Applying TCP Congestion Control Schemes on NDN and their Implications for NDN Congestion Control

Shuo Yang

Abstract

While congestion control in TCP has been well studied, the future Internet architecture NDN has posed new challenges for congestion control, given its new features such as *receiver-driver, one-Interest-one-Data, in-network-caching, multi-path and mult-source*, etc. The current research on congestion control in NDN mainly focuses on how to take advantage of these new features with the help from the network. However, this puts more burden on the network layer and further complicates the network. So far, there is no comprehensive study on what implications does congestion control in TCP have on NDN? In this project, we will apply both loss-based and delay-based TCP congestion control schemes to NDN and study their effectivenss for NDN. Further, we expect we can learn from this experience and come up with better receiver-based congestion control design for NDN.

Introduction

Congestion control is a hot research topic in current NDN research community. The current approaches can be catagorized into three catagories [1]: 1) receiver-based control; 2) hop-by-hop control; 3) hybrid method. Receiver-based control works by adjusting Interest sending rate on the reciver side based on RTO values caculated for Interest packets. Hop-by-hop control relies on intermediate nodes to detect congestion by observing queue size or PIT size and adjust Interest forwarding rate accordingly. Hybrid method is the combination of the 1) and 2). In this work, we will focus on receiver-based congestion control.

Receiver-based congestion control has the benefit of high scalability, easy deployment and testing, as it doesn't rely on additional information from the network. In addition, we can learn from most TCP and TCP-friendly congestion control schemes in which end-hosts are responsible for congestion detection and avoidance (just in TCP, this role is played by senders, while in NDN, the role is played by receiver).

However, it is quite challenging for a pure receiver-based congestion control because 1) congestion detection based on RTO is not reliable due to connectionless, multi-source and multi-path features introduced in NDN; 2) congestion detection based on three duplicate ACKs does not work for NDN; 3) a single congestion window cannot adapt to multi-source and multi-path transfer in NDN.

We argue that even with these limitations, it is still worth studying what implications does congestion control in TCP have on NDN for the following reasons.

1. In NDN, one Interest corrpesonds to one Data, which follows the reverse path of the Interest, thus makes RTT a more accuarate measurement of the network capacity bewtween consumer and the data source. Consumer should take advantage of RTT measurement.

2. For applications like real time video chatting between two persons, regardless of the location, it's essentially a point to point communication between a single consumer and a single producer, since content must be generated by producer in real time and consumed by a single consumer. In such cases, congestion control mechanisms in TCP is still useful.

3. For the foreseeable future, NDN traffic will co-exist with TCP traffic, thus developing TCP friendly congestion control for NDN is important for flow fairness among Internet traffic.

Previously, we have studied TCP Reno [2] and TCP Cubic [3] and implemeted similar algorithms in NDN's file transfer application `ndnchunks`. Both Reno and Cubic are loss-based congestion schemes, meaning that they use packet loss as congestion signal and react to congestion only after loss occurs. There is another interesting delay-based congestion control scheme for TCP, i.e., TCP Vegas [4], in which sender uses packet delay, rather than packet loss, as a signal to determine the rate at which to send packets. TCP Vegas detects congestion at an incipient stage based on increasing RTT values of the packets in the connection. The algorithm depends heavily on accurate calculation of the Base RTT value. If it is too small then throughput of the connection will be less than the bandwidth available while if the value is too large then it will overrun the connection.

We believe TCP Vegas is relevant to NDN because without routers' cooperation, RTT is the only information consumers in NDN could use to infer the changing condition of the network. In this project, we will implement a TCP

Vegas-like algorithm in `ndnchunks` and evaluate its performance together with Reno and Cubic implementations. Based on the evaluation, we hope to come up with a new receiver-driven congestion control that is more adaptive to multi-source and multi-path situation arised in NDN.

TCP Vegas Overview

TCP Vegas is a novel (at the time it's proposed) approach to TCP congestion avoidance that emphasizes packet delay, rather than packet loss, as a signal to help determine the rate at which to send packets. Its key techniques include:

1. *fast retransmission*: 1) upon receiving duplicate ACK, check if timeout expired and if so, retransmit. It differs from TCP Reno which only retransmits upon seeing three duplicate ACKs. 2) upon receiving a non-duplicate ACK (1st or 2nd after retransmission), check if timeout expired and if so, retransmit.
2. *congestion window decreases only once during one RTT*, unlike Reno, which is possible to decrease the cwnd more than once during one RTT.
3. *proactive congestion avoidance*: try to detect incipient stages of congestion by comparing the measured throughput to the expected throughtput.
4. *modified slow start*: unlike Reno, which doubles cwnd every RTT, Vegas doubles cwnd *every other* RTT so that it can make valid comparsion of the expected and the actual throughtput.

Parameters used for Congestion Avoidance Algorithm:

- *baseRTT*: RTT of a segment when the flow is not congested, Vegas sets *baseRTT* to the minimum of all measured round trip times; it is commonly the RTT of the first segment sent by the connection.
- *cwnd*: the size (in segments) of the current congestion window.
- $expected = \frac{cwnd}{baseRTT}$, this is the expected throughput assuming the connection is not congested.
- *rttLen*: number of segments trasmitted during the last RTT.
- *avgRTT*: average RTT of the segments acknowledged during the last RTT.
- $actual = \frac{rttLen}{avgRTT}$, this is the measured current throughtput.
- $diff = expected - actual$, $diff$ is always ≥ 0.
- two thresholds: $\alpha < \beta$, where α triggers window increase(linear) and β triggers window decrease(linear). Vegas sets $\alpha = 1$ and $\beta = 3$. This can be interpreted as an attempt to use at least one, but no more than three extra buffers in the connection.

Algorithm 1 TCP Vegas Congestion Avoidance Algorithm

```
1: procedure CongestionAvoidance()
2:     if $diff < \alpha$ then increase $cwnd$ linearly during the next RTT
3:     if $diff > \beta$ then decrease $cwnd$ linearly during the next RTT
4:     if $\alpha \leq diff \leq \beta$ then keep $cwnd$ unchanged during the next RTT
5:
```

Intuitively, the farther away the actual throughput gets from the expected throughput,the more congestion there is in the network, which implies that the sending rate should be reduced. On the other hand, when the actual throughput rate gets too close to the expected throughput, the connection is in danger of not utilizing the available bandwidth.

Considerations for Adapting TCP Vegas to NDN

1. *Changing baseRTT* due to multi-source and multi-path transportation introduced in NDN. Content retrieved from different sources might experience different RTTs, thus NDN must be able to adapt to different *baseRTT* timely to avoid false detection of congestions.

2. *Is fast retransmission necessary in NDN?* Fast retransmission is the key to TCP congestion control because of its cumulative acknowledgement. But in NDN, each *Data* packet is an explict acknowledgement of its corresponding *Interest* packet, thus out-of-order data delivery is accepted (maybe normal) in NDN. Moreover, because of Interest aggregation feature of PIT, retransmitting an Interest that is still kept in PIT has no effect in retrieving corresponding Data due to Interest aggregation until that Interest lifetime expires. Even worse, unlike TCP, which can use duplicate ACK as a sign of congestion, loss detection in NDN is quite unreliable due to multi-source and multi-path transportation. Often time, Data just got delayed but not lost.

Adapting TCP Vegas to NDN

The following techniques are proposed to adapt the idea of TCP Vegas to NDN:

1. *baseRTT adaption*: beyond two thresholds α and β, introduce another threshold γ where $\alpha < \beta < \gamma$. 1) if $diff > 0$ and $diff \leq \gamma$, do the same as TCP Vegas. This is the case where we might be retrieving data from a constant source. 2) if $diff < 0$, update *baseRTT* and start congestion avoidance with the updated *baseRTT*. *actual* $>$ *expected* implies we need to update *baseRTT* with the latest sampled RTT (samller value than the current *baseRTT*). In this case we are likely retrieving data from a different source that is closer, thus with smaller *baseRTT*. 3) if $diff > 0$ and $diff > \gamma$, update *baseRTT* with the latest sampled RTT (greater value than the current *baseRTT*) and start congestion avoidance with the updated *baseRTT*. $diff > \gamma$ suggests that we might be retriving data from a different source with is farther away, thus with greater *baseRTT*.

2. *postponed retransmission*: for the Interests that are possibly lost as determined by consumer, don't immediately retransmit them, instead put in the retransmission queue and only retransmit if 1) all the in-order segments have been transmitted, or 2) Interest lifetime expires. Therefore, we prioritize the transmission of in-order segments over those needed retransmission. By doing this we give those Data that might get delayed in the network some time to arrive, we expect this technique will significantly reduce the number of retransmissions.

3. *slow start*: we can 1) apply the same slow start technique used by TCP Vegas or 2) apply Reno's slow start method while using RTO to detect congestion or 3) apply Reno's slow start method while using packet hole (out-of-order delivery) as a sign of congestion to stop slow start.

Algorithm 2 NDN Vegas Congestion Avoidance Algorithm

```
1:  procedure CongestionAvoidance()
2:      if $diff > 0$ and $diff \leq \gamma$ then                ▷ retriving content from constant source
3:          if $diff < \alpha$ then increase cwnd linearly during the next RTT
4:          if $diff > \beta$ then decrease cwnd linearly during the next RTT
5:          if $\alpha \leq diff \leq \beta$ then keep cwnd unchanged during the next RTT
6:      if $diff > 0$ and $diff > \gamma$ then                   ▷ retriving content from a farther source
7:          baseRTT ← the latest sampled RTT (larger)
8:      if $diff < 0$ then                                       ▷ retriving content from a closer source
9:          baseRTT ← the latest sampled RTT (smaller)
10:
```

Related Work

ICP [5] (Interest Control Protocol) adopts the AIMD window adjustment algorithm like TCP, it uses loss-based model similar with TCP Reno. [6] studied efficiency of receiver-based timeout-driven AIMD flow-control in CCN when multi-homing is enabled. it is also a loss-based approach.

NDN Vegas implementation in `ndncatchunks`

The source code can be found at https://github.com/imsure/ndn-vegas. The files corrpesond to the Vegas are `chunks/catchunks/pipeline-interests-vegas.hpp` and `chunks/catchunks/pipeline-interests-vegas.cpp`.

Performance Evaluation under Linear Topology

In this section we study the performance of Vegas pipeline against the two loss-based pipelines: AIMD and CUBIC.

Evaluation Environment:

Two ubuntu VMs running on a Macbook Pro (4 cores, 8GB memory), one runs a consumer, the other one runs a producer. They forward packets to each other through a UDP tunnel. The link between 2 VMs is shaped to have a specific RTT using the command:

```
$ sudo tc qdisc add dev eth1 root netem delay <time in ms>
```

Evaluation Methodology:

We run three pipelines independently (i.e., no competing flows) for each scenario described below for 5 times, intending to compare the performance and study the correlation between NDN Vegas performance and link RTT, as well as vegas parameters.

Evaluation Scenarios:

- Scenario 1: Transferring a 100MB file over a link of 50ms RTT
  - Scenario 1-1: $\alpha = 5$, $\beta = 10$ and $\gamma = \infty$
  - Scenario 1-2: $\alpha = 10$, $\beta = 20$ and $\gamma = \infty$
  - Scenario 1-3: $\alpha = 20$, $\beta = 30$ and $\gamma = \infty$
- Scenario 2: Transferring a 100MB file over a link of 100ms RTT
  - Scenario 2-1: $\alpha = 5$, $\beta = 10$ and $\gamma = \infty$
  - Scenario 2-2: $\alpha = 10$, $\beta = 20$ and $\gamma = \infty$
- Scenario 3: Transferring a 100MB file over a link of 200ms RTT
  - Scenario 3-1: $\alpha = 5$, $\beta = 10$ and $\gamma = \infty$
  - Scenario 3-2: $\alpha = 10$, $\beta = 20$ and $\gamma = \infty$
  - Scenario 1-3: $\alpha = 20$, $\beta = 30$ and $\gamma = \infty$

Results

Figure 1 shows that when link RTT is 50ms, the setting of the Vegas parameters $\alpha = 5$ and $\beta = 10$ is insufficient for a Vegas flow to utilize the link capacity. Thus we get rid of the Scenario 1-1.

Figure 2 and 3 are the plots for Scenario 1 (1-2 and 1-3), where Figure 1 shows the summary stats collected for all the 5 runs. The left-most one shows how much time it was taken to complete a 100MB file transfer for each flow. The middle one shows the number of retransmitted Interests for each flow. The right-most one shows the goodput for each flow. Figure 3 shows the evolution of congestion window size (cwnd) and transmission rate for each flow within each run. Figure 4 and 5 are the plots for Scenario 2, while Figure 6 and 7 are the plots for Scenario 3.

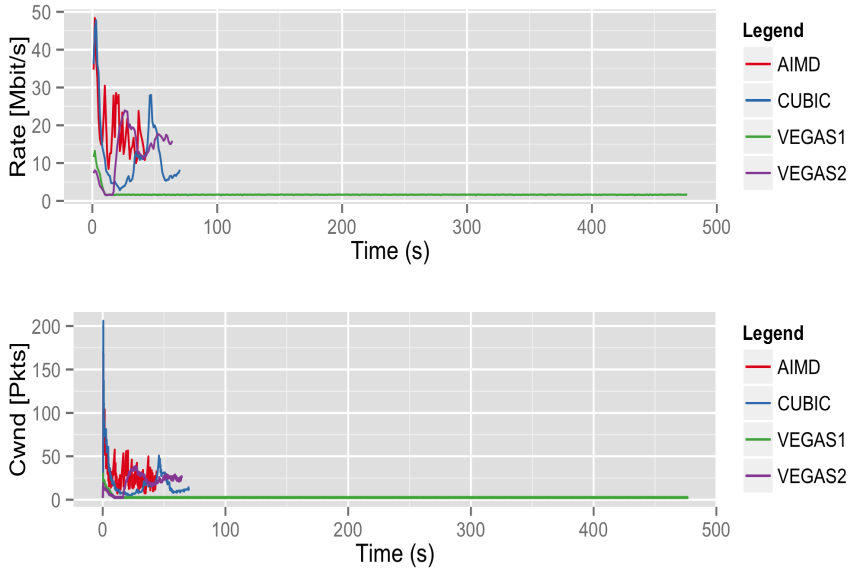


Figure 1: Plot for Scenario 1-1 (Vegas1: $\alpha = 5, \beta = 10$, Vegas2: $\alpha = 10, \beta = 20$). Vegas1 flow underutilized the link capacity .

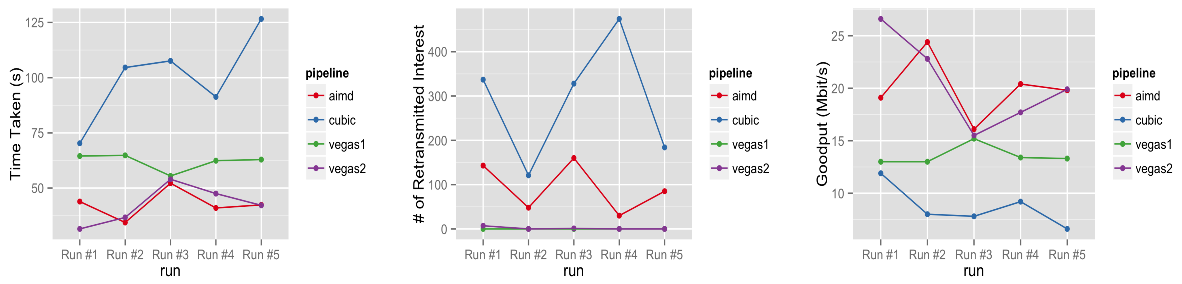
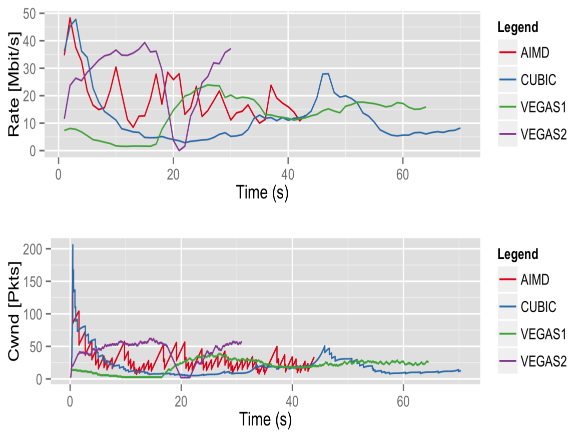
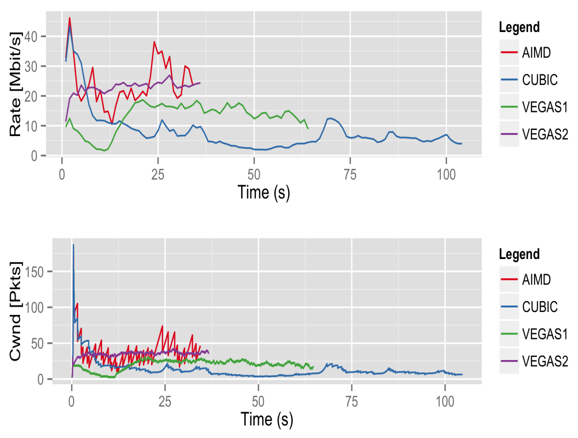


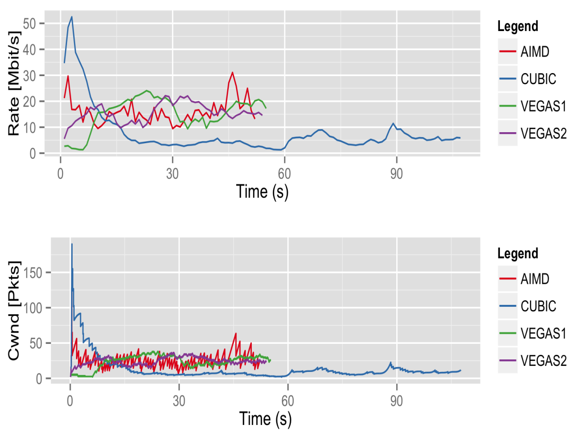
Figure 2: Summary stats plot for Scenario 1. (RTT=50ms, Vegas1: $\alpha = 10, \beta = 20$, Vegas2: $\alpha = 20, \beta = 30$)



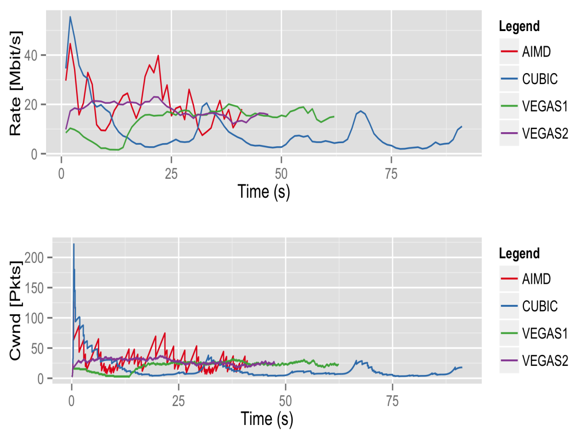
(a) Run #1



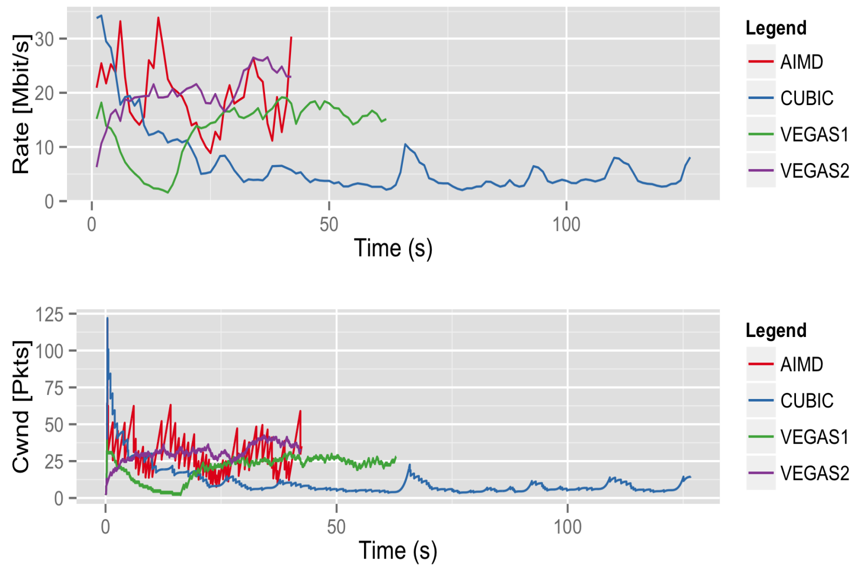
(b) Run #2



(c) Run #3



(d) Run #4



(e) Run #5

Figure 3: cwnd and rate plot for Scenario 1. ((RTT=50ms, Vegas1: $\alpha = 10, \beta = 20$, Vegas2: $\alpha = 20, \beta = 30$)

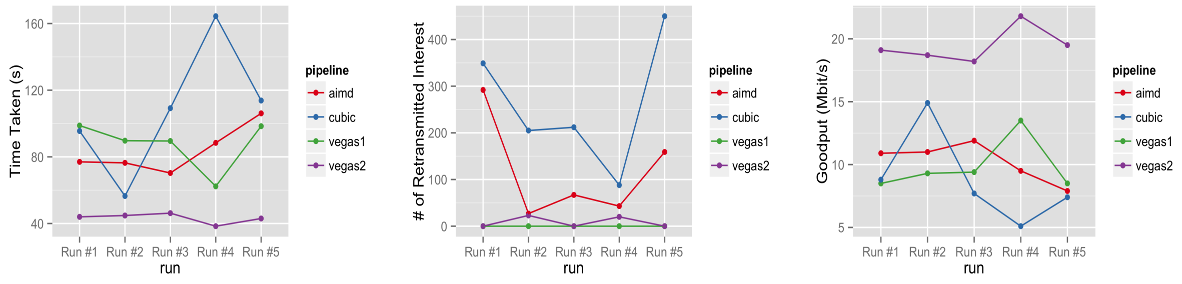
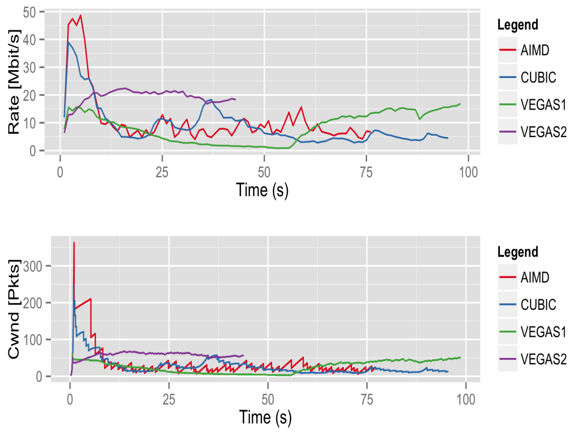


Figure 4: Summary stats plot for Scenario 2. (RTT=100ms, Vegas1: $\alpha = 5, \beta = 10$, Vegas2: $\alpha = 10, \beta = 20$)

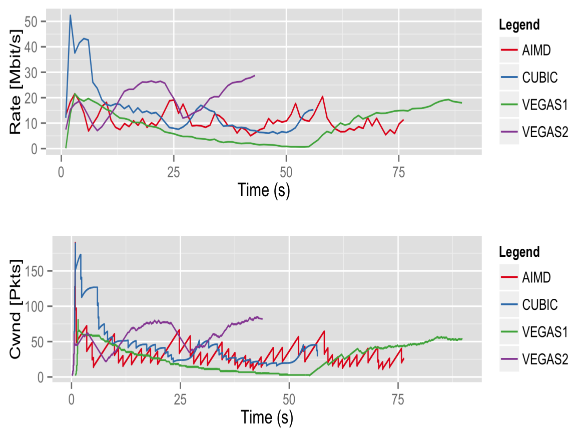
<u>Discussion</u>

There are a few observations based on the results we got.

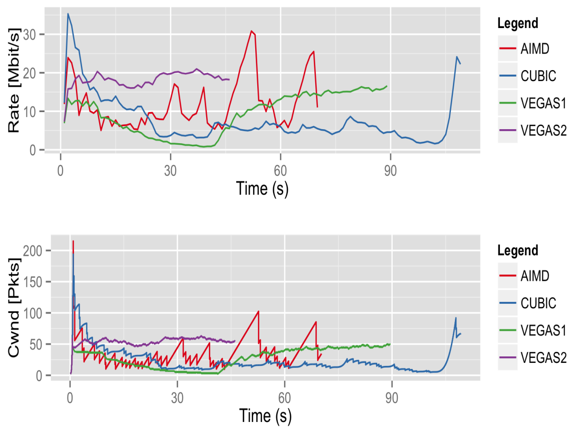
- Overall, Vegas flows with appropriate parameter settings perform better than AIMD and CUBIC flows. They took less time to complete, led to less retransmissions and achieved higher throughput. (refer to Vegas2 flow in Figure 2, 3, 4, 5, 6 and 7)
- AIMD flows favor connections with smaller RTTs. (if we look at Figure 2 where RTT=50ms, we see that AIMD is almost as good as Vegas2, except for retransmissions, but when we look at Figure 6 where RTT=200ms, the performance of AIMD is the worst). This is in accordance with RTT-bias of TCP-Reno flows, which also favor conenctions with smaller RTTs.
- Overall, CUBIC flows show the worst performance and instability. (refer to CUBIC flow in Figure 2, 3, 4, 5, 6 and 7, it only performs better than AIMD when RTT=200ms)
- Choosing appropriate parameters for Vegas is vital to its performance. (e.g., picking $\alpha = 5, \beta = 10$ for a vegas flow when RTT=50ms leads to very poor performance if we look at Vegas1 in Figure 1).
- The impact Vegas parameters has on performance is: the larger α and β, the more aggressive a Vegas flow is. (larger α means we increase congestion window more often, larger β means we decrease congestion window less often.) On the contrary, smaller α and β leads to underutilization of a Vegas flow as showed in Figure 1. Thus we need to strike a balance between overutilization (Vegas3 flow in Figure 6 shows such an example, where it leads to a higher number of retransmissions) and underutilization by picking appropriate Vegas parameters.
- There seems to be a correlation between the Vegas parameters and link RTT. From Figure 2 and 4, we see that $\alpha = 20, \beta = 30$ is good for RTT=50ms and $\alpha = 10, \beta = 20$ is good for RTT=100ms. For RTT=200ms, if we look at Figure 6, the three set of parameters don't show very big differences in terms of performance. But $\alpha = 5, \beta = 10$ seems to be more reasonable for it leads to less number of retransmissions (an ideal Vegas flow should take full use of link capacity while not overutilizing it). Thus it seems that the appropriate Vegas parameter values are inversely proportional to the RTT of links. Though it needs further study.



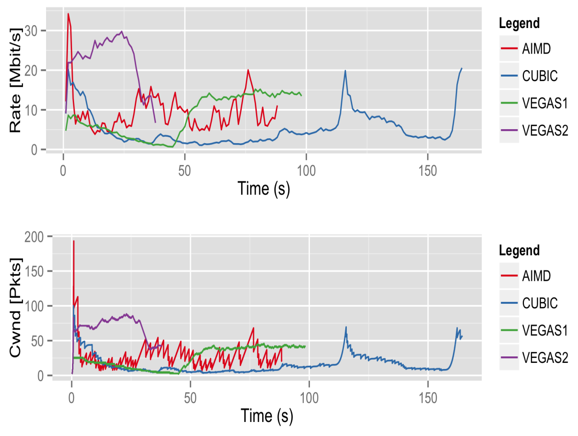
(a) Run #1



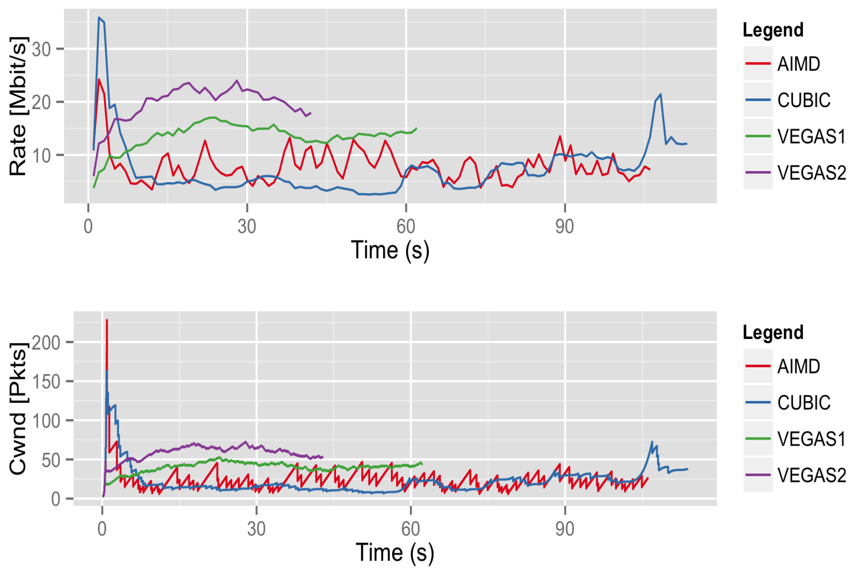
(b) Run #2



(c) Run #3



(d) Run #4



(e) Run #5

Figure 5: cwnd and rate plot for Scenario 2. (RTT=100ms, Vegas1: $\alpha = 5, \beta = 10$, Vegas2: $\alpha = 10, \beta = 20$)

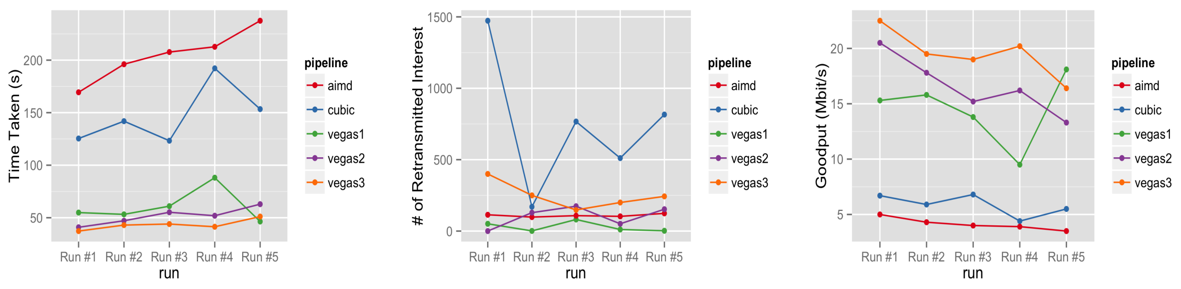


Figure 6: Summary stats plot for Scenario 3. (RTT=200ms, Vegas1: $\alpha = 5, \beta = 10$, Vegas2: $\alpha = 10, \beta = 20$, Vegas3: $\alpha = 20, \beta = 30$)

References

[1] Y. Ren, J. Li, S. Shi, L. Li, G. Wang, and B. Zhang, Congestion control in named data networking - A survey, Computer Communications, vol. 86 pp. 1-11, July 2016.

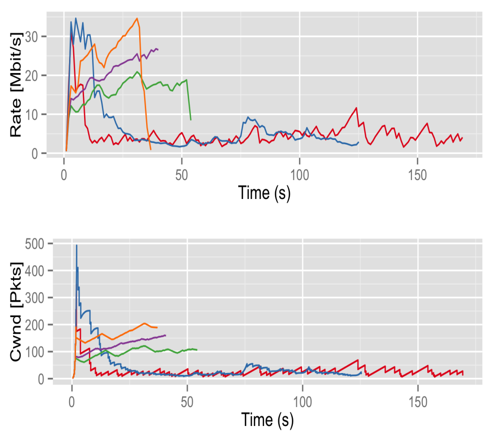
[2] V.Jacobson Modified TCP Congestion Control and Avoidance Alogrithms.Technical Report 30,Apr 1990.

[3] Sangtae Ha, Injong Rhee and Lisong Xu, CUBIC: A New TCP-Friendly High-Speed TCP Variant, ACM SIGOPS Operating System Review, Volume 42, Issue 5, July 2008, Page(s):64-74, 2008.

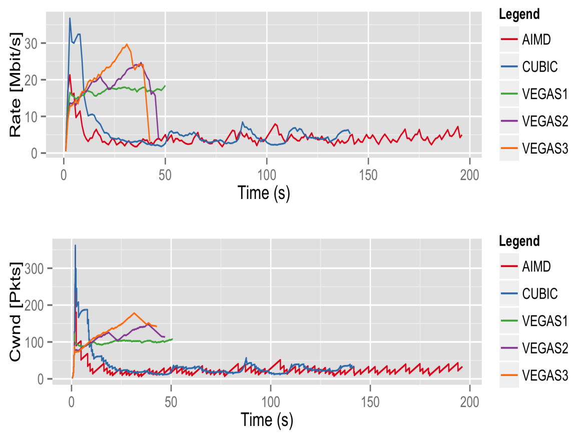
[4] Lawrence S. Brakmo and Larry L. Peterson - TCP Vegas: End to End Congestion Avoidance on a Global Internet.

[5] G. Carofiglio, M. Gallo, L. Muscariello, ICP: design and evaluation of an interest control protocol for content-centric networking, Proceeding of IEEE INFOCOM Workshop on Emerging Design Choices In Name Oriented Networking (INFO-COM NOMEN), 2012.

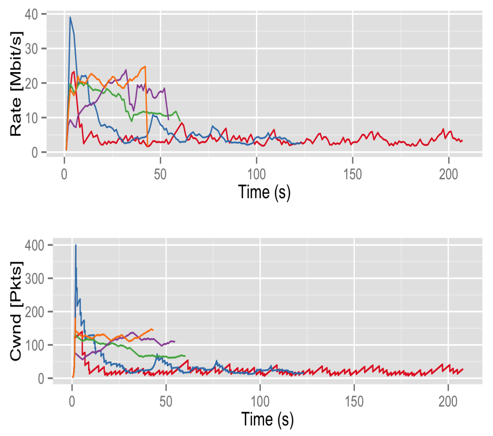
[6] S. Braun, M. Monti, M. Sifalakis, C. Tschudin, An empirical study of re-ceiver-based aimd flow-control strategies for ccn, Proceeding of IEEE ICCCN, 2013.



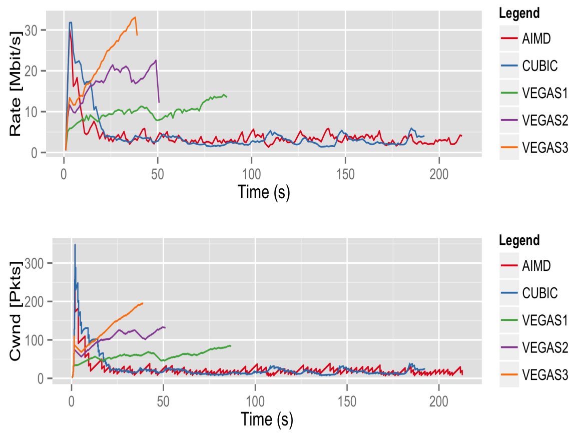
(a) Run #1



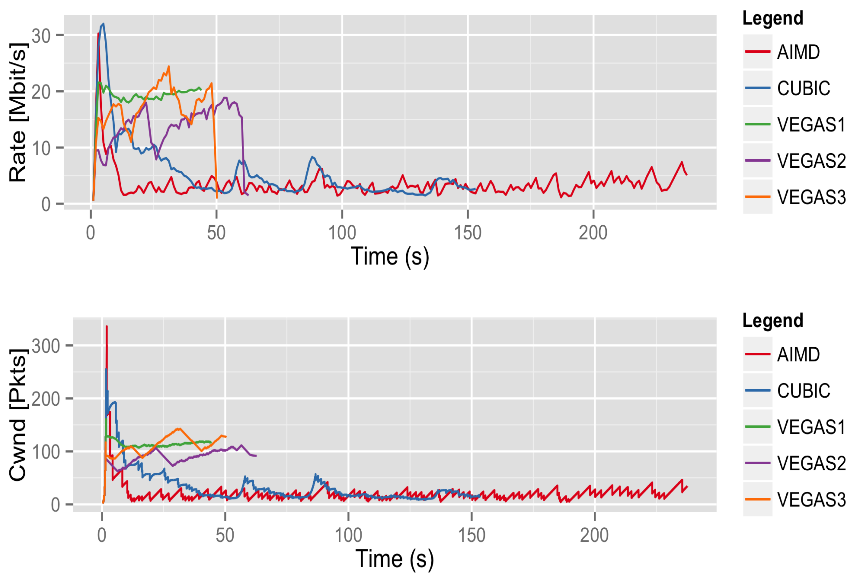
(b) Run #2



(c) Run #3



(d) Run #4



(e) Run #5

Figure 7: cwnd and rate plot for Scenario 3. (RTT=200ms, Vegas1: $\alpha = 5, \beta = 10$, Vegas2: $\alpha = 10, \beta = 20$, Vegas3: $\alpha = 20, \beta = 30$)