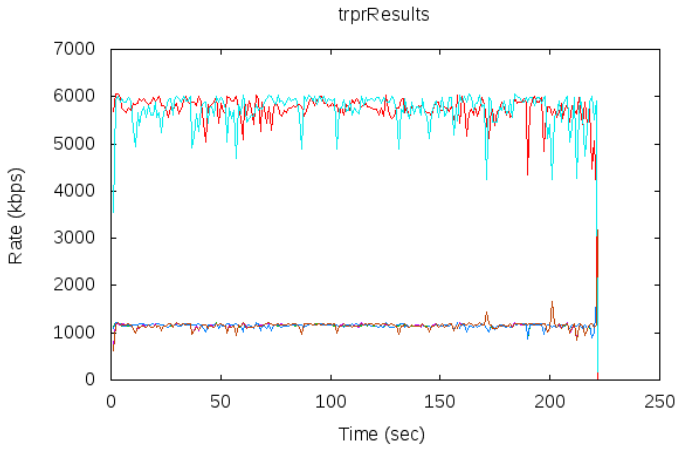
The reported latency should include latency from 3 places: 1. queue delay in the BPF (should be much smaller when ZLR is enabled), 2. queue delay in the CATs (since the default CAT configuration uses both CUBIC and COPA, this is a significant source of latency), 3. propagation delay across the 10 ms links.

4.8.4 Observed Results

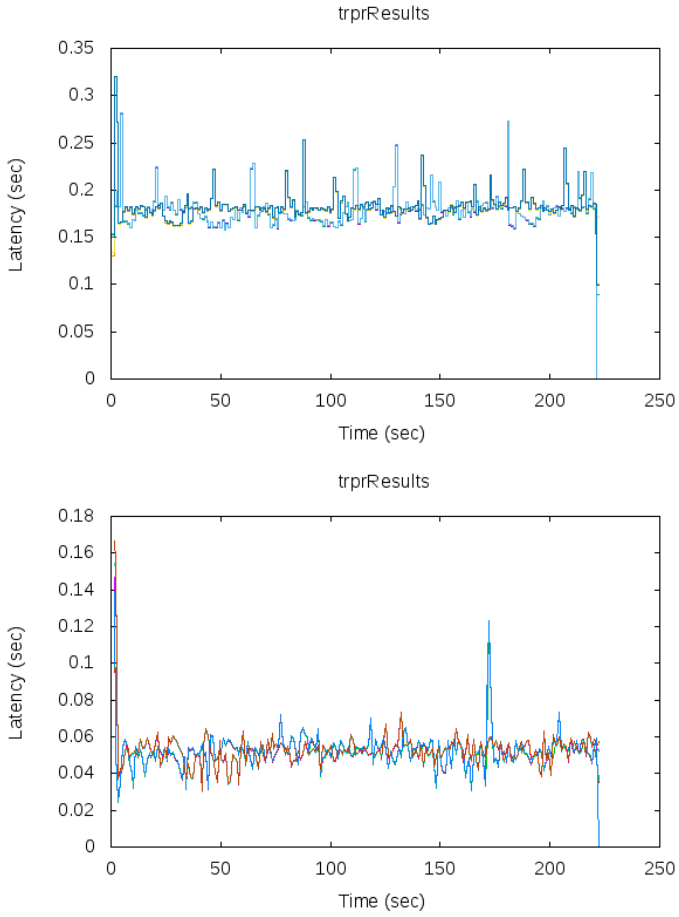
Goodput reported by the receiving application nodes at each enclave matches the expected results: slightly under 6 Mbps for high priority flows and slightly above 1 Mbps for low priority flows. The graphs below are goodputs reported at enclave 1, enclave 2, and enclave 3 (top to bottom).

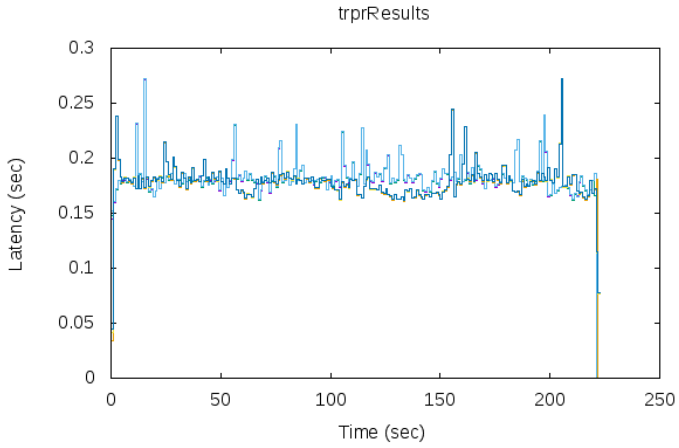


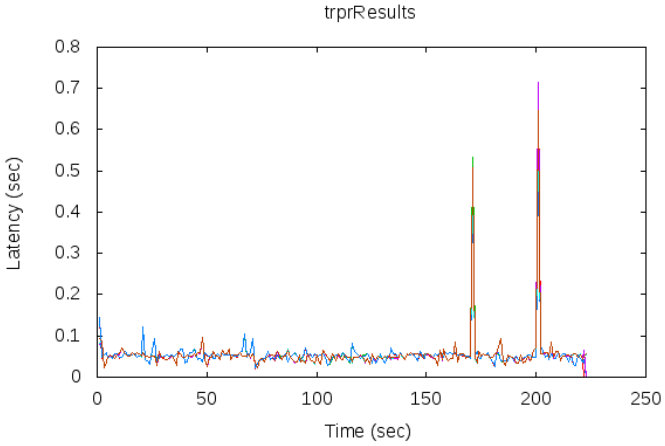


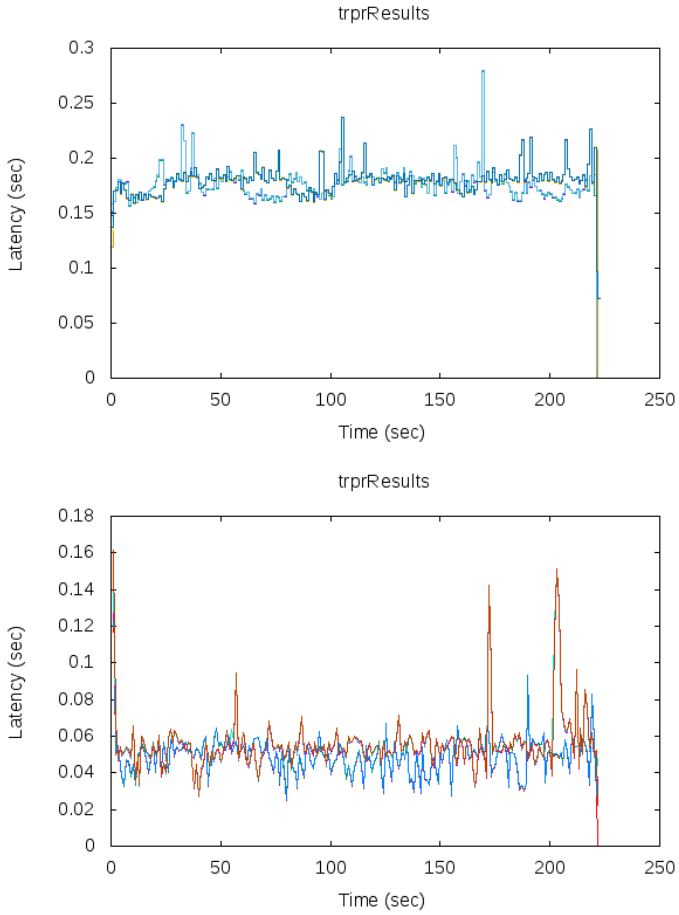


The next set of graphs shows the achieved latency without (left) and with (right) Zombie Latency Reduction enabled, at enclaves 1, 2, and 3 respectively. These show that ZLR does significantly reduce latency.




trprResults
Latency (sec)
Time (sec)
0.3
0.25
0.2
0.15
0.1
0.05
0
0
50
100
150
200
250


trprResults
Latency (sec)
Time (sec)
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
50
100
150
200
250



4.8.5 Conclusions

This experiment demonstrates proper admission onto the network, prioritization, and multi-path forwarding of multiple flows between multiple sources and destinations, and proper bidirectional CAT operation. It also highlights to significant benefits of Zombie Latency Reduction.

4.9 3-node-tcp-perf

4.9.1 Goal

This experiment tests the basic operation of GNAT under significant load of multiple TCP flows, particularly whether GNAT will be able to use the full network capacity (both paths to any destination), and limit how many packets are admitted based on per-flow utility function specifications. This experiment also tests the ability of GNAT's admission control to deliver fair throughput between flows of different priorities.

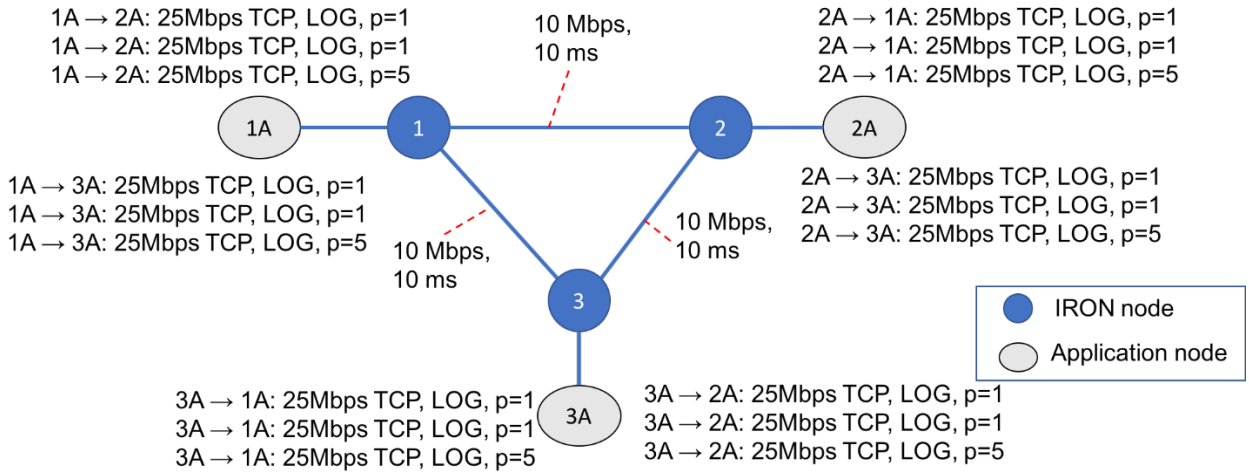
This experiment leverages and tests the following components and features:

- Bidirectional CAT operation,
- TCP proxies for LOG-utility admission and flow prioritization under many flows,
- BPF multi-path forwarding with multiple flow competition.

4.9.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below, where each enclave sends 3 TCP flows to each other enclave.

The flows between each pair of enclaves include one high priority and 2 low priority TCP flows with LOG utility, each capped at 25 Mbps.



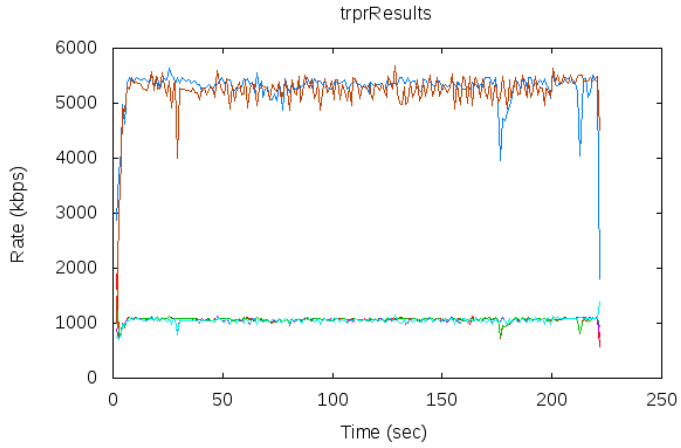
4.9.3 Expected Results

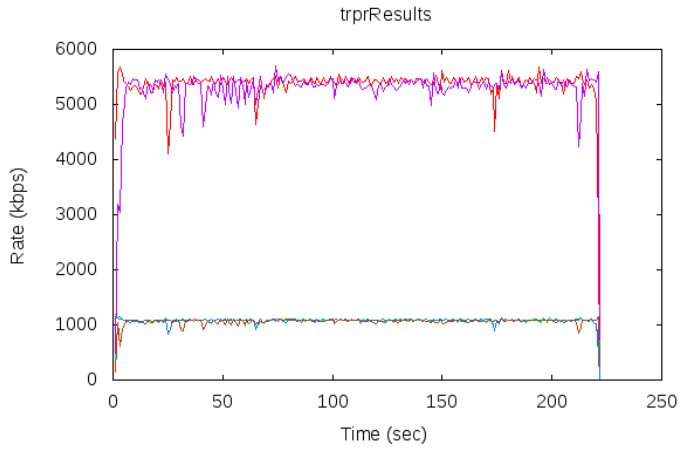
The flows should stabilize to throughput ratios of 5-to-1 between the high and low priority flows. Assuming 20% overhead, each of the high priority flows will have a rate of 5.7Kbps. Each of the low priority flows will have a rate of about 1.1 Kbps.

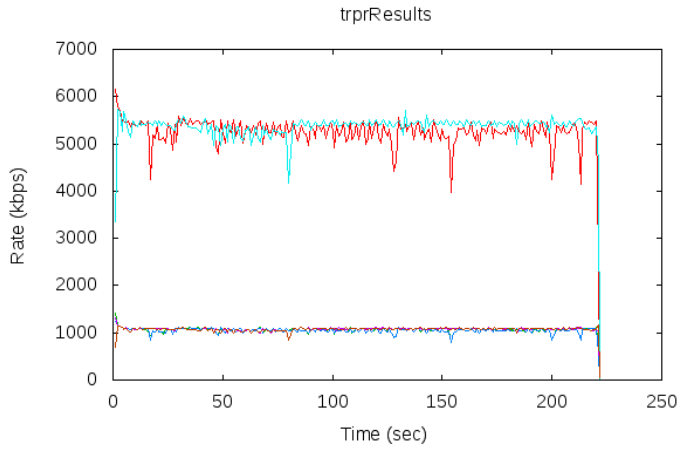
The throughput and flow ratios should be maintained throughout the simulation.

4.9.4 Observed Results

The three graphs below show the goodput observed at the application nodes in enclaves 1, 2, and 3, respectively. The high priority flows achieve goodput between 5 and 6 Mbps, and the low priority flows achieve goodput of around 1 Mbps.







4.9.5 Conclusions

This experiment demonstrates proper admission onto the network, prioritization, and multi-path forwarding of multiple flows between multiple sources and destinations, and proper bidirectional CAT operation.

4.10 3-node-perf

4.10.1 Goal

This experiment slightly stresses the system with many flows of both TCP and UDP traffic and a relatively high link capacity, running for a long enough time to show stability past all of the default configurations for cleanup periods.

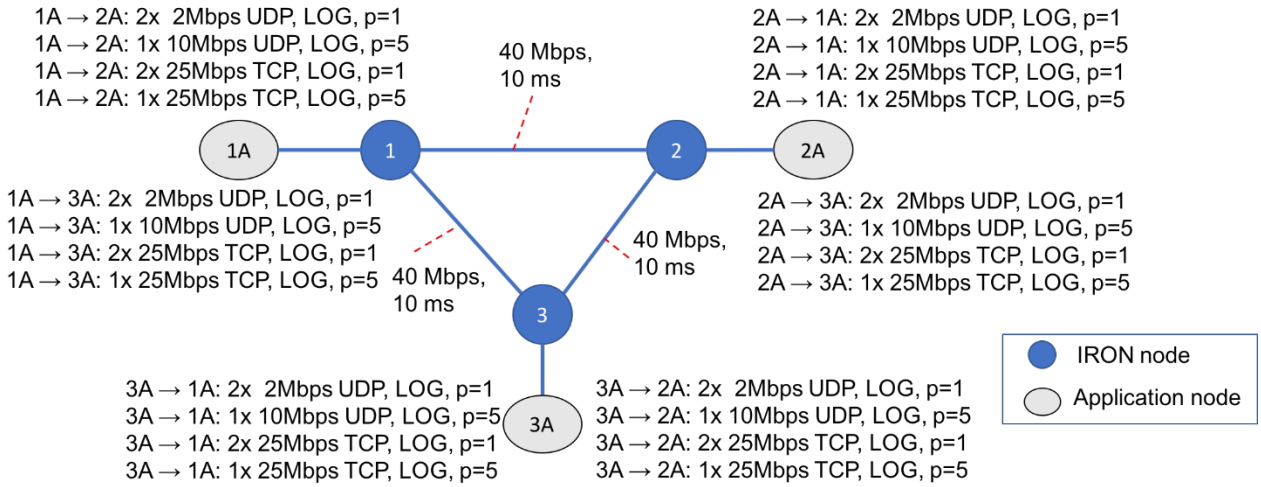
Although GNAT can support a larger number of flows, higher bandwidth per single flow, or higher aggregate bandwidth, this pushes the system in all three directions at the same time.

This experiment leverages and tests the following components and features:

- CAT operation at 30Mbps bandwidth,
- UDP and TCP proxies: prioritization (with LOG admission) and performance with 30 flows per proxy per node,
- BPF multi-path high-rate forwarding.

4.10.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below, where each application node sources traffic to each other application node. Each enclave sends 40 low priority (p=1) flows (20 UDP and 20 TCP) and 20 high priority (p=5) flows (10 UDP and 10 TCP) to each other enclave. All source-destination pairs are equal, so all traffic should (logically) flow only along its direct one-hop path. Each link has 30Mbps bandwidth and a 10ms propagation delay. All flows have offered loads high enough to never be a bottleneck.

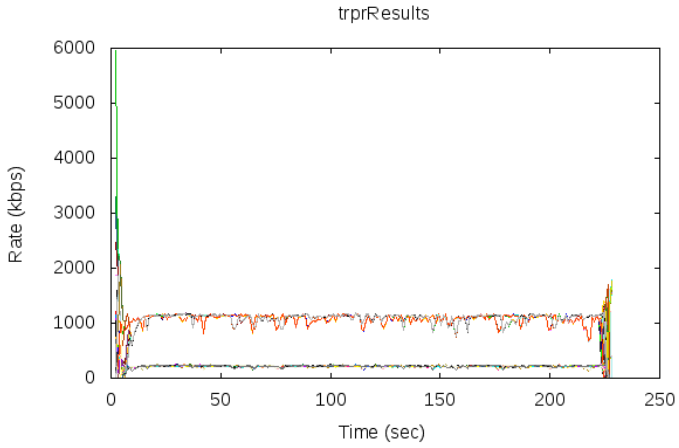


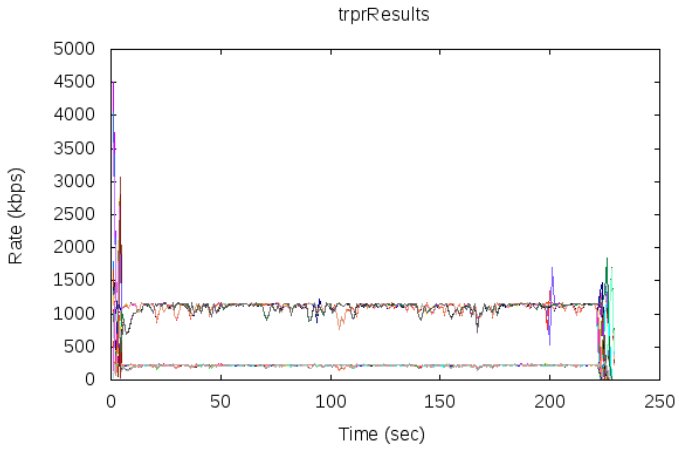
4.10.3 Expected Results

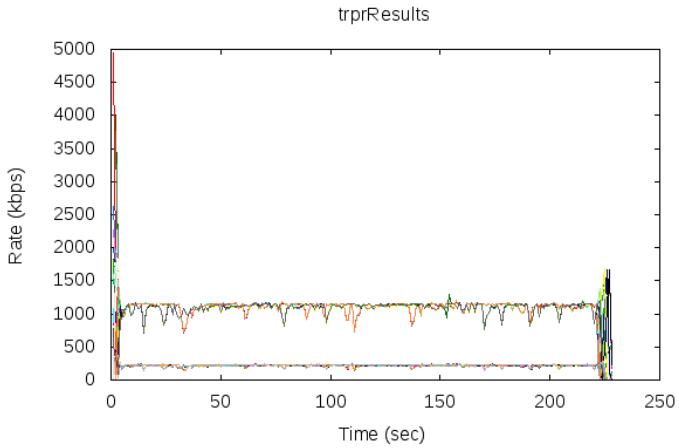
The flows should stabilize to throughput ratios of 5-to-1 between the high and low priority flows, regardless of the transport protocol. With 15% overhead, this comes out to approximately 1140kbps per high priority flow and 230kbps per low priority flow.

4.10.4 Observed Results

The three graphs below show the achieved goodput as observed at the applications nodes in enclaves 1, 2, and 3, respectively. The high priority flows get more than 1000kbps, and the low priority flows get more than 200kbps.







4.10.5 Conclusions

GNAT holds up well, providing consistent prioritized load-balancing performance under the concurrent stressors.

4.11 3-node-tcp-max

4.11.1 Goal

This experiment tests the basic ability of GNAT to send high rate flows of TCP traffic, particularly whether GNAT will be able to use the full network capacity (both paths to the destination), send bidirectional flows, and limit how many packets are admitted based on per-flow utility function specifications. This experiment also tests the ability of GNAT's admission control to deliver fair throughput between flows of different priorities at very high rates.

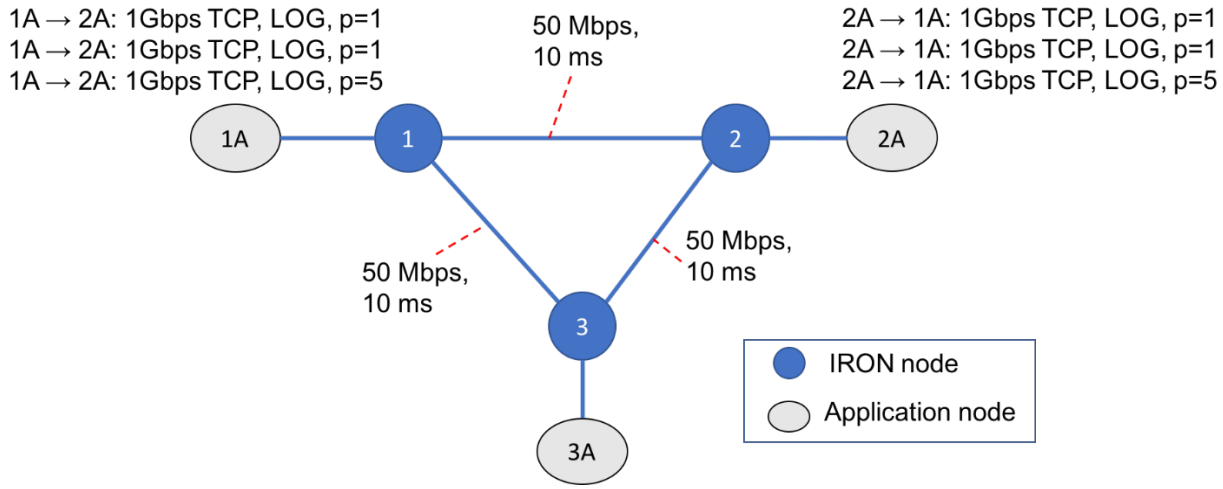
This experiment leverages and tests the following components and features:

- High-capacity bidirectional CAT operation,
- TCP proxies for LOG-utility admission and flow prioritization with a small number of high rate flows,
- BPF multi-path forwarding.

4.11.2 Topology and Traffic

The experiment runs over a 3-node triangular topology as shown in the figure below, where source enclave 1 sends three TCP flows to destination enclave 2, and vice-versa. Each packet may be forwarded over two paths: a direct 50Mbps path (from enclave 1 to enclave 2), and an indirect 50Mbps path (from enclave 1 to enclave 3 to enclave 2).

Source enclave 1 sends one high priority TCP flow with LOG utility (priority 5) and two low priority TCP flows with LOG utility (priority 1). All flows are capped at a maximum send rate of 1Gbps, far exceeding the total network capacity of 100Mbps. (Note, however: since the network cards on the test machines are only 1 Gbps cards, a 1 Gbps send rate is the equivalent of 1 Gbps divided equally across all flows, since that's the maximum rate traffic could leave the source.)



4.11.3 Expected Results

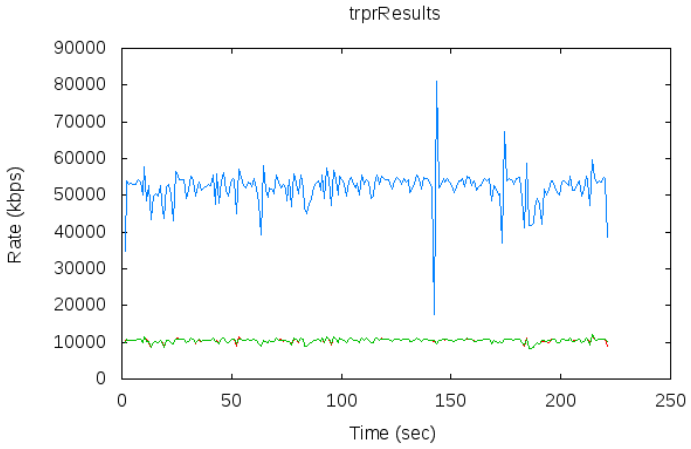
The flows should stabilize to throughput ratios of 5-to-1 between the high and low priority flows. With 15% overhead, the aggregate throughput measured at enclave 2 should be about 85Mbps,

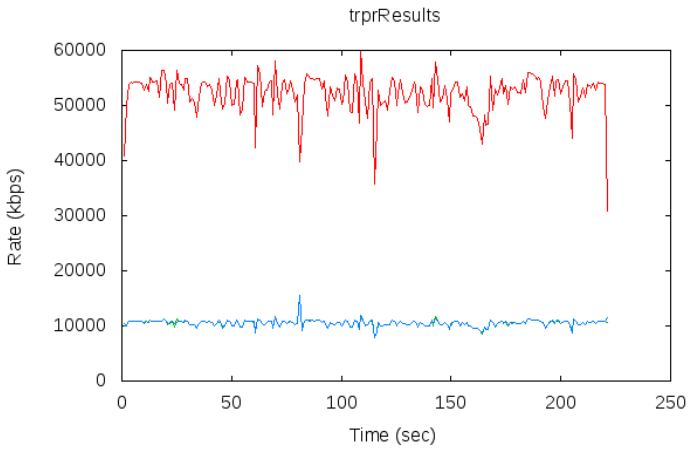
which should be split ~60Mbps to ~12Mbps between flows of priority 5 and 1.

The flow ratios should be maintained throughout the simulation, however the TCP flow can suffer from head-of-the line blocking which means its throughput (as observed by the receiving application) may visibly vary and dip. In a real application, these dips would most likely be invisible to a user.

4.11.4 Observed Results

The two graphs below show the goodput measured at enclave 1 and enclave 2, respectively. The high priority TCP goodput is slightly lower than expected.





4.11.5 Conclusions

This experiment demonstrates proper admission onto the network, prioritization, multi-path forwarding, bidirectional CAT operation under high traffic loads, and a small number of high-rate flows processed by the TCP proxy.

4.12 3-node-short-tcp

4.12.1 Goal

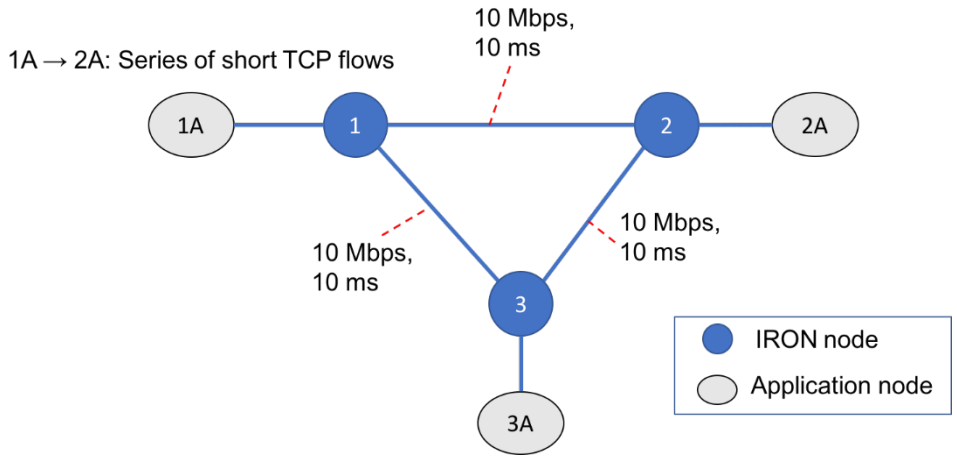
This experiment tests the operation of GNAT with the establishment and handling of short-lived TCP flows. These flows are short transfers of around 3KB and simulate a series of HTTP GETs. There is overhead associated with setting up and tearing down TCP connections which becomes more noticeable with many short flows. This experiment was designed to gauge GNAT's performance under these conditions.

This experiment leverages and tests the following components and features:

- CAT operation
- TCP proxy with LOG-utility admission and establishing/destroying short end-to-end connections.
- BPF forwarding

4.12.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below. The application node in enclave 1 sends a series of short TCP flows to the application node in enclave 2. Five flows of 3000 bytes each are sourced approximately once every 10 seconds, with some randomization of the start times. The flows have identical LOG utility and do not fill the links.



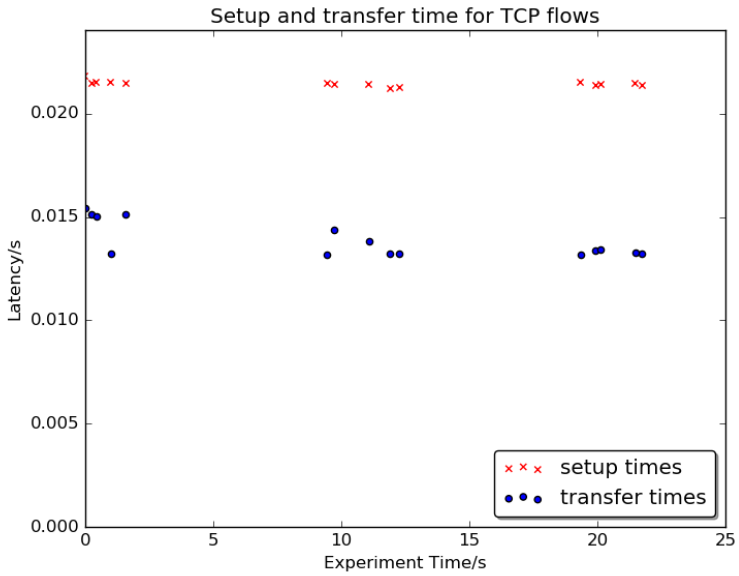
4.12.3 Expected Results

The goodput for this experiment is uninteresting, since the flows are small enough that network capacity is not a bottleneck. There are no bandwidth or latency constraints, so the flows are all identical.

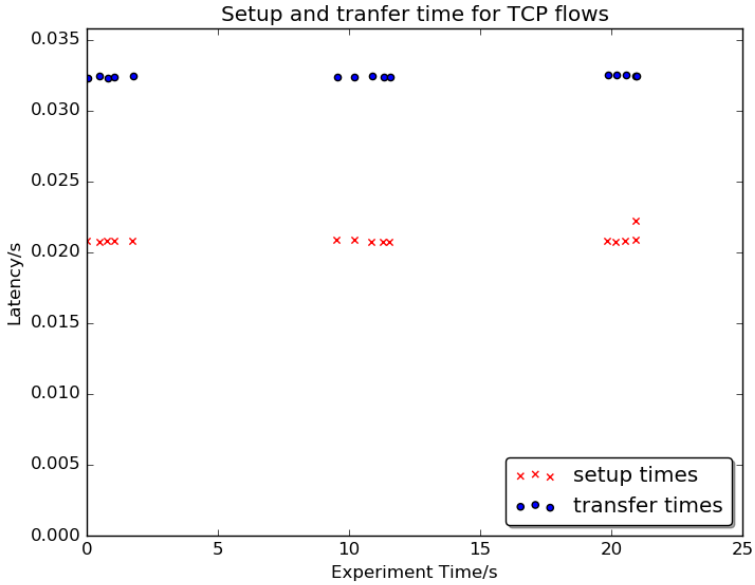
The interesting results are the setup time and total transfer time for the flows. These should be small and identical across flows.

4.12.4 Observed Results

The plot below shows the setup times and the total transfer times of each of the short flows. We see that this is about the same for each of the flows.



The plot below shows the setup and total transfer times without GNAT. The setup times is about the same but the transfer time is more than double what it is with GNAT. This is due to slow-start in TCP: the baseline sends 2 packets, and must wait for an ACK before it sends the third. The round trip time is around 20ms. With GNAT, however, the TCP proxy acts like a Performance Enhancing Proxy (PIP) by buffering data and acknowledging packets almost immediately. The application does not have to wait a round-trip-time before sending additional packets.



4.12.5 Conclusions

This experiment demonstrates that the TCP proxy can properly handle short TCP flows, and in-fact, does better than TCP directly between the applications. It also shows that the BPF does not starve small flows.

4.13 3-node-tcp-large-num-flows

4.13.1 Goal

This experiment tests the basic ability of GNAT to simultaneously handle a large number of TCP flows with the same source and destination, particularly whether GNAT will be able to use the full network capacity (both paths to the destination), send bidirectional flows, and limit how many packets are admitted based on per-flow utility function specifications. This experiment also tests the ability of GNAT's admission control to deliver fair throughput between a large number of flows of different priorities.

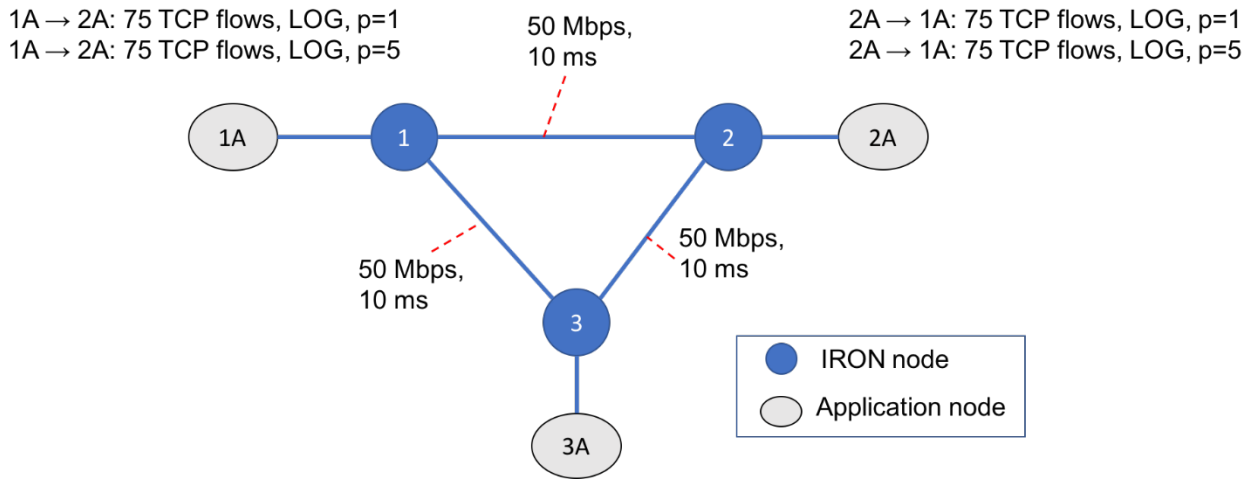
This experiment leverages and tests the following components and features:

- High-capacity bidirectional CAT operation,
- TCP proxies for LOG-utility admission and flow prioritization with a large number of flows,
- BPF multi-path forwarding.

4.13.2 Topology and Traffic

The experiment runs over a 3-node triangular topology as shown in the figure below, where source enclave 1 sends 150 TCP flows (75 with priority 1 and 75 with priority 5) to destination enclave 2, and vice-versa. Each packet may be forwarded over two paths: a direct 50Mbps path (from enclave 1 to enclave 2), and an indirect 50Mbps path (from enclave 1 to enclave 3 to enclave 2).

All flows are capped at a maximum send rate of 100Mbps, so that the set of all flows at full rate would far exceed the total network capacity of 100Mbps. (Note, however: since the network cards on the test machines are only 1 Gbps cards, an aggregate 15Gbps send rate is actually the equivalent of 1 Gbps divided equally across all flows, since that's the maximum rate traffic could leave the source.)



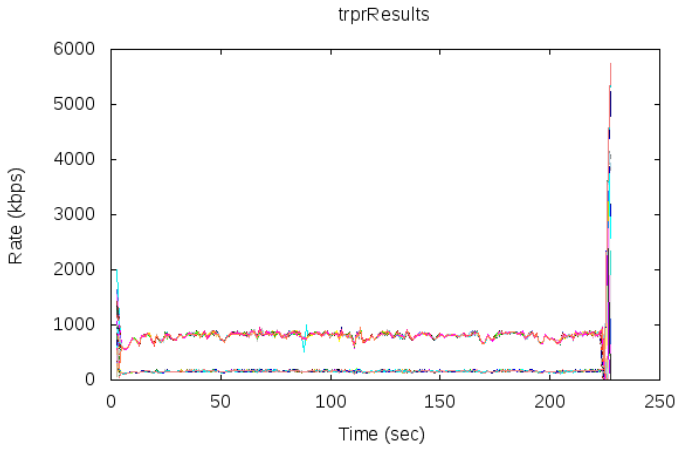
4.13.3 Expected Results

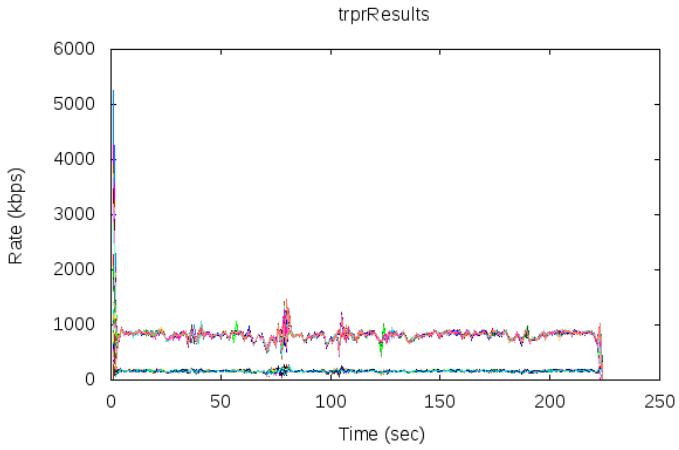
The flows should stabilize to throughput ratios of 5-to-1 between the high and low priority flows filling the entire 100Mbps capacity between each pair. With 20% overhead, that leaves 152kbps for each low priority flow and 962kbps for each high priority flow.

The flow ratios should be maintained throughout the simulation, however the TCP flow can suffer from head-of-the line blocking which means its throughput (as observed by the receiving application) may visibly vary and dip. In a real application, these dips would most likely be invisible to a user.

4.13.4 Observed Results

The two graphs below show the goodput measured at enclave 1 and enclave 2, respectively. There are large goodput spikes during the periods of stabilization at the start and end, but otherwise the goodput distribution is correctly 5-to-1.





4.13.5 Conclusions

GNAT is able to maintain consistent, prioritized load balancing with larger numbers of flows.

4.14 3-node-dynamics

4.14.1 Goal

This experiment tests GNAT's behavior under system dynamics, particularly whether GNAT will be able to respond to changes in flow priorities, latency, link capacity, etc.

This experiment leverages and tests the following components and features:

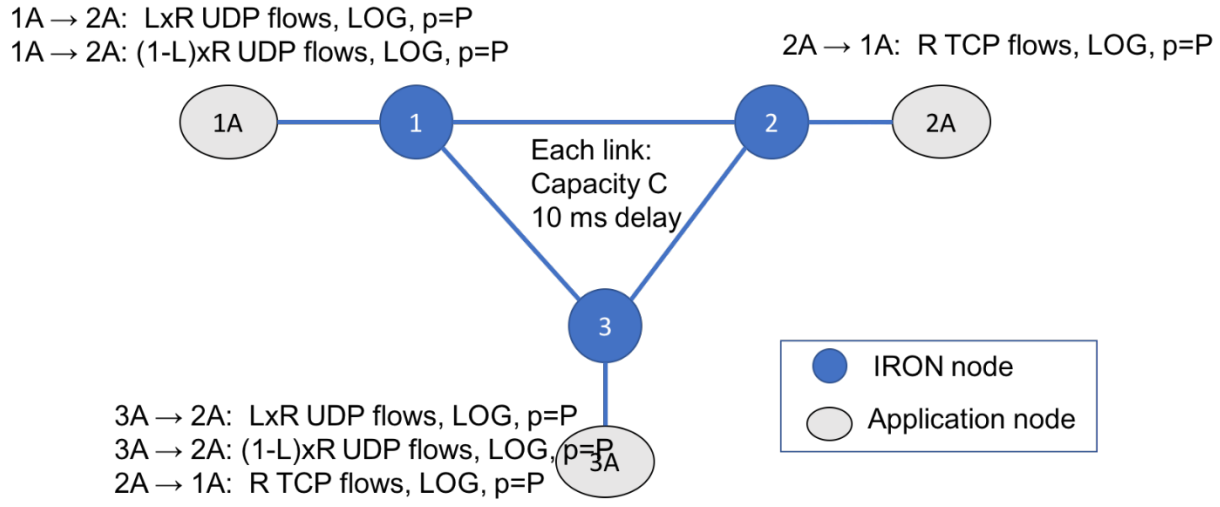
- CAT operation,
- UDP and TCP proxies,
- BPF multi-path and latency-aware forwarding.

4.14.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below, where the application nodes in enclaves 1 and 3 send UDP traffic to the application node in enclave 2, and the application nodes in enclaves 1 and 3 send TCP traffic to the application node in enclave 1. Each combination of flows is enough to fill the network independently, so the offered load is not a constraint. A portion (L) of the UDP flows are latency sensitive. The number and priority of the flows changes throughout the experiment.

All link capacities (C) are identical at any moment, but they vary as the experiment progresses, and they all have a 10ms delay throughout the experiment.

The experiment is divided up into 25 second intervals. The specific flows and network parameters at each interval are detailed in the table below. The specified number/type of flows always runs between each pair, as described above. For instance, during the first interval, there are 10 UDP flows from enclave 1 to enclave 2, 10 UDP flows from enclave 3 to enclave 2, 10 TCP flows from enclave 2 to enclave 1, and 10 TCP flows from enclave 3 enclave 1. One fifth of each group of UDP flows are flagged as latency sensitive (L = 0.2). For instance, interval 1 has eight latency-insensitive UDP flows from enclave 1 to enclave 2 and two latency sensitive flows from enclave 1 to enclave 2.



0,1, and 2 are application nodes. 3, 4, and 5 are GNAT nodes.

Interval	Seconds	Num Flows per Src / Dst Pair (R in the diagram above)	Capacity per Link (C in the diagram)	Flow Priority (P in the diagram)	Expected Goodput per Flow
1	10-35	10	10Mbps	1	Slightly less than 1Mbps
2	35-60	10	10Mbps	10	Slightly less than 1Mbps
3	60-85	10	10Mbps	1	Slightly less than 1Mbps
4	85-110	10	10Mbps	20	Slightly less than 1Mbps
5	110-135	10	10Mbps	1	Slightly less than 1Mbps
6	135-660	5	10Mbps	1	Slightly less than 2Mbps
7	160-185	20	10Mbps	1	Slightly less than 500kbps
8	185-210	10	10Mbps	1	Slightly less than 1Mbps
9	210-235	10	5Mbps	1	Slightly less than 500kbps
10	235-260	10	20Mbps	1	Slightly less than 2Mbps
11	260-285	10	10Mbps	1	Slightly less than 1Mbps
12	285-310	5	5Mbps	1	Slightly less than 1Mbps
13	310-335	20	20Mbps	1	Slightly less than 1Mbps
14	335-360	10	10Mbps	1	Slightly less than 1Mbps
15	360-385	5	20Mbs	1	Slightly less than 4Mbps
16	385-410	20	5Mbps	1	Slightly less than 250kbps
17	410-435	10	10Mbps	1	Slightly less than 1Mbps

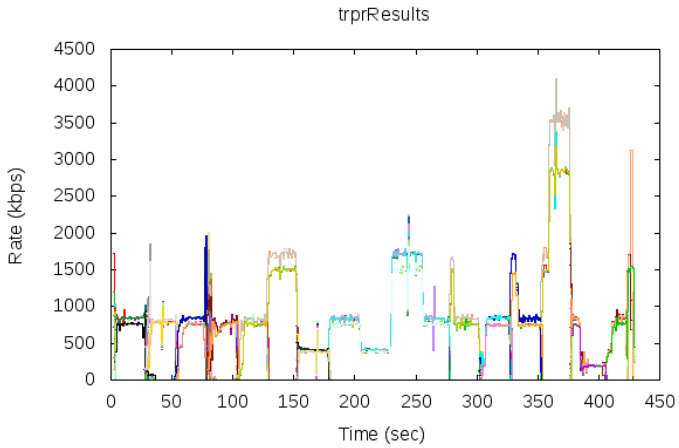
4.14.3 Expected Results

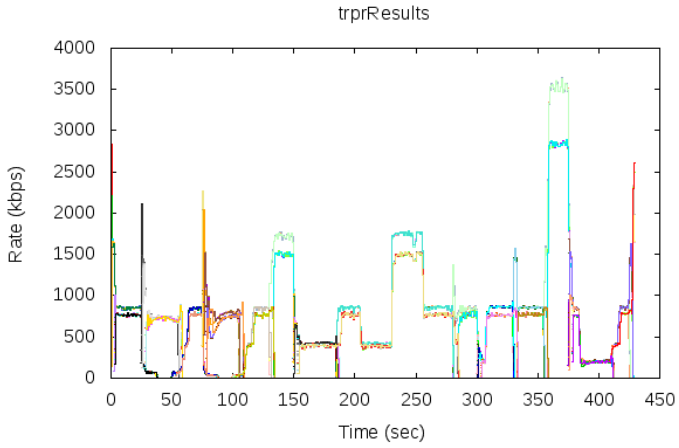
The expected results are described in the table above.

4.14.4 Observed Results

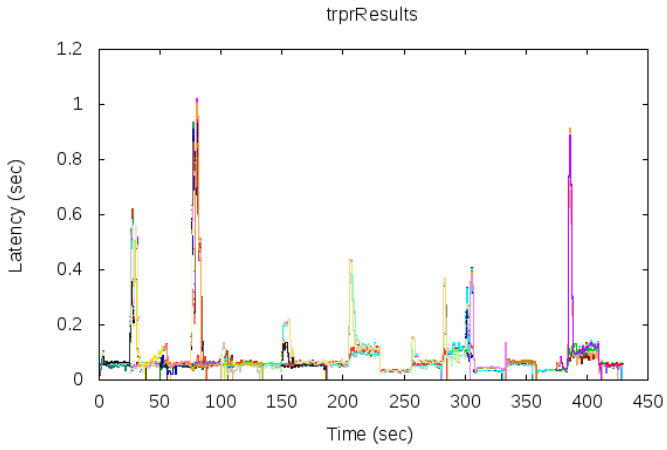
The observed goodput (see figures below, enclave 1 on left, enclave 2 on right) is almost as expected. The flow rates converge rapidly (at worst, within a few seconds).

Prioritization between flows is generally correct (all flows get the same goodput). However, in the intervals with higher per-flow throughput, prioritization is not quite right.





Latency of TCP flows (observed at destination enclave 1) cannot be easily graphed because of head-of-the-line blocking. However, the observed latency of the UDP flows is included here. Convergence periods show up as latency spikes, during which queue depths rapidly increase and zombie latency reduction cannot react. Latency should otherwise be fairly constant.



4.14.5 Conclusions

Overall, GNAT responds rapidly to network and flow dynamics, but there is some unfairness at high rates.

4.15 y3_edge

4.15.1 Goal

This experiment demonstrates GNAT's ability to handle a bottleneck link that is multiple hops downstream of the source of traffic, including fairness between multiple flows and reaction time when traffic patterns change (even though the bottleneck is downstream).

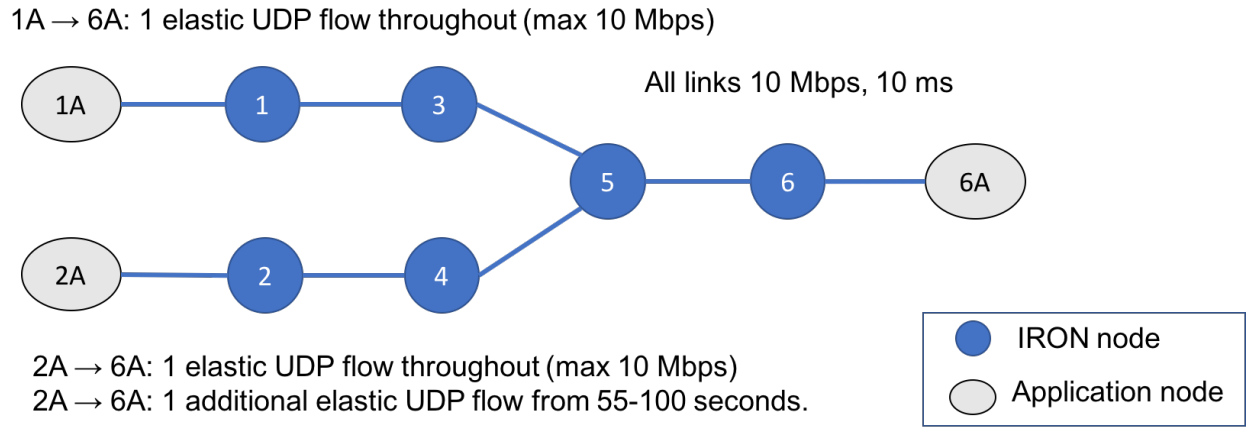
This experiment leverages and tests the following components and features:

- UDP proxy for prioritizing flows,
- BPF fair forwarding of flows.

This experiment also showcases the latency benefits (to all traffic, not just latency-sensitive traffic) from using zombie latency reduction. By increasing the link delays, this experiment can be used to showcase queue depth oscillation reduction (see Queue Depth Oscillation Reduction).

4.15.2 Topology and Conditions

The experiment runs over a 6-enclave Y-shaped topology as shown in the figure below. Each of enclaves 1 and 2 sends a single elastic UDP flow (at priority 1) to enclave 6 throughout the experiment. For a short period of time in the middle of the experiment, enclave 2 sends an additional elastic UDP flow to enclave 6.



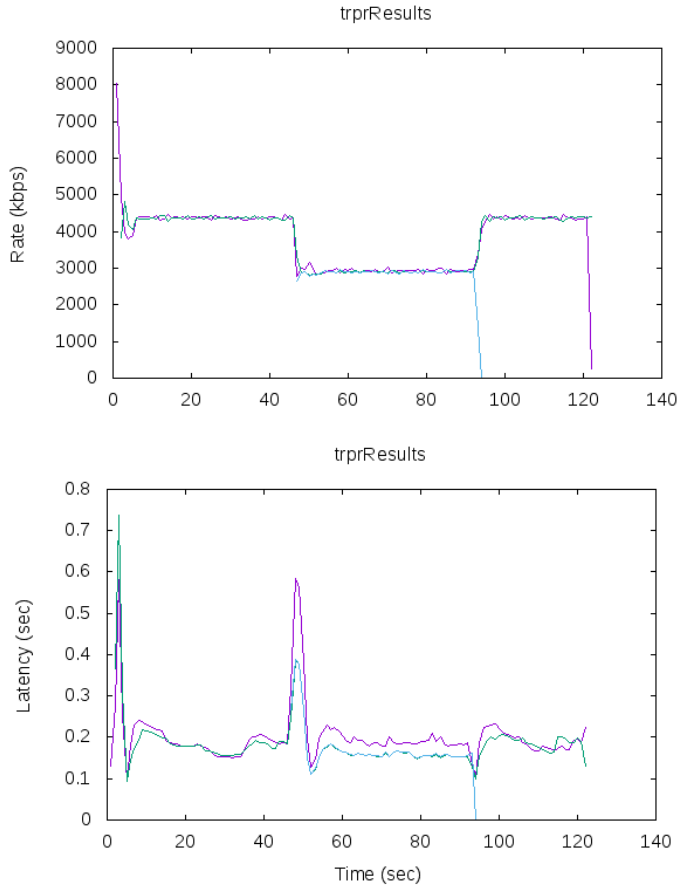
4.15.3 Expected Results

During the first and last thirds of the test, we expect the bottleneck capacity (10 Mbps minus overhead, or about 18.5 Mbps) to be evenly divided across two flows. This leaves 4.25 Mbps per flow. During the middle portion of the test, we expect the same capacity to be evenly divided across 3 flows, leaving 2.83 Mbps per flow.

Because ZLR reactively maintains a constant number of non-zombie packets on the queues, we expect latency to remain fairly constant across all flows.

4.15.4 Observed Results

Goodput and latency per flow, observed at enclave 6.



4.15.5 Conclusions

This experiment demonstrates the ability of GNAT to achieve fair prioritization and quick reaction times with remote bottlenecks.

4.16 12enclave_concurrent_mixed

4.16.1 Goal

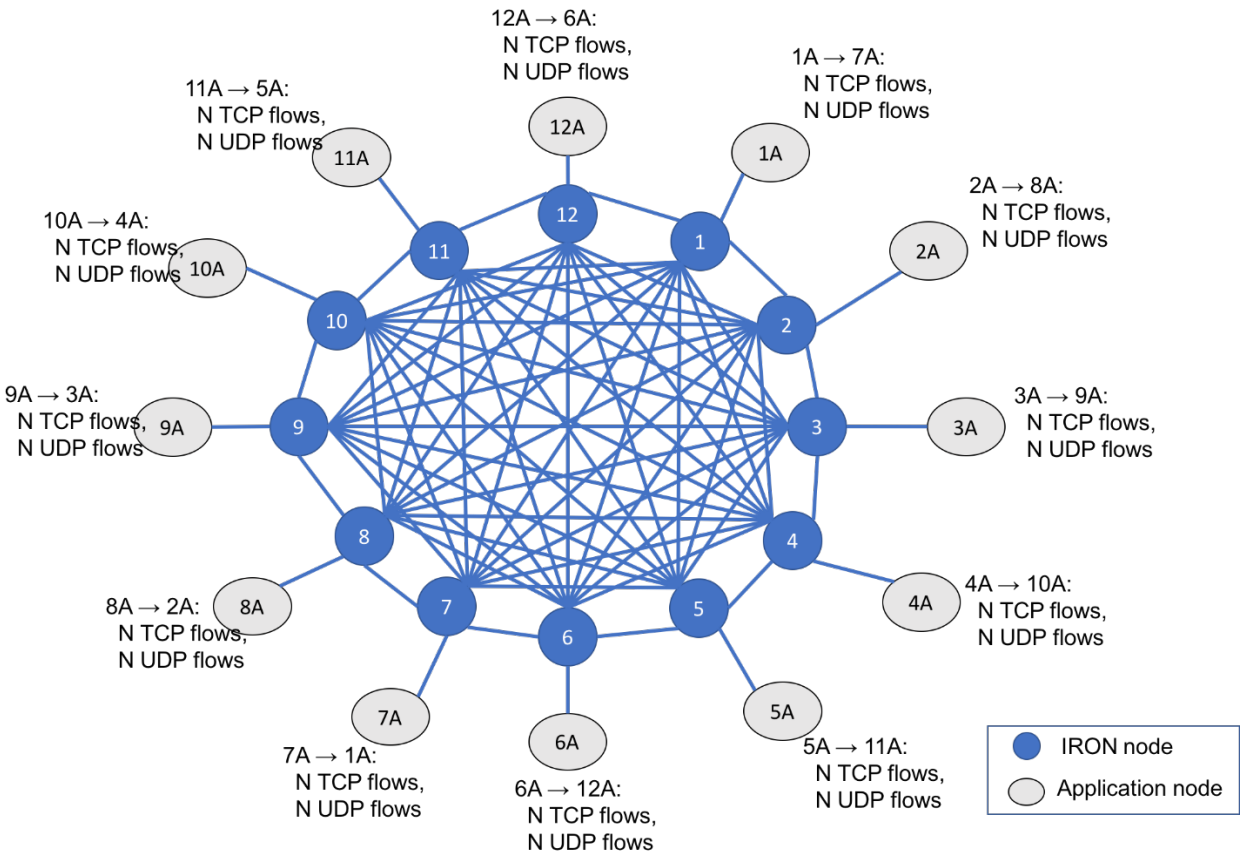
This experiment tests GNAT's ability to support 12 enclaves in a full mesh topology with many concurrent UDP and TCP flows.

This experiment leverages and tests the following components and features:

- CAT operation,
- UDP and TCP proxies,
- BPF multi-path forwarding,
- System scalability to 12 enclaves

4.16.2 Topology and Conditions

The experiment uses a 12 node full mesh topology as shown in the figure below. Each application node sources traffic destined to the application node opposite it if we view the 12 enclaves as sitting in a ring. For example, enclave 1 sends to enclave 7, enclave 2 sends to enclave 8, etc.



Each link has a 10Mbps capacity, giving each enclave an aggregate capacity of 110Mbps (10Mbps to each other node). If all nodes are sending at a maximum rate, the maximum aggregate rate that can be sourced by each node is 60Mbps (in the simplest maximum flow solution, the direct link is not shared with any other pair, while every other link is shared with exactly one other pair),

which is effectively 48Mbps after accounting for overhead. To find the maximum number of concurrent flows supported, we increase the number of flows per source-destination pair incrementally while maintaining the aggregate offered load per source at 50Mbps. We also modify the scaling parameter K to roughly maintain the same queue depths (this is for the same of run-to-run fairness: we did not attempt to find a common K value for this experiment). The table below shows the values for the series of experiment runs.

N (number of flows per transport protocol)	offered load per flow	K	Expected goodput per flow
400	62.5Kbps	2.5e10	60Kbps
200	125Kbps	5e10	120Kbps
100	250Kbps	1e11	240Kbps
50	500Kbps	2e11	480Kbps
25	1Mbps	4e11	960Kbps
12 TCP, 13 UDP	2Mbps	8e11	1920Kbps

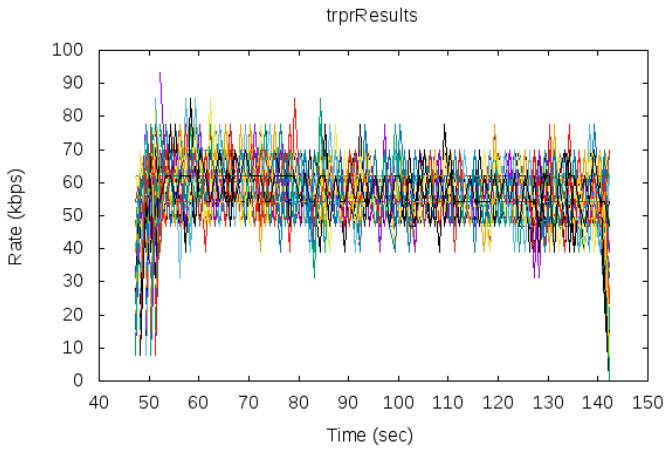
4.16.3 Expected Results

The expected results are described in the table above.

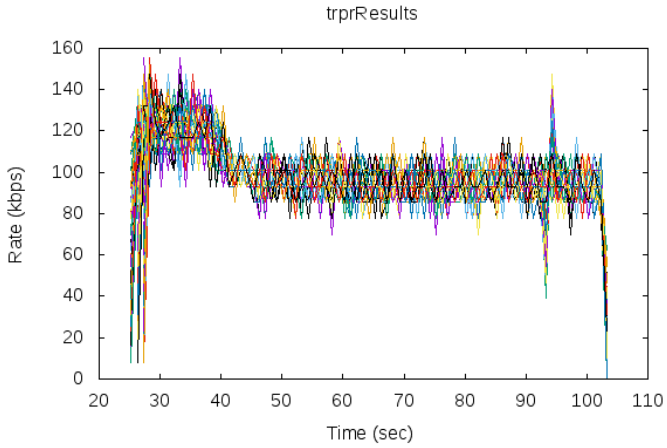
4.16.4 Observed Results

The observed goodput at enclave 1 for each of the 6 runs is shown below. The flows all achieve approximately equal goodput, and the goodput matches the expected values in the table above.

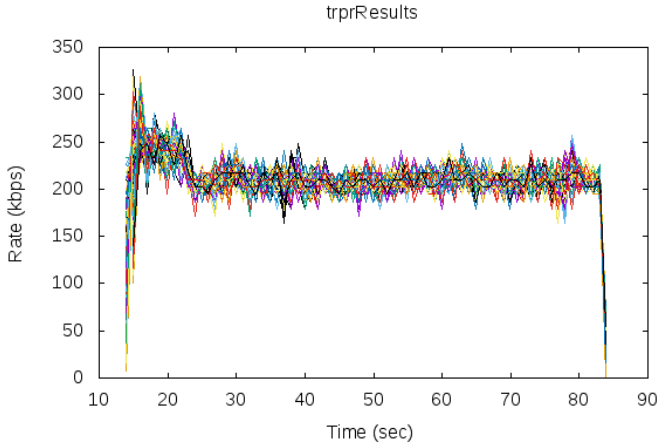
800 total flows:



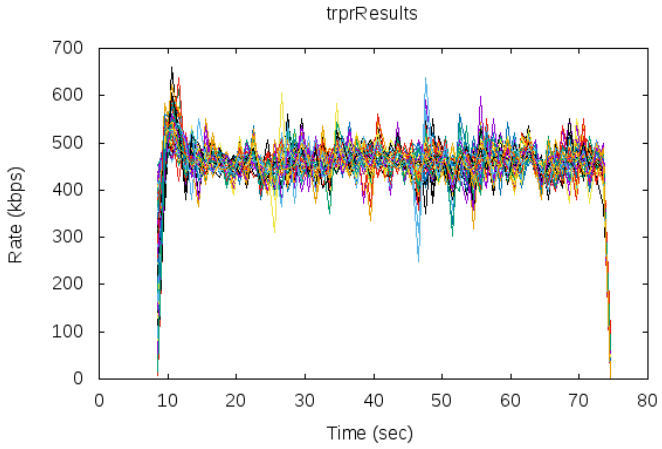
400 total flows:



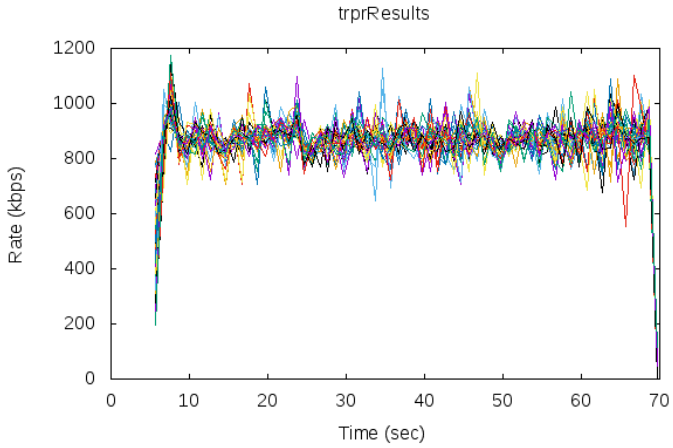
200 total flows:



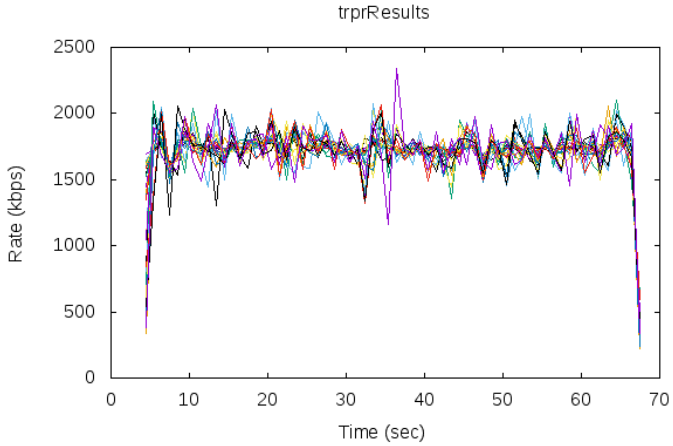
100 total flows:



50 total flows:



25 total flows:



4.17 3-node-system-lat

4.17.1 Goal

This experiment tests the basic operation of GNAT for latency-constrained traffic, particularly whether GNAT will be able to use the full network capacity (both paths to the destination), and send latency-sensitive traffic onto paths with lower delay, leaving the slower path for latency-insensitive traffic. It also evaluates how priorities and latency constraints interact.

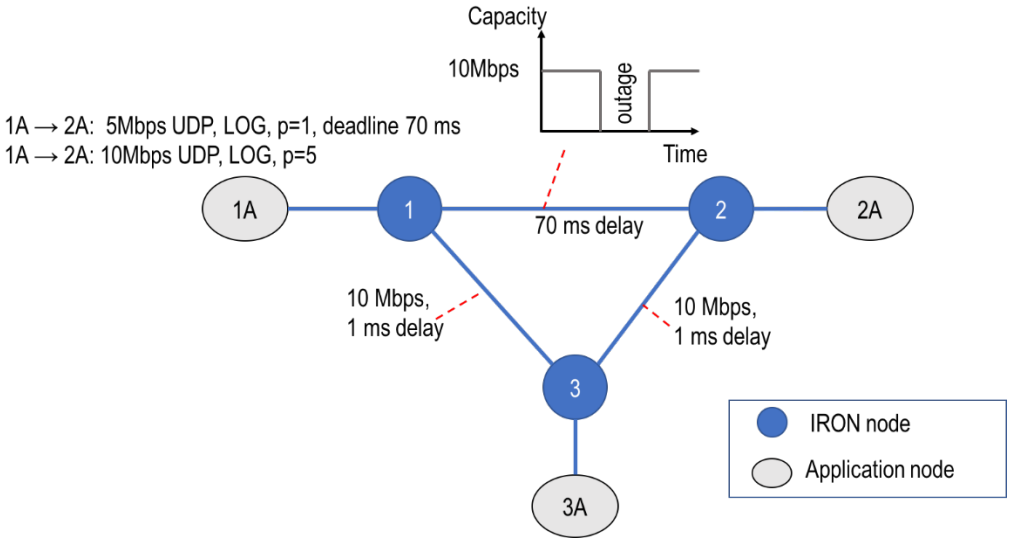
This experiment leverages and tests the following components and features:

- CAT operation for latency-sensitive traffic,
- UDP proxies for LOG-utility admission,
- BPF multi-path latency-aware forwarding, including during impairments,
- GNAT's ability to maintain a low queuing delay.

4.17.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below, where the application node in enclave 1 sends two flows to the application node in enclave 2. There are two paths to the destination enclave: a direct "high-delay" 10Mbps path (from enclave 1 to enclave 2), and an indirect "low-delay" 10Mbps path (from enclave 1 to enclave 3 to enclave 2). The direct path is intentionally impaired by setting its packet loss rate to 100% around 12 seconds into the experiment, with the impairment subsequently removed around 22 seconds into the experiment. The direct path's delay of 70ms (plus CAT and BPF queue delays) also precludes any low-latency traffic sent along that path from being delivered on time to the destination. The indirect path incurs a delay of 2ms (plus queue delays), but at 10Mbps capacity, this path can accommodate only a portion of both flows at the same time.

Source enclave 1 sends one latency-sensitive UDP flow with LOG utility, a priority of 1 and a deadline of 70ms. The flow is capped at a maximum offered load of 5Mbps. The enclave also sends a second UDP flow (capped at 10Mbps) with a LOG utility function, priority 5 and no latency constraints. The system is configured not to reorder UDP, so head of the line blocking is not an issue.



4.17.3 Expected Results

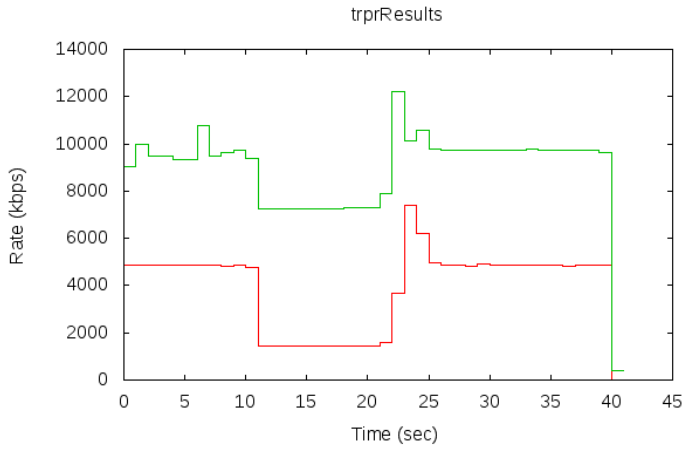
Both flows should be delivered on time before and after the outage: 5Mbps for the latency-sensitive flow and 10Mbps for the latency-insensitive flow. It is expected that the BPF forwards the former along the low-delay path, and is able to deliver all of both flows, implying that the high-delay capacity is used only for latency-insensitive traffic. Therefore, the latency-sensitive flow should be delivered with a 2ms propagation delay (plus queuing delay and transmission delay) and the latency-insensitive traffic with a 70ms propagation delay (plus queuing and transmission delay). Note that with CUBIC congestion control in the CATs (the default), the CAT queue delays are non-trivial.

During the outage, both flows must be served via the indirect, low-delay 10Mbps path. Assuming 15% overhead and a 5:1 prioritization ratio, the low priority latency-sensitive flow should get 1.4Mbps and the high priority latency-insensitive flow should get 7Mbps. The packets of the low-latency flow should be delivered within the latency bound.

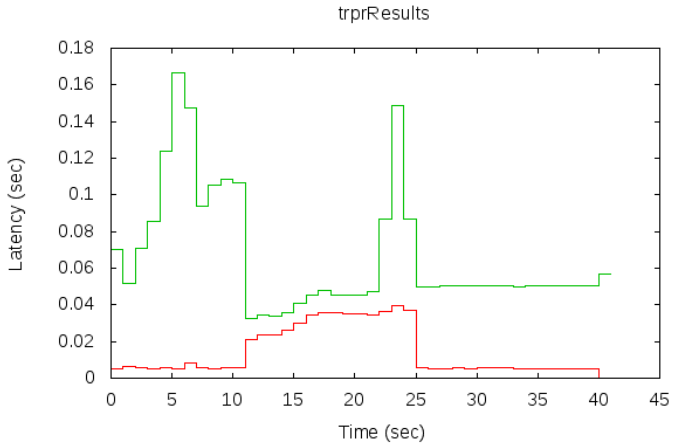
Note that the BPF drops packets that will not make the deadline, so if queuing delay is too high along the low-delay path to meet the latency sensitive flow's deadline, the observed effect will be a lower goodput for the latency sensitive flow.

4.17.4 Observed Results

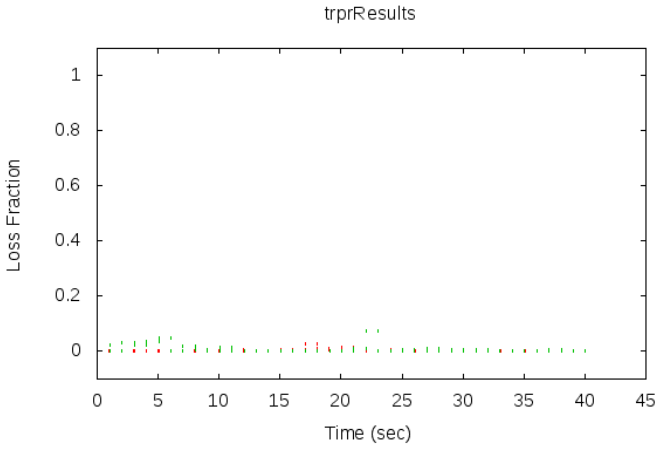
In the observed goodput (below), the latency sensitive flow is shown in red and the latency insensitive flow in green. Before and after the outage, both flows get all traffic through (5 and 10Mbps, respectively). During the outage, we observe the expected 5:1 ratio.



The observed latencies show that the latency-sensitive flow does take the low delay path, which has a latency of less than 10 ms except during the outage. During the outage, the queues build up and the latency increases, but it stays under 40 ms. The latency insensitive flow sees slightly higher latency even during the outage (when the two flows share the same path and the same queues) because latency sensitive packets are dequeued first at the BPF.



The measured loss rates indicate that loss mainly occurs during transition periods.



4.17.5 Conclusions

This experiment demonstrates proper multi-path latency-aware forwarding, outage avoidance and CAT operation of latency-sensitive traffic.

4.18 4-node-system

4.18.1 Goal

This experiment tests the advanced operation of GNAT for latency-constrained traffic, particularly whether GNAT will be able to use the full network capacity (multiple paths to the destination), and send latency-sensitive traffic onto faster paths, leaving the slower paths for less constrained latency-sensitive traffic or latency insensitive flows.

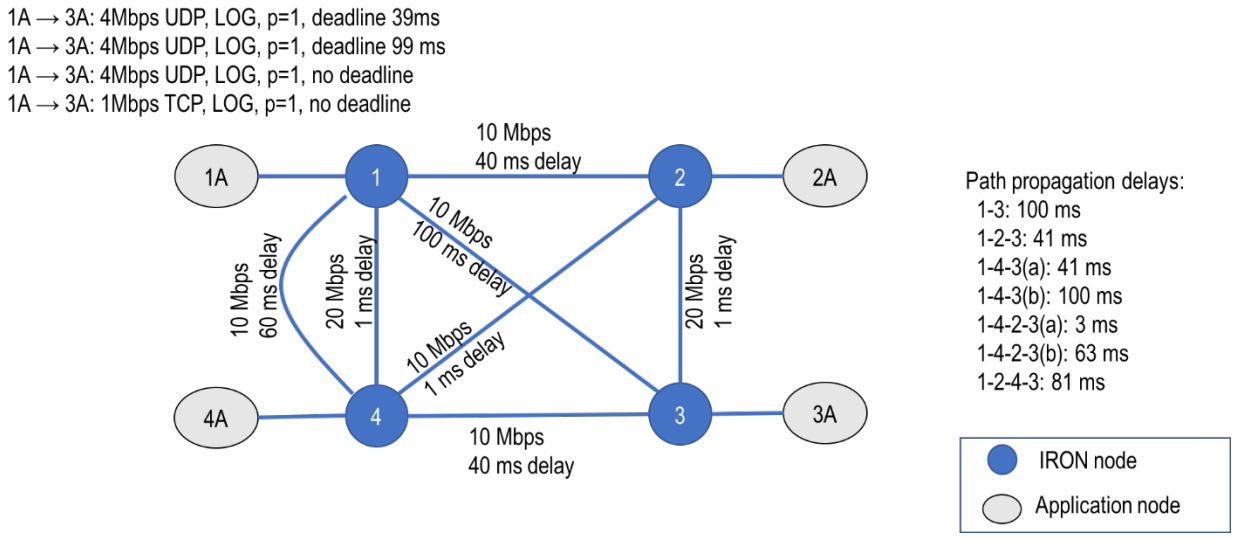
This experiment leverages and tests the following components and features:

- CAT operation for latency sensitive traffic,
- UDP proxy for LOG-utility admission,
- BPF multi-path latency-aware forwarding,
- Dual-homed enclaves.

4.18.2 Topology and Conditions

The experiment runs over a 4-node square topology as shown in the figure below, where the application node at enclave 1 sends three elastic UDP flows and one TCP flow to the application node in enclave 2. The first is a latency-sensitive UDP with a tight 39ms deadline, capped at a maximum send rate of 4Mbps. The second is also latency-sensitive, capped at 4Mbps, but with a longer deadline of 99ms to be delivered. The third flow is latency insensitive capped at 4Mbps. The TCP flow is capped at 1 Mbps.

There are many paths from the source to the destination, but they do not support all flows. (Note that the stated link delay is just propagation delay. There is additional delay due to queuing and transmission times.) The first flow with a 39ms deadline can only be delivered on time via one path. Some but not all paths can support the second 99ms flow. The third UDP flow and the TCP flow can take any available path.



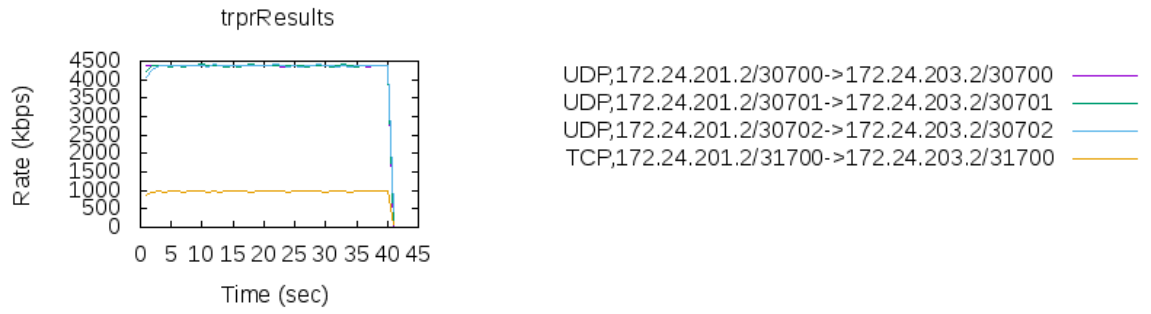
4.18.3 Expected Results

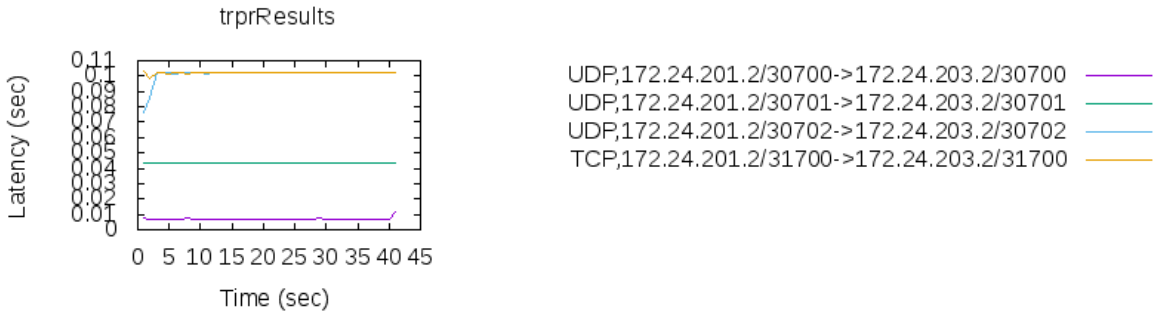
Because the direct link between enclave1 and enclave3 is too slow to meet the latency

requirement of the low latency link, these packets take a multi-hop route through enclave4 and enclave2 respectively before reaching enclave3. All flows are able to operate at 4Mbps and packet loss should be 0. The latency of each flow will be dictated by corresponding latency restriction, but should remain steady throughout the experiment.

4.18.4 Observed Results

The observed results (recorded at enclave 3) show that all traffic reaches the destination. The latencies fit into three tiers based on the deadlines.





4.18.5 Conclusions

This experiment demonstrates proper complex multi-path latency-aware forwarding.

4.19 3-node-latency-routing

4.19.1 Goal

This experiment tests that latency-sensitive traffic is correctly routed along a low delay path, even if that displaces latency-insensitive traffic. It also tests that this latency-aware displacement only happens to the degree that it is appropriate to optimize utility.

This experiment leverages and tests the following components and features:

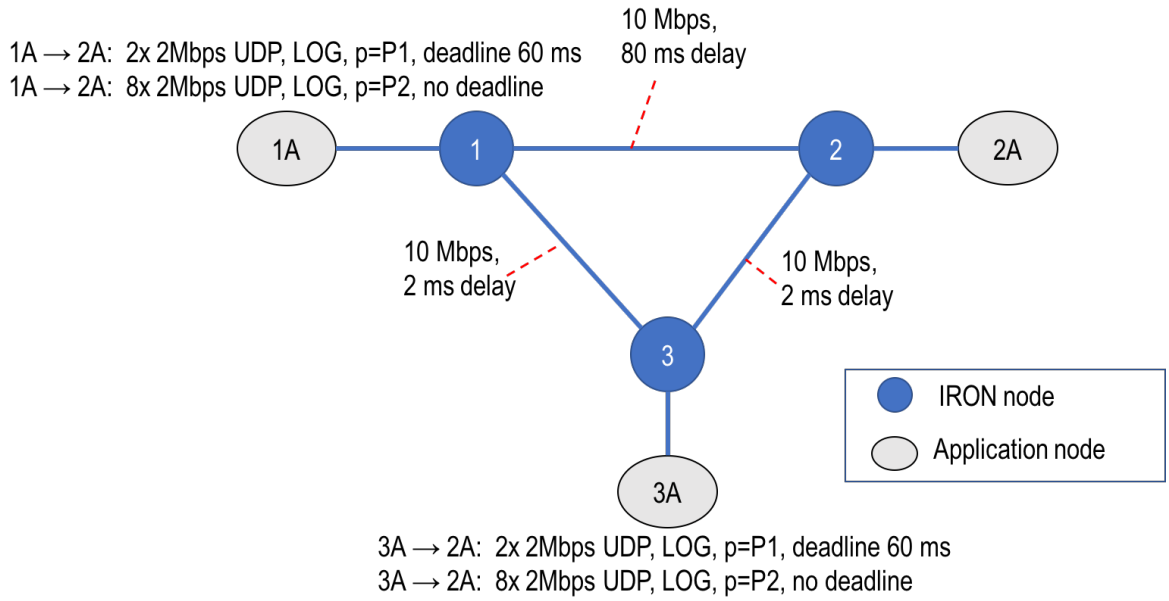
- CAT operation, including estimation of link delays
- BPF estimation of route delays
- UDP proxy with LOG-utility admission and flow prioritization,
- BPF multi-path forwarding with latency-aware routing

4.19.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below, where the application nodes in enclaves 1 and 3 both send traffic to the application node in enclave 2.

Each source sends 10 UDP flows to the destination, each with an offered load of 2Mbps. If latency were not taken into account, each flow would take only the direct path, and each source would fill the direct link. However, 2 of the flows from each source are latency-sensitive with a deadline of 60 ms. These two flows cannot use the high delay link from enclave 1 to enclave 2 and therefore these two 1-2 flows must take the indirect path through enclave 3. This leaves less capacity for the 3-2 flows, so some of the latency insensitive flows from 3-2 must instead take the path from 3 to 1 to 2.

There are 3 experiment runs. In the first, all the flows have priority 1. In the second, the latency-insensitive traffic has priority 5, while latency-sensitive still has priority 1. In the third run, the priorities are reversed, so latency-sensitive traffic has priority 5. The three runs are intended to test that latency-aware routing does not break prioritization.



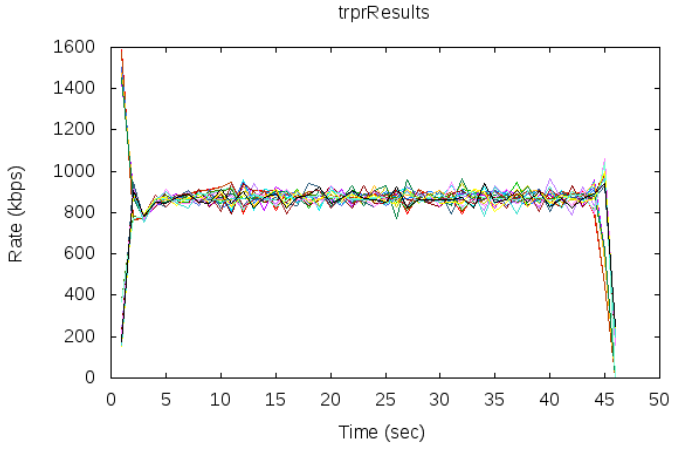
4.19.3 Expected Results

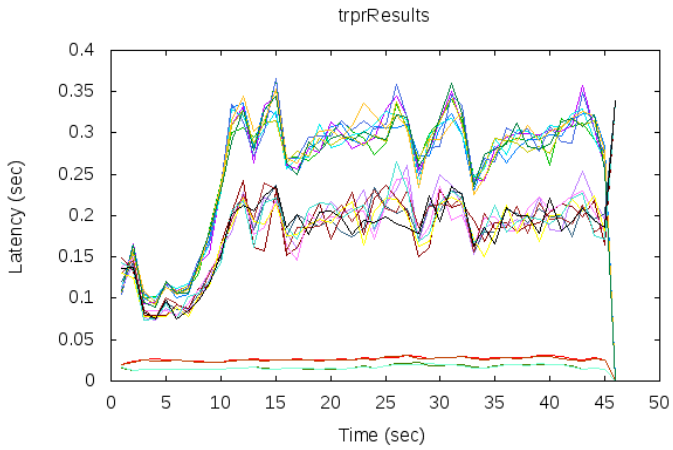
In order to meet the deadline, the low latency traffic from enclave 1 must be routed along the indirect (1->3->2) route. In order for that low latency traffic to use that route, some of the latency insensitive traffic from enclave 3 must be displaced to its indirect route (3->1->2). Assuming 15% overhead, the expected goodput is as follows:

Run	Situation	Latency-sensitive goodput	Latency-insensitive goodput
1	P1=P2=1	17Mbps/20 = 850kbps	850kbps
2	P1=1, P2=5	17Mbps/(5x16 + 4)=202kbps	5x202kbps = 1012kbps
3	P1=5, P2=1	17Mbps x 5 / (5x4 + 16) = 2361kbps However, the indirect path can only handle 8.5Mbps, so we can only expect to achieve 2125kbps per flow.	The entire high-delay path is available for latency-insensitive traffic, so we expect 8.5Mbps/16 = 531kbps

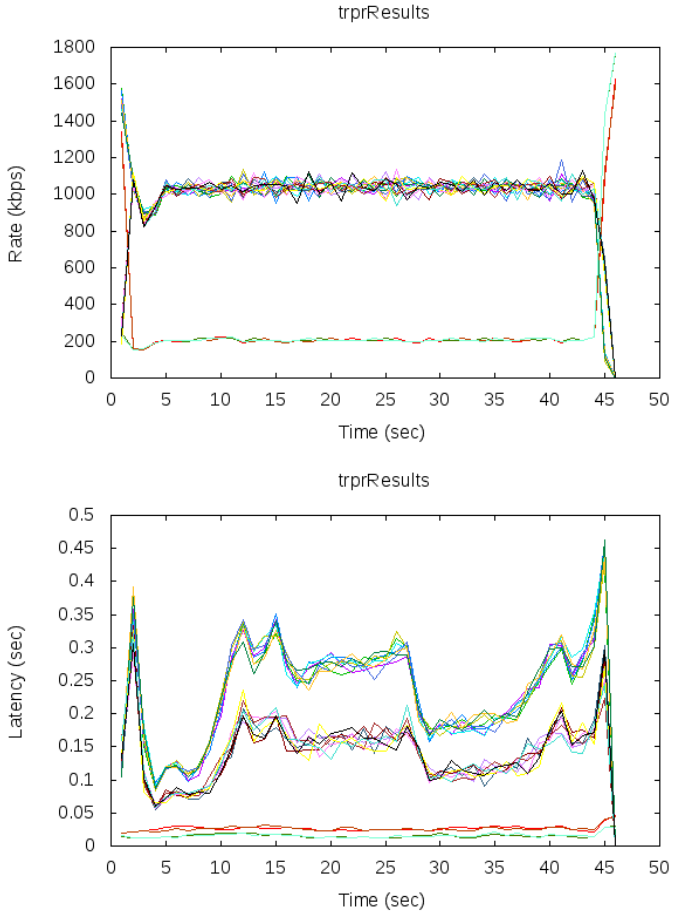
4.19.4 Observed Results

In the first run, all flows do achieve equal goodput around 850kbps, as expected. The latency graph shows that the latency-sensitive flows meet their deadline, and the latency insensitive flows achieve a latency that is somewhere between what we'd expect from each of the two paths, showing that those flows follow both paths.

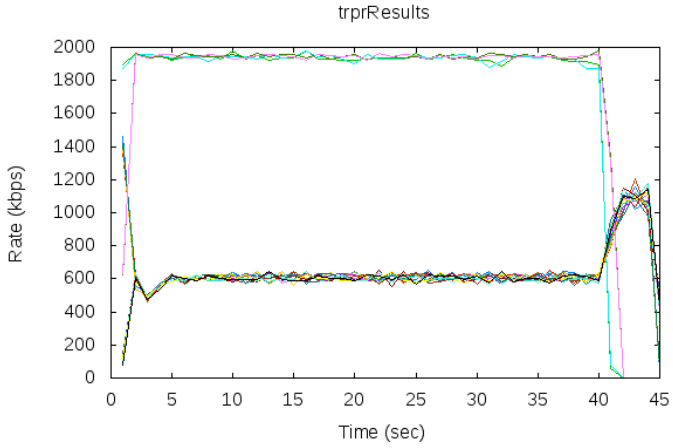


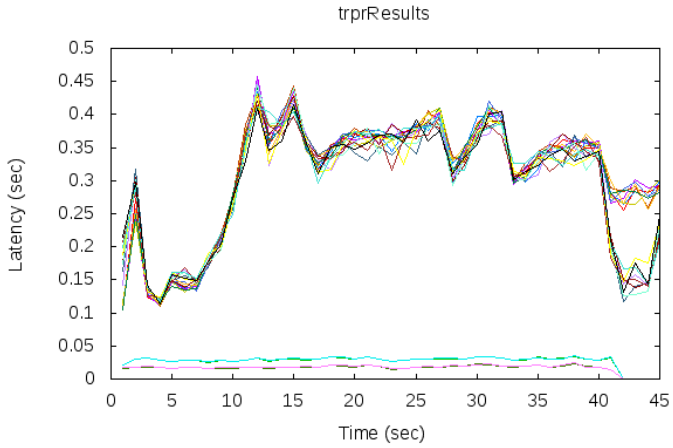


In the second run, prioritization is shown to work correctly, even while meeting the latency constraints for the lower priority flow. The latency for the high priority flows is lower in run 2 than in run 1, because there is more capacity available along the low delay path due to having less latency sensitive traffic.



In the third run, results show that the latency sensitive traffic is able to use almost all of the indirect path, and the latency insensitive traffic takes up the entire high delay path. This shows that GNAT is work conserving and does not waste bandwidth that cannot be used by latency sensitive traffic.





4.19.5 Conclusions

This experiment demonstrates that Latency-Aware routing behaves correctly and does not break prioritization.

4.20 3-node-smallhighprio

4.20.1 Goal

This experiment showcases GNAT's ability to counter a common problem with backpressure forwarding: namely, pure backpressure routing tends to starve small flows that don't maintain a big enough backlog of packets (i.e., a large enough *gradient*, in backpressure terms), regardless of flow priority. GNAT does not have this problem. This experiment tests the ability of GNAT to admit and forward low-rate flows when other high-rate flows are competing.

This experiment leverages and tests the following components and features:

- UDP proxy admitting low-rate flows if the priority allows it,
- BPF fair forwarding, including low-rate flows.

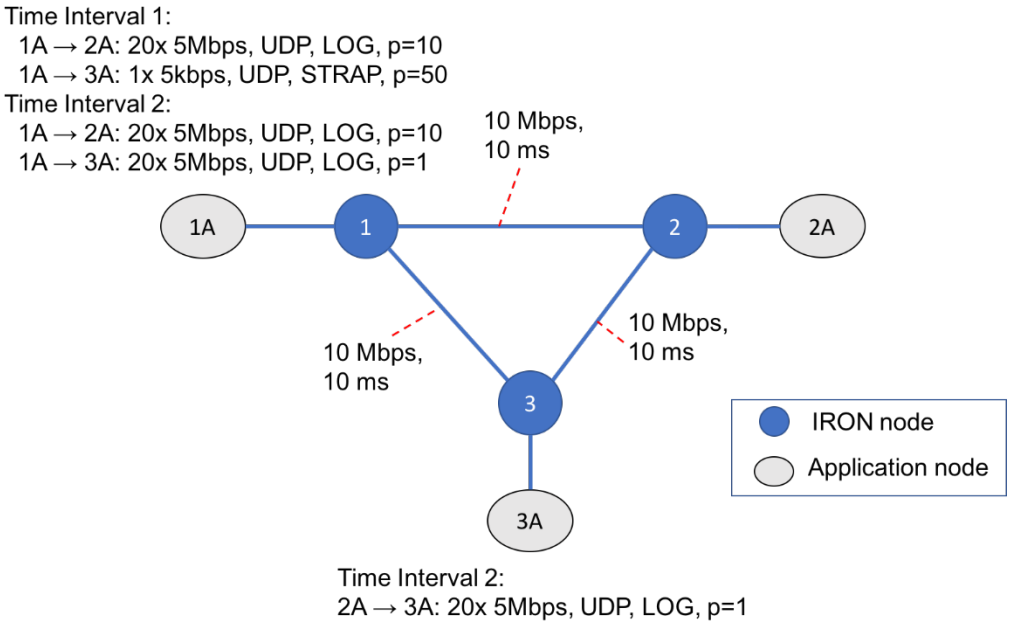
Although not expressly included in this experiment, this GNAT feature also prevents GNAT from starving flows that have a low rate because they are just starting up or shutting down.

4.20.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below. The application node in enclave 1 sends 20 medium-priority large flows to the application node in enclave 2, and one small high-priority flow to the application node in enclave 3. The 20 flows to enclave 2 have a priority of 10, and the small flow to enclave 3 has priority of 50. With basic backpressure forwarding, this small flow would be completely starved, because the queue at enclave 1 for enclave 3 would never have a higher gradient than that for enclave 2.

After 40s, the small high-priority flow is replaced with 20 large flows to enclave 3, and enclave 3 sends 20 flows to enclave 2, all with priority 1, to ensure that whatever mechanism we employ does not give other flows an unfair advantage. The second phase of the experiment also demonstrates prioritization when optimal throughput does not generate optimal utility.

All flows are UDP with LOG-utility, except for the small high-priority flow, which is inelastic (STRAP utility).



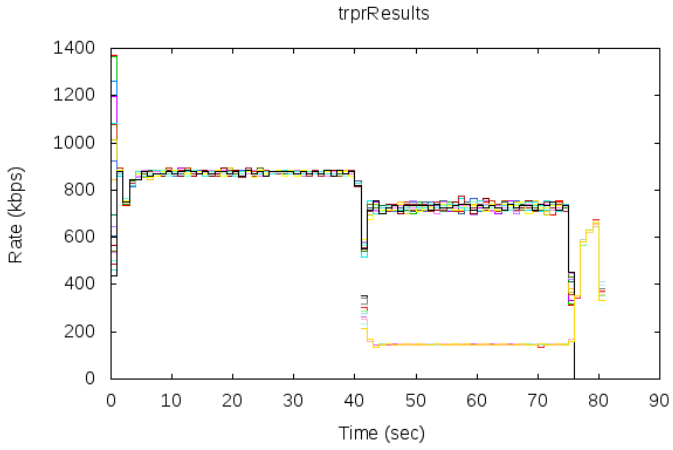
4.20.3 Expected Results

During the first half of the test, we expect most of the 20Mbps capacity to be spread evenly across all low-priority flows from enclave 1 to enclave 2 (85% of 1Mbps each, accounting for 15% overhead). The 5Kbps flow should not be starved or delayed excessively.

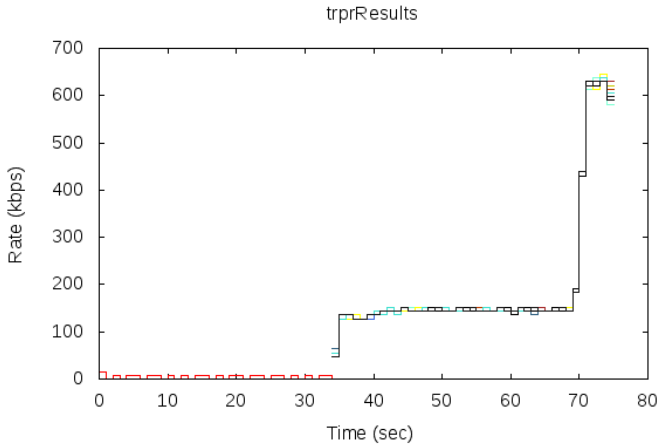
During the second half of the test, we expect to see 10/11 of the 17Mbps capacity (20Mbps less 15% overhead) go to the priority 10 flows, and 1/11 go to the priority 1 flows, or about 770kbps for each priority 10 flow and 80kbps for each priority 1 flow.

4.20.4 Observed Results

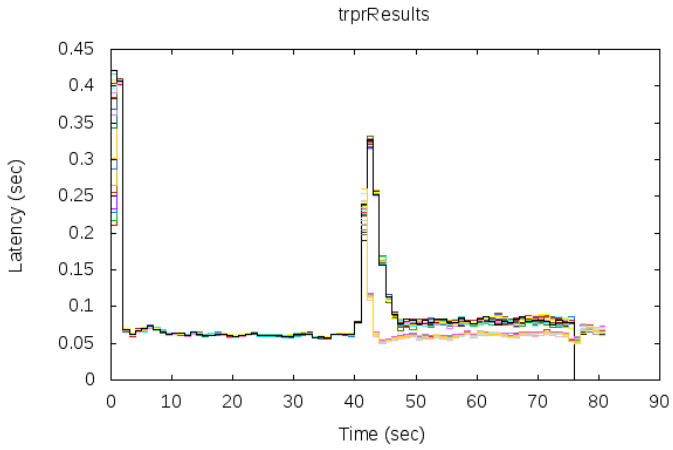
The observed goodput at enclave 2 shows equal goodput for all low priority flows during the first time period, and the expected split between high and low priority flows during the second time period.



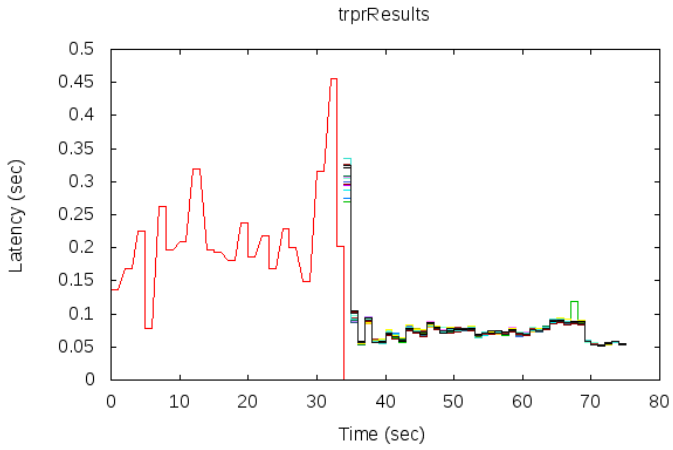
The observed goodput at enclave 3 shows that the small high priority flow is not starved (and in fact gets goodput for the entire 35 seconds before the next time period starts). During the second time period, the low priority flows get the expected goodput, matching the goodput observed at enclave 2.



The latency results observed at enclave 2 show fairly constant latency across all flows, which shows that zombie latency reduction is effectively maintaining a fairly constant queue depth.



The observed latency at enclave 3 shows that the small flow is not subjected to abnormally large delays.



4.20.5 Conclusions

This experiment demonstrates the ability of GNAT to support low-rate flows in the presence of competing traffic.

4.21 3-node-util-vs-thruput

4.21.1 Goal

This experiment tests GNAT's ability to optimize network utility based on LOG utility functions, even when the maximal utility solution does not optimize throughput. In particular, a higher priority flow must displace **two** lower priority flows in order to increase its throughput, so if the priorities are similar, the two flows combined will still achieve higher utility than the single flow, but as the priorities diverge, the single flow will start to displace the lower priority flows.

This experiment leverages and tests the following components and features:

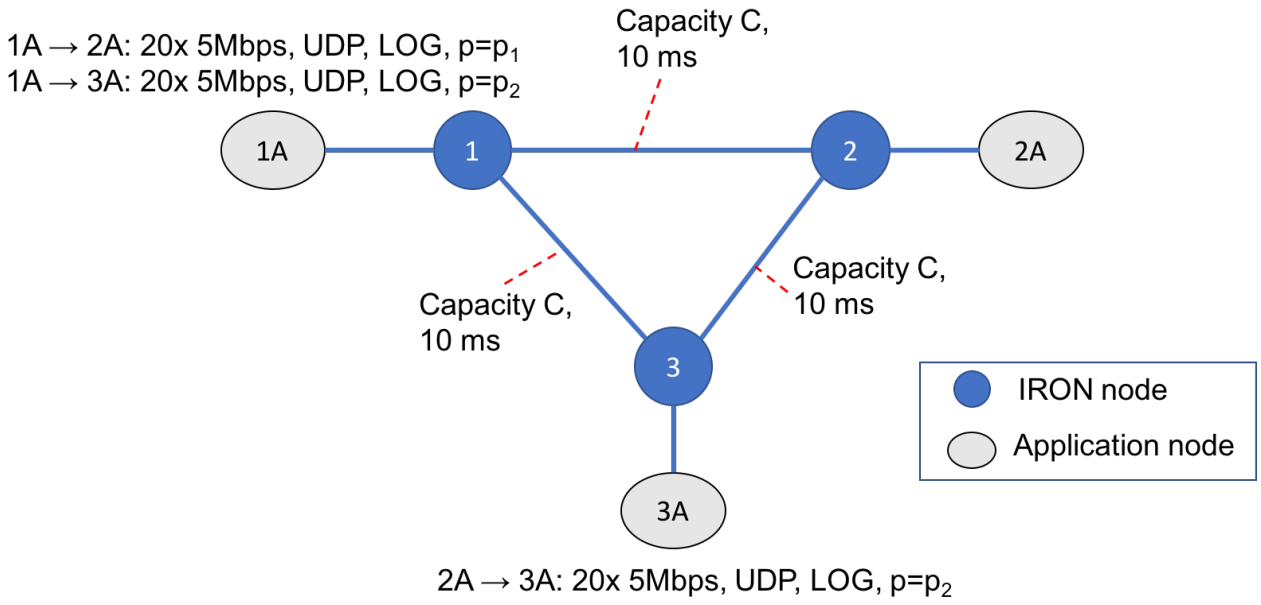
- CAT operation,
- UDP proxy with LOG-utility admission and flow prioritization,
- BPF multi-path forwarding
- Distributed network utility optimization

4.21.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below, with 20 high priority LOG utility UDP flows from enclave 1 to enclave 2, 20 low priority LOG utility UDP flows from enclave 1 to enclave 3, and 20 low priority LOG utility UDP flows from enclave 3 to enclave 2. Each high priority packet may be forwarded over one of two paths: a direct path (from enclave 1 to enclave 2) or an indirect path (from enclave 1 to enclave 3 to enclave 2). High priority traffic taking the indirect path will necessarily cut down on the throughput available for twice as much low priority traffic. Each flow has sufficiently high offered load that offered load will never be the bottleneck.

The test includes four runs with different link capacities and priorities, as shown in the table below.

Test run	High priority (1 to 2) p_1	Low prio (1 to 3 & 3 to 2) p_2	Link Bandwidths (all are the same) C
1	16	4	10Mbps
2	16	4	50Mbps
3	8	4	10Mbps
4	8	4	10Mbps



4.21.3 Expected Results

We expect the network utility to be optimized according to a log utility function. If the high priority flows have priority p_1 and goodput rate r_1, low priority flows have priority p_2 and rate r_2, and overhead is 15%, the observed goodput should maximize:

$20\ p_1 \log r_1 + 20\ p_2 \log r_2 + 20\ p_2 \log r_2$

subject to

$20\ r_1 + 20\ r_2 <= 0.85 * C$

This maximization can be solved by finding where the derivative is 0:

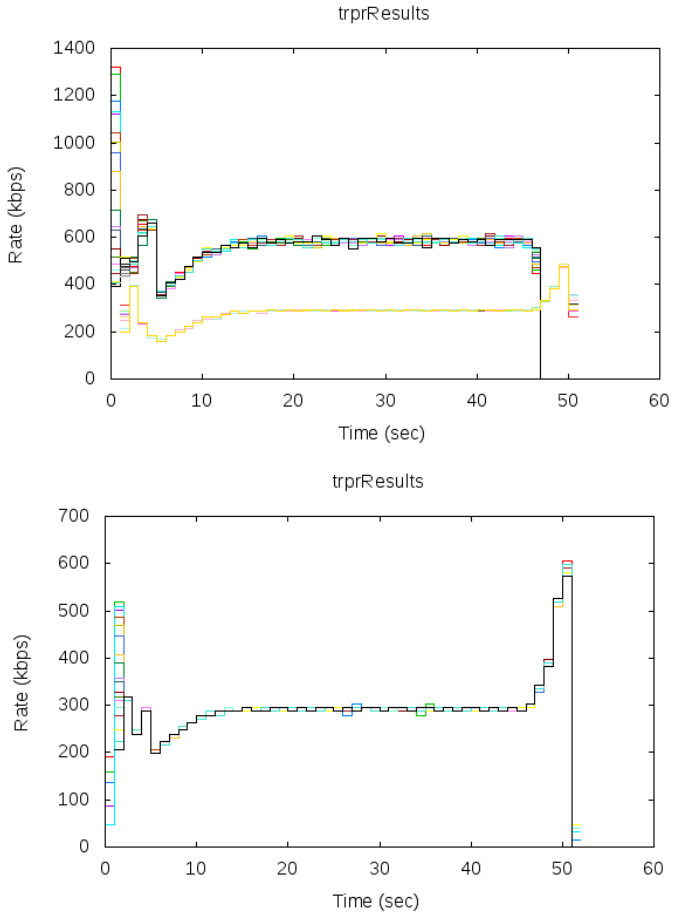
$20\ p_1 / (0.85 * C - r_2) = 40\ p_2 / r_2$

The solution to the above equations for our 4 test cases is shown in the table below.

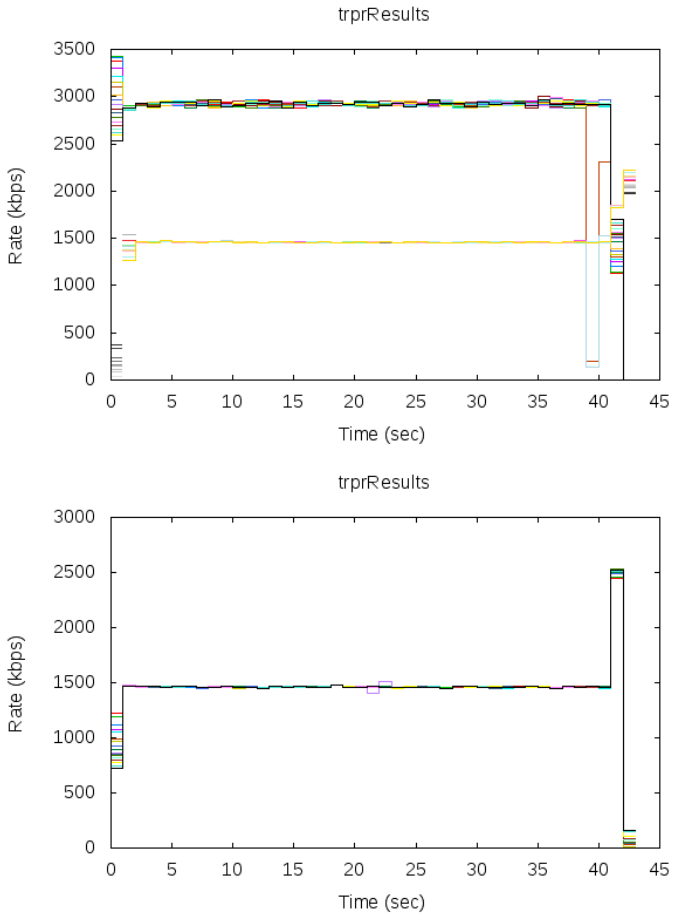
Test run	p_1	p_2	C	r_1	r_2
1	16	4	10Mbps	583kbps	292kbps
2	16	4	50Mbps	2.9Mbps	1.45Mbps
3	8	4	10Mbps	437.5	437.5
4	8	4	10Mbps	2.18Mbp	2.18Mbps

4.21.4 Observed Results

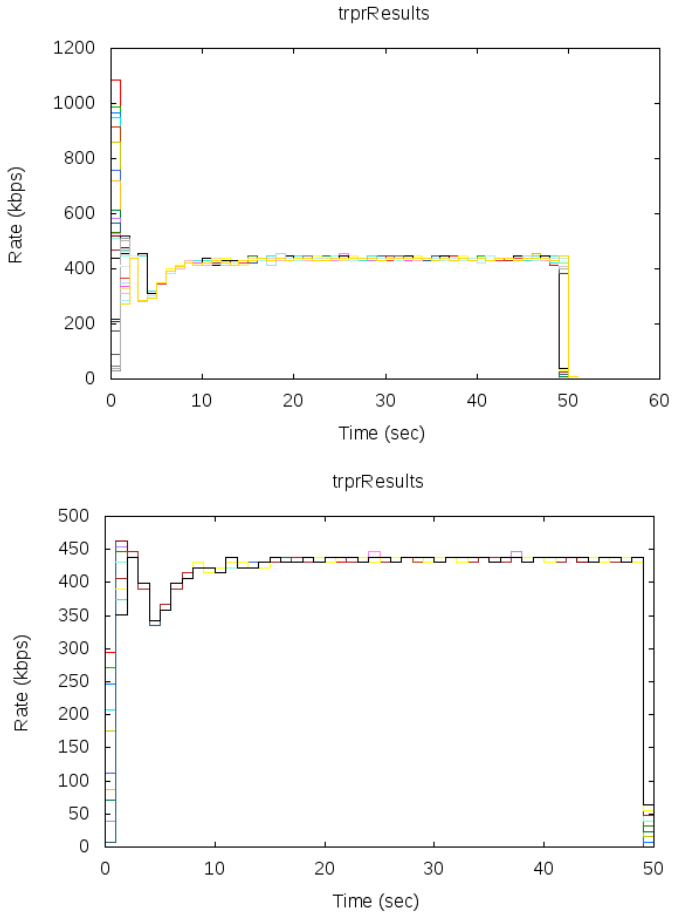
The results from the first run show the expected goodput ratio. The first graph shows the goodput observed at enclave 2, with the high priority flows getting approximately twice as much goodput as the low priority flows. This means that the high priority flows must have sent some packets along the indirect path, displacing the low priority flows. The second graph shows the goodput at enclave 3 of the 1->3 flows, which (as expected) matches the goodput of the low priority flows from enclave 3 to enclave 2.



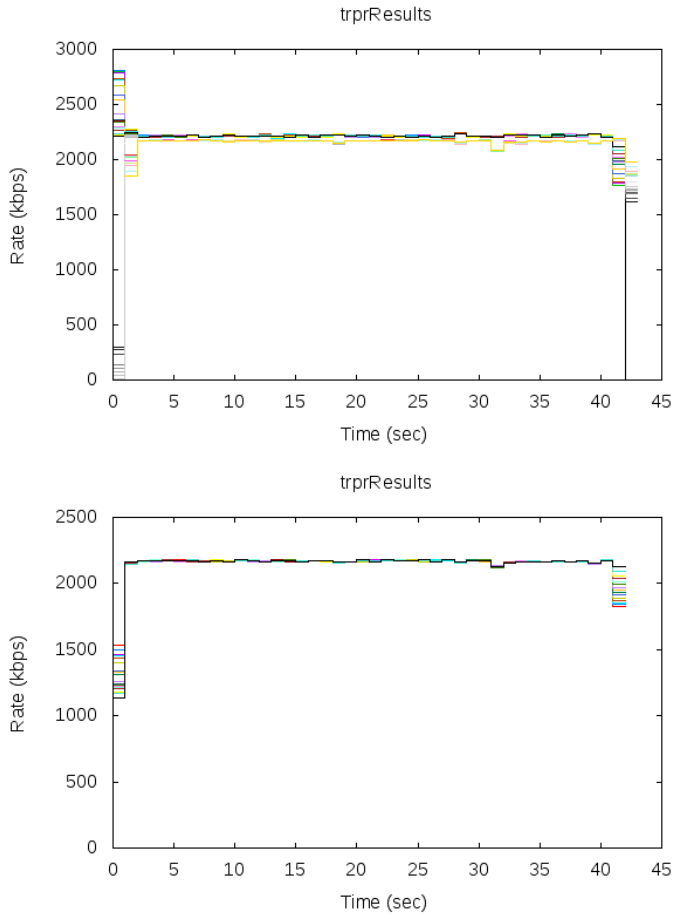
The results from the second run, which is identical except with 5x as much bandwidth, also show the expected goodput ratio.



In the third test run, below, the priorities were assigned such that we get the same utility from running one unit of the high priority flow as from running one unit of each of the low priority flows. Thus, the achieved goodput is equal for all flows.



The fourth run has higher bandwidth and show get the same goodput for all flows. This matches the results shown below.



4.21.5 Conclusions

This experiment shows that network utility is optimized, even at the expense of total achieved goodput.

4.22 4-node-strap

4.22.1 Goal

This experiment tests triage and prioritization when using the STRAP utility function.

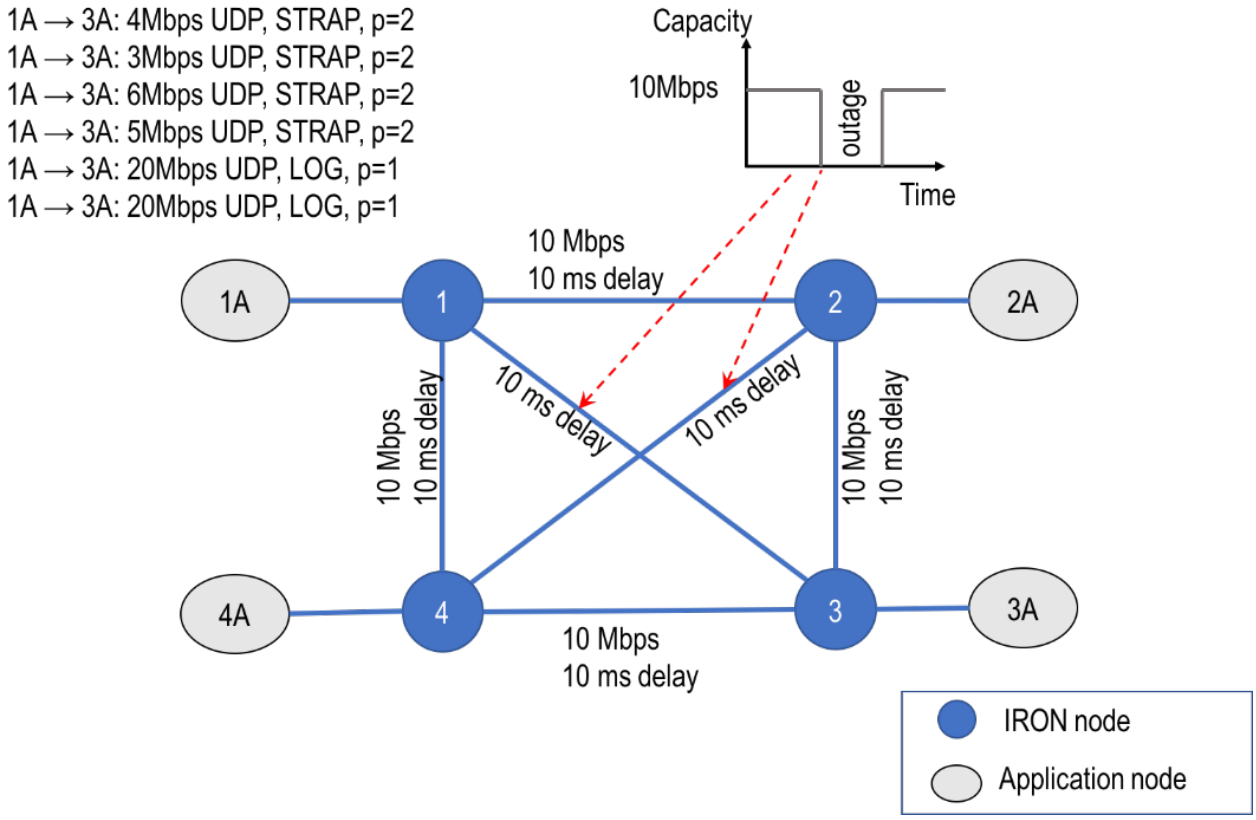
4.22.2 Topology and Conditions

This experiment uses a 4-node full mesh topology with all the links set to 10 Mbps with 10ms delay between the GNAT nodes.

There are 4 inelastic flows from the application node in enclave 1 to the application node in enclave 3 , all with priority 2, and rates of 3Mbps, 4Mbps, 5Mbps and 6Mbps.

There are also 2 elastic flows from enclave 1 to enclave 3 with priority 1.

During the experiment, the direct paths between enclaves 1 and 3, and also between enclaves 2 and 4 are removed. They are restored after 10 seconds.



4.22.3 Expected Results

Initially all flows should fit, and both elastic flows will operate at the same rate: (25.5Mbps - 18Mbps)/2 = 3.75Mbps.

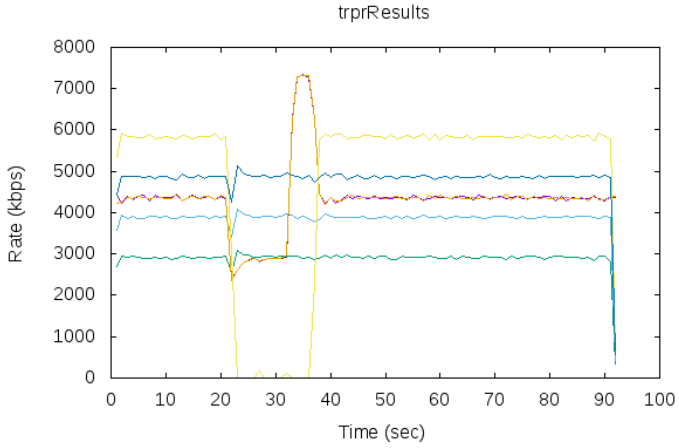
During the outage, there is a total of 18Mbps of inelastic traffic while the capacity is 17Mbps (20Mbps minus overhead). Since all the inelastic traffic cannot fit, and the flows are all the same priority, the flow with the largest rate should be triaged, since it has the least utility per bit per second. The elastic traffic should be reduced to fill the remaining capacity (17Mbps - 12Mbps)/2 = 2.5Mbps.

When the direct links are restored, all flows should resume after a short convergence time.

4.22.4 Observed Results

In the observed goodput results below, the orange and red lines are the elastic flows, which operate at around 4.3Mbps other than during the outage. The inelastic flows each operate at their full offered load. Note that we're getting slightly better goodput than expected, presumably because the overhead is slightly lower than what we've accounted for.

During the outage, the 6Mbps STRAP flow is triaged, as expected, and the elastic flows drop back to slightly under 3Mbps (slightly higher than expected). After the outage ends, the elastic flows react quickly and pick up the extra capacity until the 6Mbps STRAP flow has time to re-start.



4.22.5 Conclusions

STRAP provides a viable approach for performing triage an prioritization of inelastic traffic.

4.23 3_node_960_flows

4.23.1 Goal

This experiment tests the operation of GNAT with a very large number of flows and under load, particularly whether GNAT will be able to use the full network capacity (both paths to the destination), admit and prioritize close to 1,000 TCP and UDP flows. It also tests operations of admission with inelastic (STRAP) utility and prioritization between flows with different utility functions.

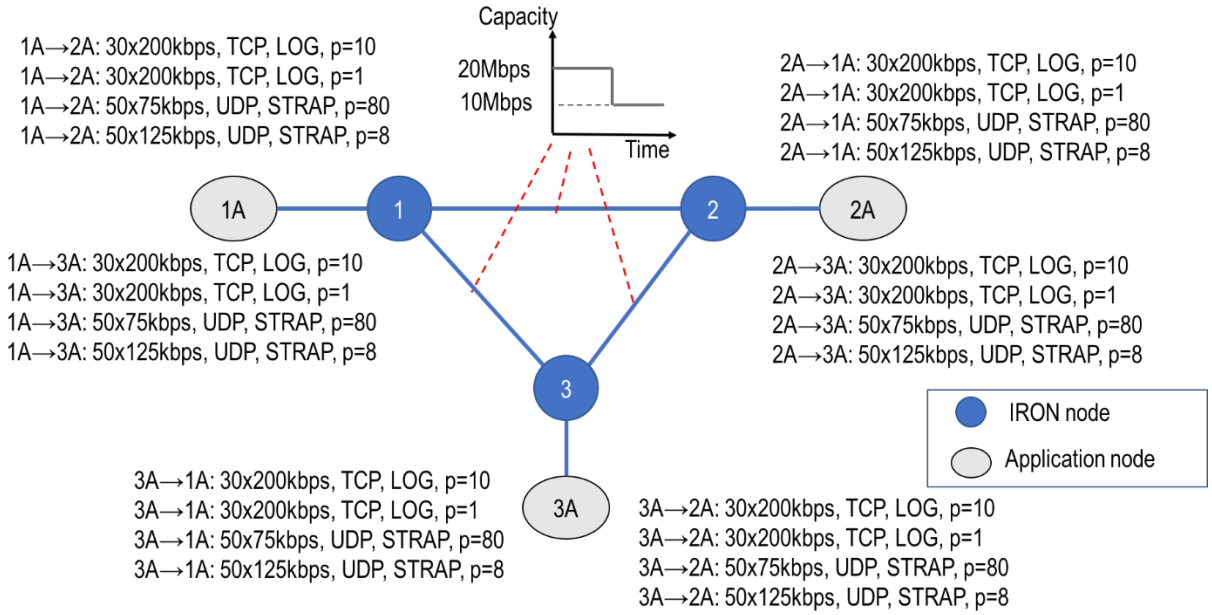
This experiment leverages and tests the following components and features:

- CAT operation under load,
- UDP proxy handling a large number of flows with STRAP utility,
- TCP proxy handling a large number of flows with LOG-utility,
- Flow prioritization between different proxies and utility functions,
- BPF multi-path forwarding under load.

4.23.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below, where each enclave's application node sends 160 flows to each other enclave's application node. Each packet may be forwarded over one of two paths: a direct 20Mbps path, and an indirect 20Mbps (2-hop) path. All links are intentionally impaired by setting their capacity down to 10Mbps 22 seconds into the experiment, with the impairment lasting for the remainder of the experiment.

Each source node sends 30 medium-priority (p=10) TCP LOG-utility flows, 30 low-priority (p=1) TCP LOG-utility flows, 50 high-priority (p=80) 75kbps UDP inelastic STRAP-utility flows and 50 medium-priority (p=8) 125kbps inelastic STRAP-utility flows. The TCP flows are capped at a send rate of 200kbps to avoid hitting non-GNAT limitations on the ingress links.



4.23.3 Expected Results

The STRAP-utility flows will claim a total of 10Mbps per source-destination pair (3750 for the high priority and 6250kbps for the medium priority). Either the network can support them at their nominal rate, or they will triage out in priority order. Before the impairment, there is enough network capacity (around 17Mbps per source after overhead) to admit all STRAP flows. Therefore, each UDP flow should receive its nominal delivery rate of 75kbps and 125kbps. This leaves around 7Mbps for the LOG-utility traffic, which will be receive a 10-to-1 ratio between flows of priority 10 and 1.

During the impairment, the all STRAP-utility flows cannot be supported, so the medium priority flow is triaged out (and therefore not admitted). The smaller, high priority STRAP-utility flows can still be supported by the 10Mbps network and it should receive an aggregate rate of 3.75Mbps, or 75kbps per flow. This leaves 4.75Mbps for LOG flows, which should still exhibit at 10-to-1 ratio.

4.23.4 Observed Results

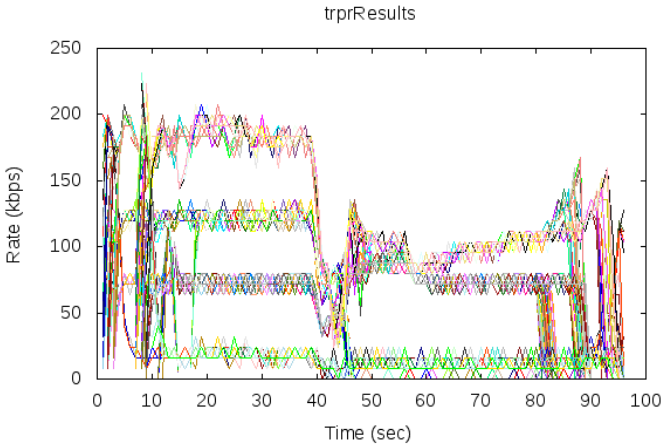
As shown in the goodput graphs below, flows converge rapidly to a stable throughput before the impairment:

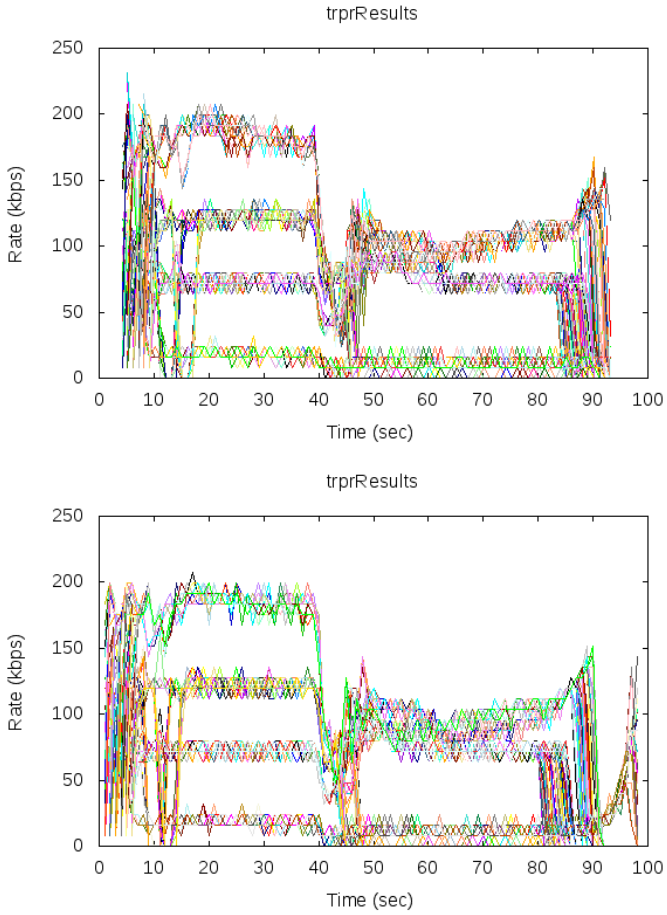
- Each high-priority TCP flow sees around 180kbps goodput
- Each medium-priority TCP flow sees around 20kbps goodput, matching the 10-to-1 ratio.
- The inelastic STRAP UDP flows get 125kbps and 75kbps, as configured.

During the impairment at 22s, we observe the following (after convergence):

- The high-priority TCP flow sees around 110kbps goodput,
- The low-priority TCP flow sees around 10kbps,
- The medium-priority UDP flow has been triaged out: it is no longer admitted or delivered,
- The low-priority UDP flow continues to see 75kbps goodput, as expected.

The results are consistent across all nodes.





4.23.5 Conclusions

This experiment demonstrates proper admission onto the network, prioritization, multi-path forwarding, and CAT operation for a large number of flows.

4.24 4enclave_edge_big_flows

4.24.1 Goal

This experiment tests the operation of GNAT with a very large number of flows and under load, particularly whether GNAT will be able to use the full network capacity (both paths to the destination), admit and prioritize more than 1,000 TCP and UDP flows. It also tests operations of admission with inelastic (STRAP) utility and prioritization between flows with different utility functions, and dual homed nodes.

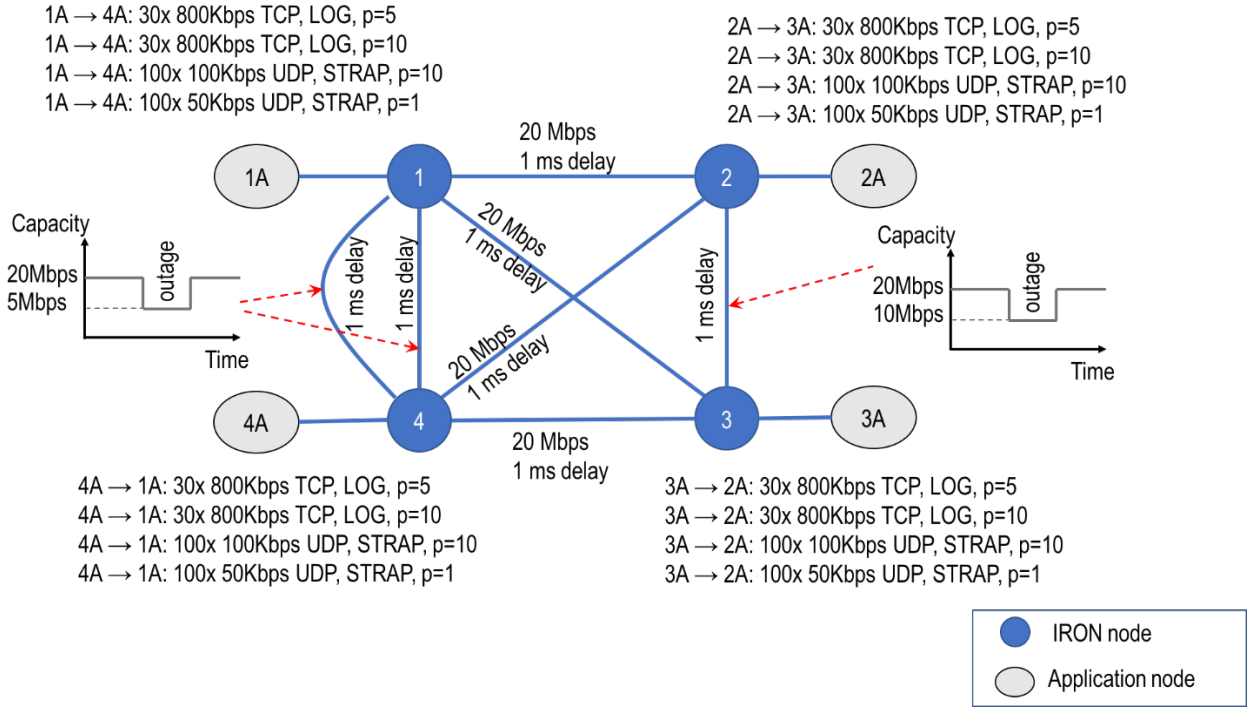
This experiment leverages and tests the following components and features:

- CAT operation under load, including dual homing
- UDP proxy handling a large number of flows with STRAP utility,
- TCP proxy handling a large number of flows with LOG-utility,
- Flow prioritization between different proxies and utility functions,
- BPF multi-path forwarding under load.

4.24.2 Topology and Conditions

The experiment runs over a 4-node mesh topology with two dual-homed enclaves as shown in the figure below, where each "north" enclave's application node sends 260 flows to the corresponding "south" enclave's application node, and vice versa. The north-south vertical links are intentionally impaired midway through the experiment by reducing their capacity.

Each source node sends 30 high-priority (p=10) TCP LOG-utility flows, 30 medium-priority (p=5) TCP LOG-utility flows, 100 high-priority (p=10) 100kbps UDP inelastic STRAP-utility flows and 100 low-priority (p=1) 50kbps inelastic STRAP-utility flows. The TCP flows are capped at a send rate of 800kbps to avoid hitting non-GNAT limitations on the ingress links.



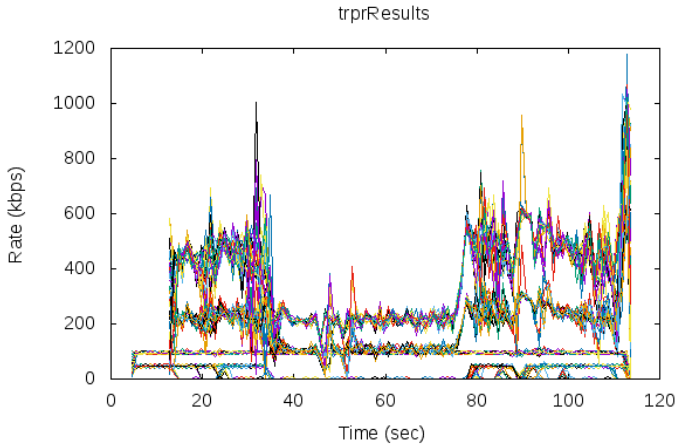
4.24.3 Expected Results

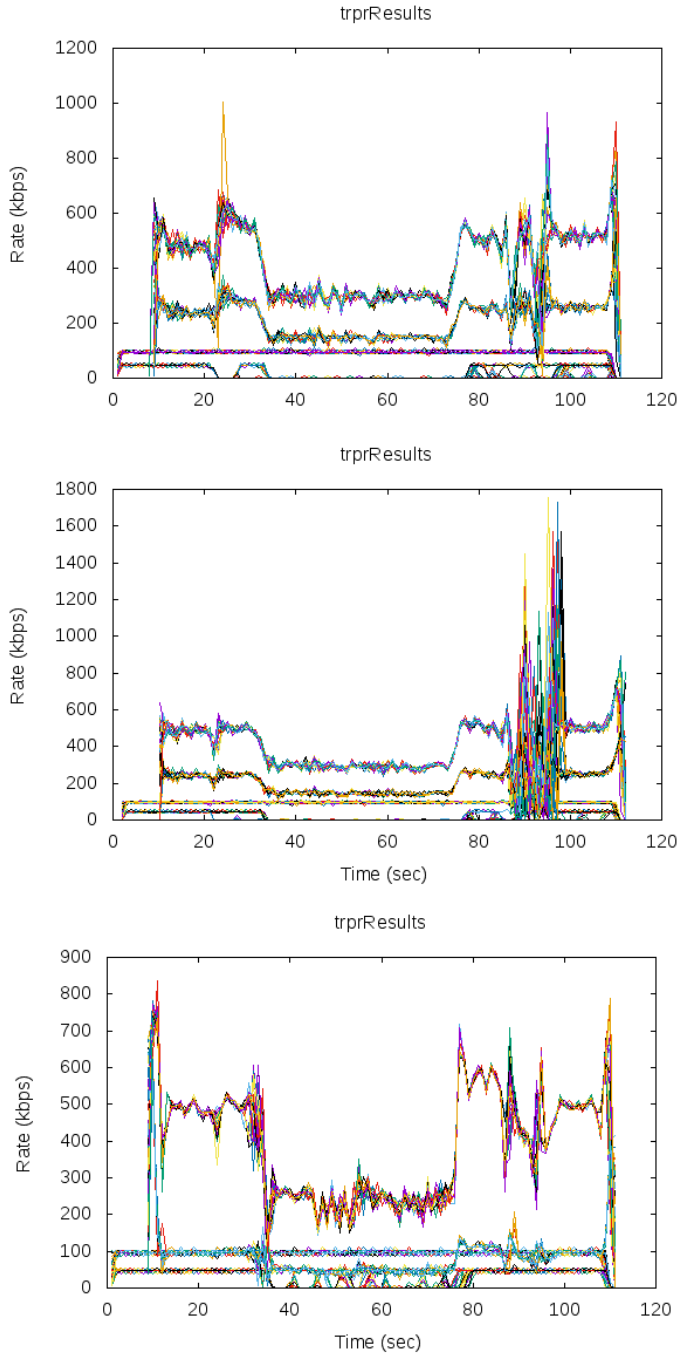
The STRAP-utility flows will claim a total of 15Mbps per source-destination pair. Either the network can support them at their nominal rate, or they will triage out in priority order. Before the impairment, there is 100Mbps of north-south capacity, leaving 50Mbps minus overhead for each source-destination pair, or 42.5Mbps. This is sufficient to service all STRAP flows, leaving 12.5Mbps for the elastic flows. Therefore, each UDP flow should receive its nominal delivery rate of 100kbps and 50kbps. The TCP flows should be admitted at a 2:1 ratio, achieving 305kbps for each medium priority flow and 611 kbps for each high priority flow.

During the impairment, there is only 40Mbps of north-south capacity, leaving 17Mbps (after overhead) per pair. The STRAP-utility flows could all be supported, but only at the expense of the higher priority elastic flows. Because elastic-only queue depths for higher priorities are taken into account for STRAP triage, the low priority STRAP flow is triaged out (and therefore not admitted). The high priority STRAP-utility flows should still get 100Kbps each. This leaves 7Mbps per pair for TCP flows, which means 77kbps per medium priority flow and 155kbps per high priority flow.

4.24.4 Observed Results

As shown in the goodput graphs below, flows converge to a mostly stable goodput before, during, and after the impairment. The STRAP flows behave exactly as expected: achieving almost exactly 100kbps and 50kbps before the impairment, with the 50kbps low priority flows triaged out during the impairment. The TCP flows get slightly lower goodput than expected, but they do converge to a 2-to-1 ratio between priority 10 and priority 5 flows.





4.24.5 Conclusions

This experiment demonstrates proper admission onto the network, prioritization, multi-path forwarding, and CAT operation for a large number of flows.

4.25 3-node-flog

4.25.1 Goal

This experiment tests triage with the Floored Log (FLOG) utility.

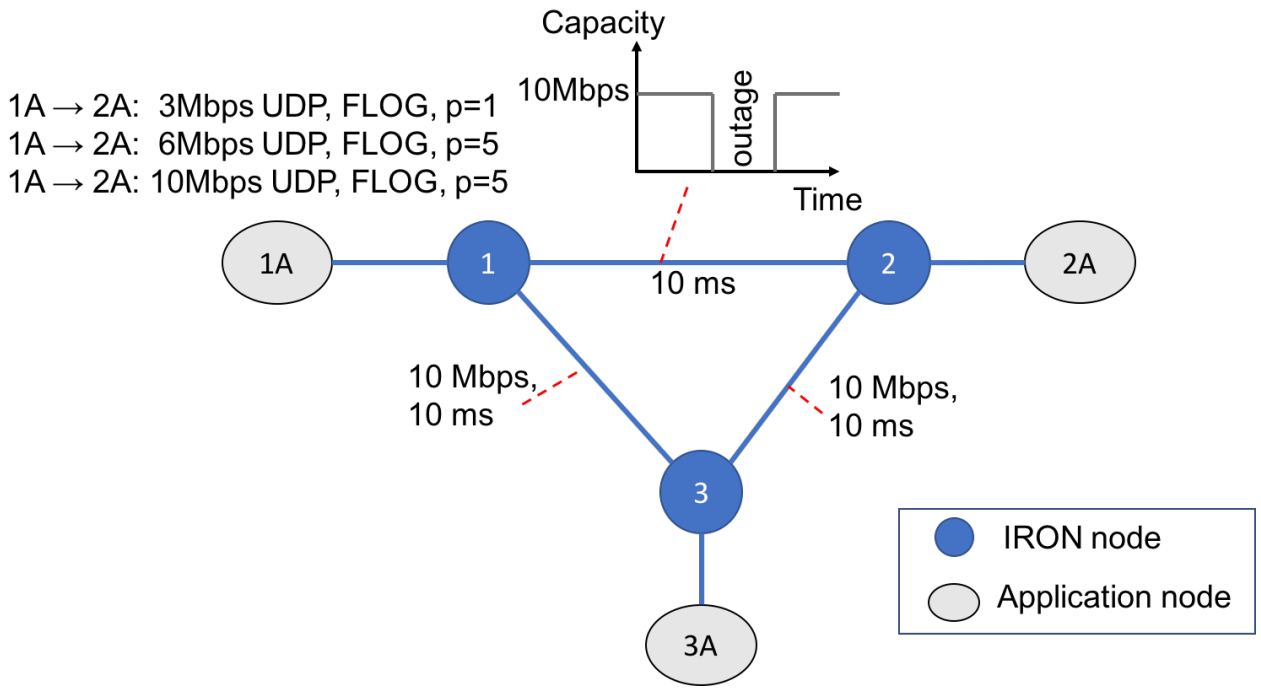
This experiment leverages and tests the following components and features:

- CAT operation,
- UDP proxies for FLOG-utility admission and flow prioritization,
- BPF multi-path forwarding and broken-link avoidance.

4.25.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below, where the application node in enclave 1 sends three flows to the application node in enclave 2. Each packet may be forwarded over two paths: a direct 10Mbps path (from enclave 1 to enclave 2), and an indirect 10Mbps path (from enclave 1 to enclave 3 to enclave 2). The direct path is intentionally impaired by setting its packet loss rate to 1 around 20 seconds into the experiment.

Enclave 1 sends 3 UDP floored LOG flows to enclave 2, being sourced at rates of 3Mbps, 6Mbps and 10 Mbps with priorities of 1, 5 and 5 respectively.



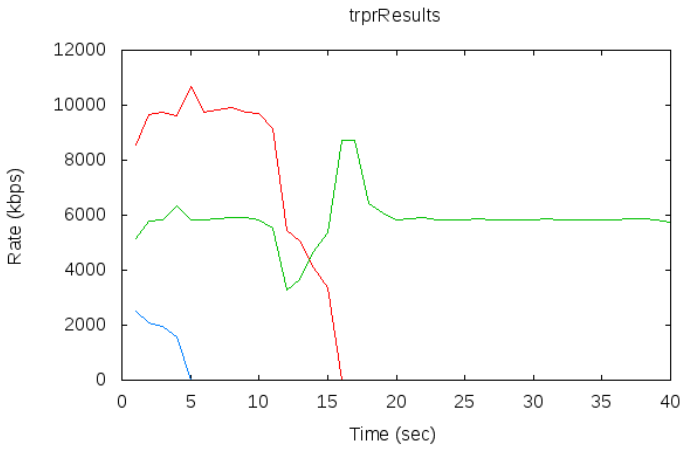
4.25.3 Expected Results

Initially, all the flows will not be able to fit, since there is approximately 16Mbps of available capacity. The low priority flow (3Mbps) should be triaged early.

When the direct link is removed, only the 6Mbps flow will fit and the 10 Mbps flow will be triaged.

4.25.4 Observed Results

The observed results match the expected results described above.



4.25.5 Conclusions

This experiment demonstrates proper admission onto the network using FLOG utility, prioritization, multi-path forwarding, outage avoidance and CAT operation.

4.26 3-node-loss-triage

This experiment tests the supervisory control feature in the GNAT admission planner.

Supervisor control is designed to triage flows which are not being properly serviced due to loss or late packet arrival at the destination. Even if an inelastic flow is admitted by the UDP proxy at the rate it is being sourced, it can get no utility if there are too many packets that arrive too late for the latency requirements. In this case, the bandwidth that it is using will be wasted. Supervisory control keeps track of the packets admitted by the source and those acknowledged by the destination as being received on time and uses this information to determine whether a flow is being properly serviced. Any flows not being properly serviced are triaged, one at a time, until the remaining flows are meeting their latency constraints.

This experiment tests that flows are properly triaged if and only if latency requirements cannot be met.

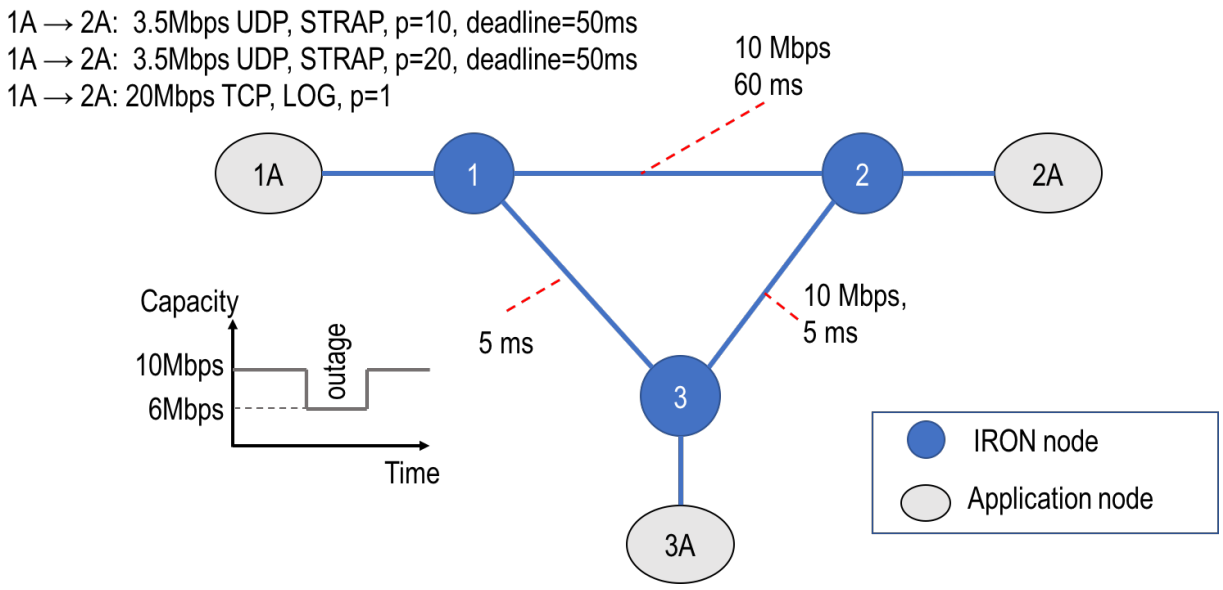
This experiment leverages and tests the following components and features:

- CAT operation for sending traffic.
- TCP proxies handling LOG-utility admission.
- UDP proxies handling STRAP-utility admission, including turning flows on or off in response to supervisor control in AMP.
- UDP proxies detecting missing or late packets at the destination and reporting them to the source UDP proxy.
- UDP proxies relaying packet acknowledgements to AMP
- BPF routing packets will deadlines on paths that can satisfy the deadline requirements (if possible).
- AMP turning flows off or on in response to losses reported by the UDP proxy.

4.26.1 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below, where the application node in enclave 1 sends three flows to the application node in enclave 2. There are two paths to the destination enclave 2: a direct 10Mbps path (from enclave 1 to enclave 2) with 60ms of propagation delay, and an indirect 10Mbps path (from enclave 1 to enclave 3 to enclave 2) with 10 ms of propagation delay (5 ms on each link). At approximately 20 seconds into the experiment the capacity of the link between enclave 1 and enclave 3 is reduced to 6Mbps and it is restored to 10 Mbps after another 20 seconds.

Source enclave 1 sends two UDP flows, each sourced at a rate of 3.5Mbps, with STRAP inelastic utility and one TCP flow, sourced at 20Mbps, with LOG utility. The inelastic flows have a deadline of 50 ms, so they cannot use the direct path.

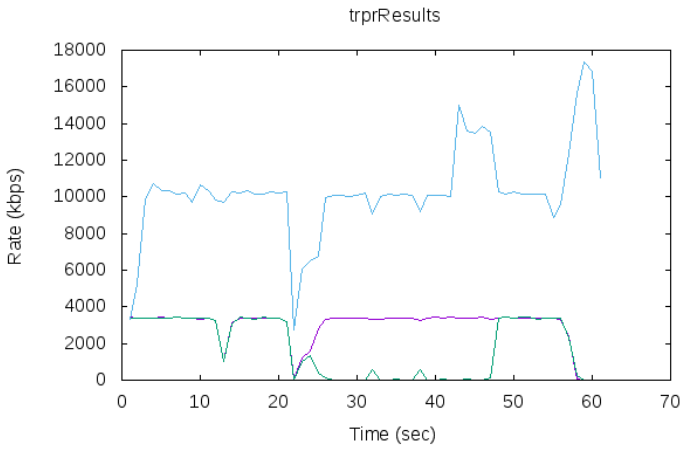


4.26.2 Expected Results

Initially, both inelastic flows will be able to use the indirect path and meet the deadline. When the bandwidth of the link between enclave 1 and enclave 3 is reduced to 6 Mbps, both flows will not be able to meet the deadline. There is still enough total capacity for both flows to be admitted, but packets taking the high-delay route will not meet the deadline. AMP should quickly detect these losses based on feedback from the destination and should turn off the lower priority STRAP flow. The remaining inelastic flow should be routed along the low-delay indirect path, which has sufficient capacity for that flow.The higher priority inelastic flow should thus be properly serviced with no packet loss or late packets. There should be periodic probing of the lower priority inelastic flow, but it should not fully restart. After 20 seconds, the capacity is restored and both inelastic flows should be on.

4.26.3 Observed Results

The observed goodput results (see figure below) are as expected. All three flows are initially on. One of the inelastic flow is triaged when the bandwidth of the indirect path is reduced, since the low-delay path can no longer support both latency-sensitive flows. When the bandwidth is restored, the second inelastic flow turns on and stays on.



4.26.4 Conclusion

This experiment demonstrates the ability to detect and react to latency-based packet losses in the network, even in cases with remote bottleneck links. This is particularly important with inelastic flows that get no utility if the loss rate is greater than some threshold

4.27 3-node-latency-thrash

4.27.1 Goal

This goal of this experiment is to test the supervisory control feature in the GNAT admission planner. In particular, this tests that supervisory control will triage low latency flows if they cannot be delivered on time.

Queue-based admission control does not account for flow deadlines and without supervisory control flows can be admitted even if there is no path or insufficient bandwidth that meets their latency requirements. Admitting these flows wastes of network resources, since these flows will never make it to their destination within the deadline. To counter this, the UDP proxy uses loss rates reported by the destination in RRMs to temporarily triage flows and AMP selects chooses to restart some of these flows and turn others off. The goal is to a have single thrashing flow to each destination, allowing the node to quickly take advantage of increases in bottleneck capacity and to probe the available capacity at different latency constraints by rotating the probes.

The Link State Announcement (LSA) mechanism used within GNAT propagates per-hop latency information across the network. The admission planner (AMP) uses this information to calculate the lowest latency of all paths on each CAT. Combining this information with the capacity estimate of the CATs, AMP determines whether it is possible to fit all flows in a way that meets deadlines. The measured capacity is an upper bound, since nodes only have local capacity information. Therefore, this approach does not account for remote bottlenecks. Loss-based triage is used to deal with these conditions, and is tested in the 3-node-loss-triage experiment.

This experiment leverages and tests the following components and features:

- CAT operation, including capacity and latency estimates.
- TCP proxies with LOG-utility admission.
- UDP proxies with STRAP-utility admission, including turning flows on or off in response to AMP.
- BPF relaying latency-sensitive packets on paths that can satisfy the deadline requirements.
- BPF relaying capacity and latency estimates to AMP.
- AMP turning flows off or on based on the estimates of capacity and latency from the BPF.

4.27.2 Topology and Conditions

The experiment runs over a 3-node triangular topology as shown in the figure below, where the application node in enclave 1 sends multiple flows to the application node in enclave 2. There are two paths to the destination enclave 2: a direct 10Mbps path (from enclave 1 to enclave 2) with 10 ms of propagation delay, and an indirect 10Mbps path (from enclave 1 to enclave 3 to enclave 2) with 60 ms of propagation delay (30 ms on each link). There will be additional queuing and transmission delays. At approximately 20 seconds into the experiment the capacity of the link between enclave 1 and enclave 2 is reduced to 5Mbps. The capacity is increased to 9Mbps after a further 40 second, and then reduced to 5Mbps after another 20 seconds.

There a TCP flow from enclave 1 to enclave 2 for the duration of the experiment.

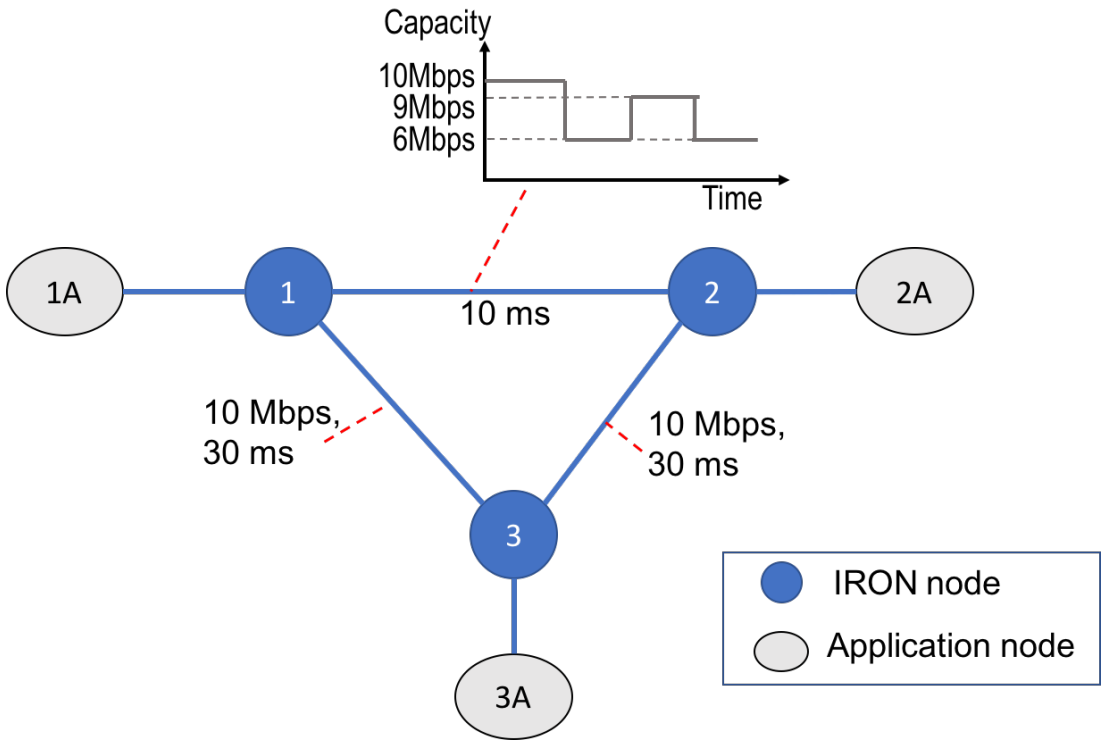
There is initially two UDP flows, with STRAP utility, from enclave 1 to enclave 2, each sending with an average rate of 3Mbps and with an initial TTG=60ms, both with priority 8.

At 30 seconds into the experiment, we start to add more flows with different rates and different

deadlines, as specified below:

Flow	Start time	Rate	TTG	Prority
0	10s	TCP	N/A	1
1	10s	3Mbps	60ms	8
2	10s	3Mbps	60ms	8
3	30s	2.5Mbps	60ms	8
4	35s	200Kbps	60ms	8
5	38s	600Kbps	60ms	8
6	40s	2Mbps	120ms	8
7	50s	400Kbps	60ms	8
8	52s	3Mbps	60ms	8
9	75s	1.5Mbps	120ms	8

The topology is shown in the figure below.



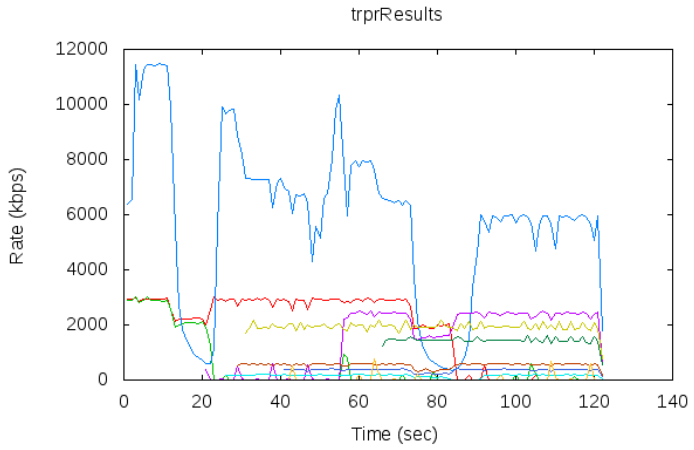
4.27.3 Expected Results

- Initially all 3 flows should fit. The low latency flows have a deadline of 60ms and should only be able to meet this deadline on the direct path. The capacity of the direct path is 10 Mbps and each flow is 3 Mbps. The TCP flow will use all the capacity on the indirect path and the residual capacity on the direct path.
- When the capacity of the direct path is reduced to 5Mbps, both low latency flows will start experiencing loss as there is not enough capacity to deliver 6Mbps of traffic within 60ms. This results in one of the UDP flows being triaged (randomly), while the other will persist. The TCP flow will use all the capacity on the indirect path and the residual capacity on the direct path.

- Flow 3 should not be admitted as there is less than 2Mbps of available capacity on the direct path, and the indirect path will not deliver the packets in time.
- Flows 4 and 5 should be admitted as they can fit on the direct path.
- Flow 6 should be admitted as it has a longer TTG and can fit on the indirect path.
- When the capacity on the direct path is increased, then Flow 3 should fit and it should stabilize once it starts probing.
- Flow 7 should fit.
- Flow 8 should not fit, as there is not enough available capacity to on the direct path.
- Flow 9 should be admitted as it has a higher TTG and can fit on the indirect path.
- The capacity on the direct path is again reduced to 5Mbps, Flow6 should be triaged and the other flows should remain on.
- Although there are multiple flows that cannot fit for the latter half of the experiment, there should be a single thrashing probe (selected by AMP), acting as a probe.

4.27.4 Observed Results

The observed goodput results (see figure below) are as expected. All three flows are initially on. When the bandwidth of the direct path was reduced one of the inelastic flows was triaged. The subsequent flows are admitted or rejected as expected.



4.27.5 Conclusions

This experiment demonstrates the ability to triage flows based on reported loss and use probes to gauge available capacities.

4.28 3-node-thrash

4.28.1 Goal

This experiment demonstrates thrash reduction in AMP when there are remote bottlenecks in an oversubscribed network.

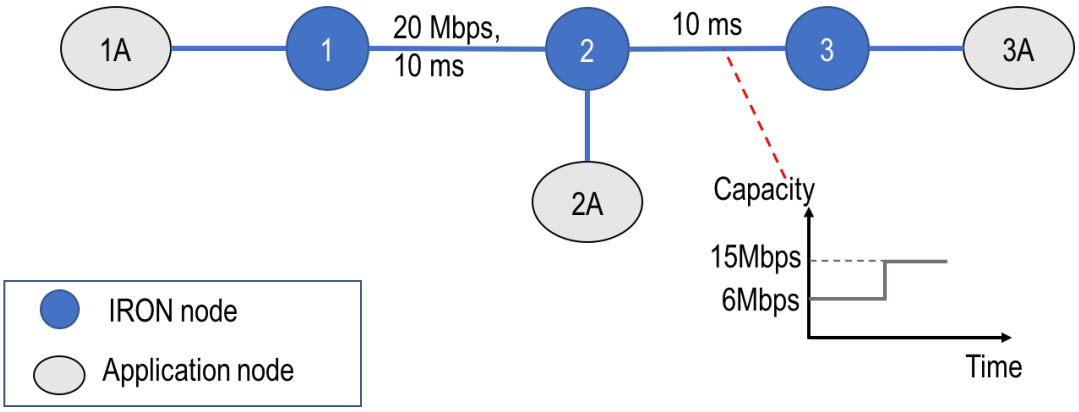
This experiment leverages and tests the following components and features:

- CAT operation,
- UDP proxies for STRAP utility admission and flow prioritization,
- TCP proxies for LOG utility admission and flow prioritization,
- BPF multi-path forwarding and broken-link avoidance.

4.28.2 Topology and Conditions

We consider a 3-node linear topology as shown in the figure below. The first hop is a 20Mbps capacity, and the second hop starts at 6Mbps and increases to 15Mbps partway through the experiment.

1A → 3A: 20x 1Mbps UDP, STRAP, p=3
1A → 3A: 2x 1Mbps UDP, STRAP, p=4
1A → 3A: 25Mbps UDP, LOG, p=1
1A → 2A: 25Mbps TCP, LOG, p=1
2A → 1A: 25Mbps TCP, LOG, p=1



There is elastic background traffic between enclave 1 and enclave 2 in each direction, and from enclave 1 to enclave 3. This traffic has a priority of 1.

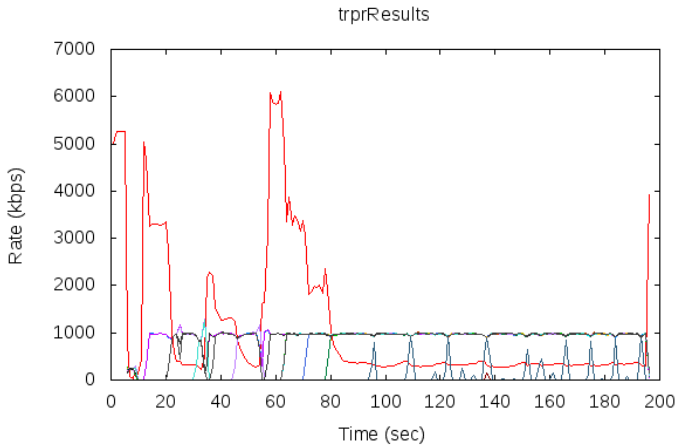
There are 22 inelastic (STRAP) UDP flows from enclave 1 to enclave 3. All of these flows have a nominal rate of 1Mbps. Of these flows 2 are priority 4 and the remaining 20 are priority 3.

4.28.3 Expected Results

Initially, there should be enough capacity for the two high priority inelastic flows, two low priority flows, and the elastic flows will use the remaining capacity. When the capacity of the bottleneck link is increased, there will be room for 8 additional inelastic flows. Once the flows turn on, they should stay on, except for at most one inelastic flow which acts as a probe and may thrash while the queue is stable.

4.28.4 Observed Results

The observed results are shown in the plot below. We see that inelastic flows are turning on when the queues are decreasing or stable *(i.e. the rates for the red elastic flow is increasing or steady). Then the capacity is increased at around 60 seconds into the experiment the elastic flow spikes up and we add inelastic traffic until there is one that cannot fit. The steady rate of the elastic traffic indicates that there are no thrashing flows.



4.28.5 Conclusions

This experiment demonstrates thrash reduction in the face of remote bottleneck links.

4.29 6-node-thrash

4.29.1 Goal

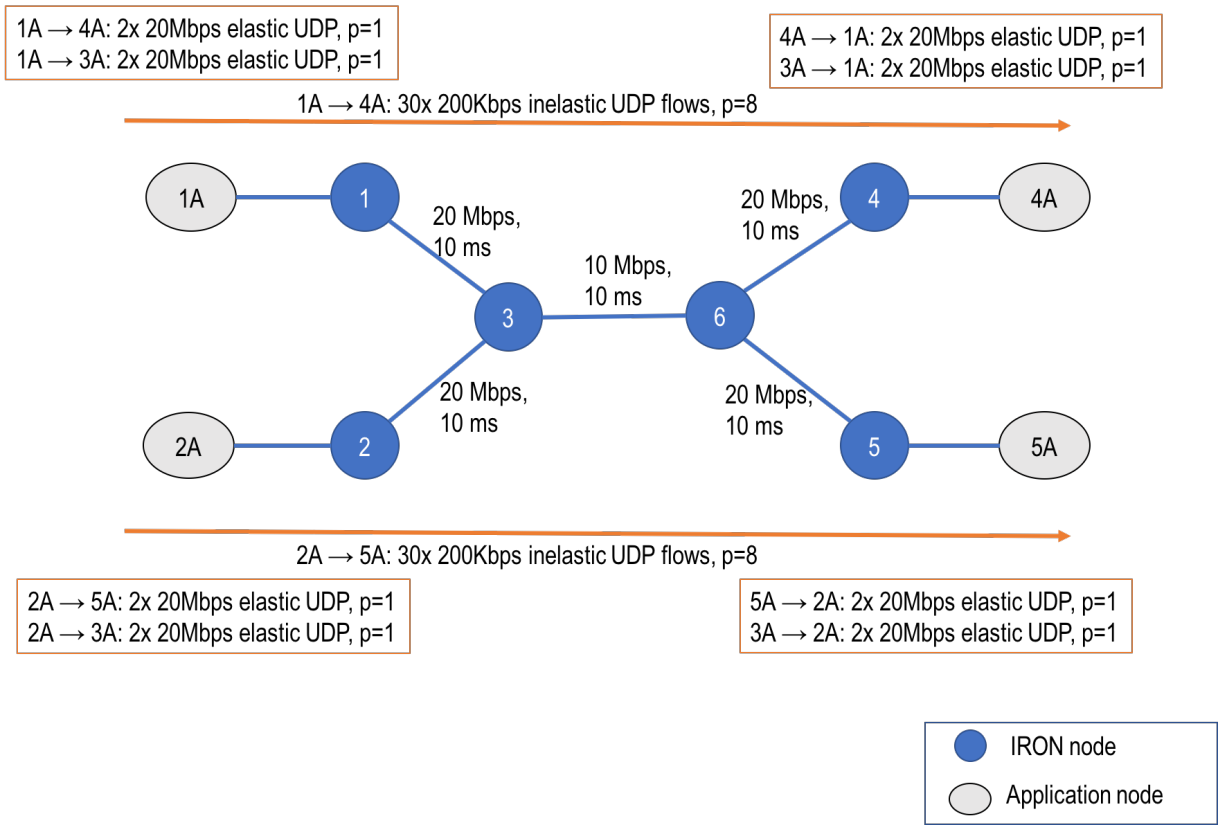
This experiment demonstrates distributed thrash reduction in AMP when there are remote bottlenecks in an oversubscribed network, where there are multiple nodes using the same bottleneck link for inelastic traffic.

This experiment leverages and tests the following components and features:

- CAT operation,
- UDP proxies for STRAP utility admission and flow prioritization,
- TCP proxies for LOG utility admission and flow prioritization,
- BPF multi-path forwarding and broken-link avoidance.

4.29.2 Topology and Conditions

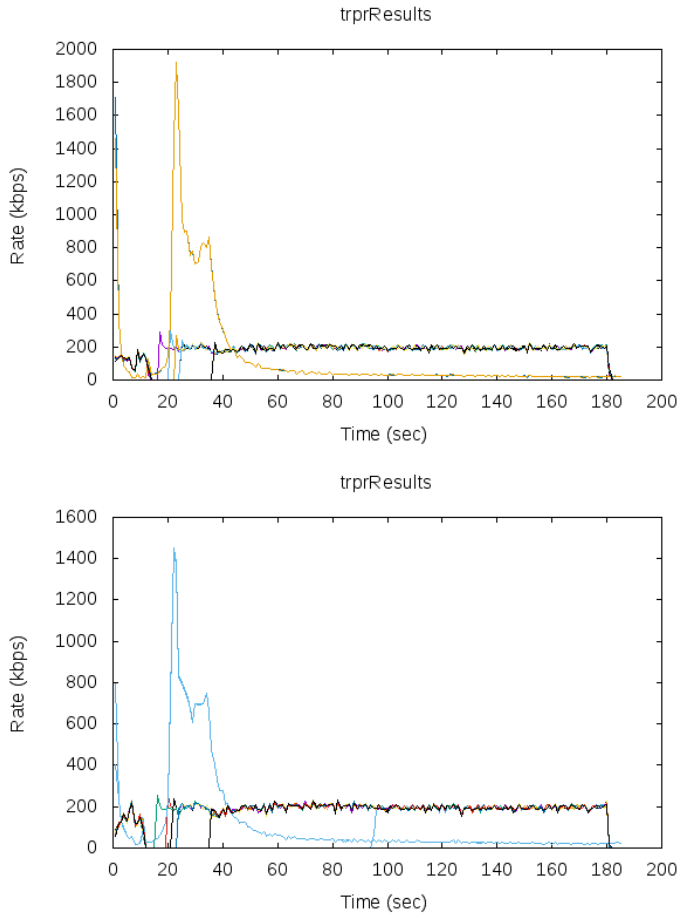
We consider a 6-node bottleneck topology where enclave 1 and enclave 2 source traffic for enclave 4 and enclave 5 respectively. The link between enclave 3 and enclave 6 is the bottleneck. This topology is shown in the figure below.



There are 30 flows with inelastic utility from enclave 1 to enclave 4, and another 30 from enclave 2 to enclave 5. All of these flows are being sourced at a rate of 200Kbps and have been assigned equal priority. There is a single 10Mbps path that must be shared among all the inelastic flows. There are also background elastic flows running at a lower priority between 1 and 3, 1 and 4, 2 and 3, and 2 and 5.

4.29.3 Expected results

The resulting goodput at enclaves 4 and 5 are shown in the plot below. Initially, all the flows are on and there isn't enough capacity so they all gets triaged. As the queue drains we add more flows, but the queues drain faster than each new flow which results in adding more flows between 20-45 seconds into the experiment. At this point the queue begins to build and we see the goodput of the elastic flow begin to drop. At this point, when we add a flow, the queue builds up slowly (at the rate of the new flow added ~200Kbps). Only when the queue stabilizes we can attempt to add another flow. We see that most flows that fit are enabled in a short period of time, but the last few flows that fit take a longer period of time. If we attempt to add flows before the queue stabilizes we can add too many that don't fit and this would cause many more flows to be triaged.



4.29.4 Conclusions

This experiment demonstrates thrash reduction in a more complex network, where there is competing nodes for the same network bottleneck.

4.30 4-node-dist-triage

This experiment tests the supervisory control feature in the GNAT admission planner.

Supervisor control is designed to triage flows which are not being properly serviced due to loss or late packet arrival at the destination. Even if an inelastic flow is admitted by the UDP proxy at the rate it is being sourced, it can get no utility if there are too many packets that arrive too late for the latency requirements. In this case, the bandwidth that it is using will be wasted. Supervisory control keeps track of the packets admitted by the source and those acknowledged by the destination as being received on time and uses this information to determine whether a flow is being properly serviced. Any flows not being properly serviced are triaged, one at a time, until the remaining flows are meeting their latency constraints. This is particularly important in scenarios with remote bottleneck, where it is difficult to measure or estimate the available capacity that can meet a latency requirement.

This experiment tests that flows are properly triaged if and only if latency requirements cannot be met, and demonstrates distributed decision making between AMPs without explicit coordination. There are two sources, that share a common bottleneck, and they must make decisions to not admit flows that cannot fit while sharing the available bandwidth.

This experiment leverages and tests the following components and features:

- CAT operation for sending traffic.
- TCP proxies handling LOG-utility admission.
- UDP proxies handling STRAP-utility admission, including turning flows on or off in response to supervisor control in AMP.
- UDP proxies detecting missing or late packets at the destination and reporting them to the source UDP proxy.
- UDP proxies relaying packet acknowledgements to AMP
- BPF routing packets will deadlines on paths that can satisfy the deadline requirements (if possible).
- AMP turning flows off or on in response to losses reported by the UDP proxy.

4.30.1 Topology and Conditions

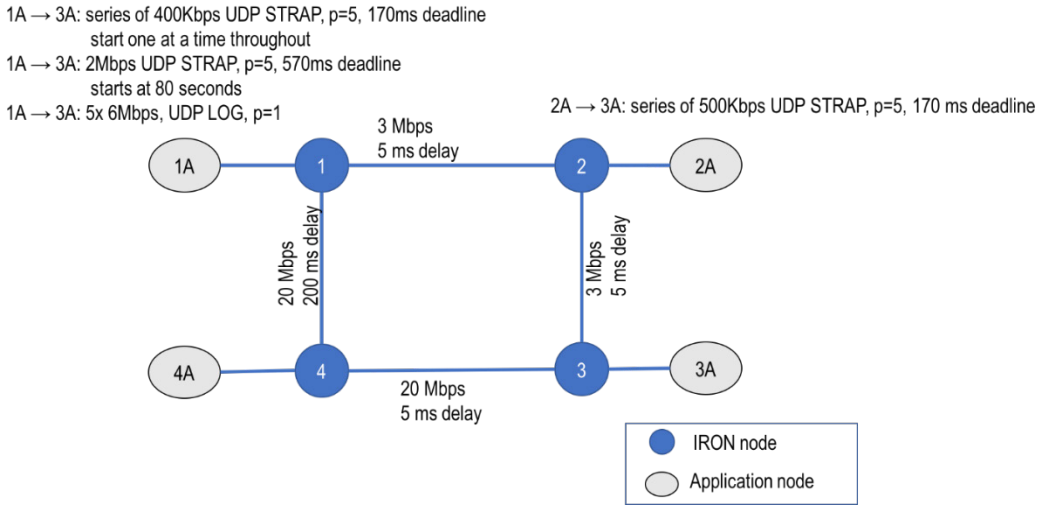
The experiment runs over a 4-node square topology as shown in the figure below.

- There is a low latency path (10ms + queuing delays) from enclave 1 to enclave 3, via enclave 2
- There is a higher latency path (205ms + queuing delays) from enclave 1 to enclave 3, via enclave 4

Both enclave 1 and enclave 2 source traffic destined for enclave 3. Most of this traffic (multiple flows) have a TTG of 170ms, which means it cannot take the higher latency path.

- There are multiple flows from enclave 1 to enclave 3, sourced at a rate of 400Kbps and TTG=170ms (priority 5)
- There are multiple flows from enclave 2 to enclave 3, sourced at a rate of 500Kbps and TTG=170ms (priority 5)

- At around 80 seconds into the experiment a 2Mbps flow from enclave 1 to enclave 3 is started with a TTG of 570ms (priority 5)
- Throughout the experiment, there are 5 priority 1 elastic UDP flows from enclave 1 to enclave 3.



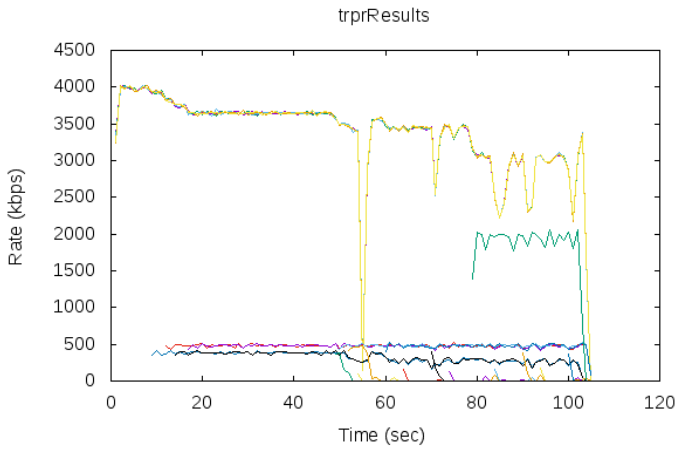
4.30.2 Expected Results

Initially, some flows from enclave 1 and enclave 2 should fit and remain on. Once the link between enclave 2 and enclave 3 is saturated, additional flows will not fit and they should be triaged soon after they start.

The flow from enclave 1 to enclave 3 with TTG=570ms should fit on the path through enclave 4, so this flow should stay on.

4.30.3 Observed Results

The observed goodput results (see figure below) are as expected. Most of the flows are in one of two groups: at 500Kbps and 400Kbps based on the origin. We see that flows from both sources are sustained throughout the experiment. We see that these flows stay on, and new flows are triaged with the exception of the 2Mbps flow (which has a higher deadline).



4.30.4 Conclusion

This experiment demonstrates the ability to detect and react to latency-based packet losses in the network, even in cases with remote bottleneck links and multiple sources.