Marooned Magic Numbers – An Adaptive Congestion Control Architecture

Somaya Arianfar, Pasi Sarolahti, and Jörg Ott
Aalto University, Department of Communications and Networking
{firstname.lastname}@aalto.fi

***Abstract*—TCP and other Internet transport protocols rely on series of hard-coded initial values for their connection state, particularly for the congestion control parameters. For example, recently the initial value of congestion window has been under much debate, as there is a desire to make TCP more efficient for common use cases, while not endangering its performance on scenarios with limited network bandwidth. Our take on this discussion is that there is no clear single set of initial values that would work for all cases.**

Earlier research has proposed sharing connection and congestion control state among multiple connections over time, but that approach is limited to sharing connections to a particular host, which is not sufficient, because services are often distributed across multiple hosts, and opening multiple connections to the same host is a rather rare use case. We aim to solve this problem by proposing the Pathlet Transport Architecture that models the network paths as a series of pathlets, and uses those as the basis of initializing and maintaining the various transport parameters, particularly those related to congestion control. We analyze our initial instantiation of the PTA architecture using ns-3 simulations for TCP congestion control parameters, and show how it improves the communication performance in various different network scenarios, where single common set of magic values would fail.

I. Introduction

Internet congestion control has remained an active research topic since publication of the classic TCP Congestion Control paper in 1982 [11]. There have been countless variations of TCP's congestion control algorithms, often optimized for specific environment or specific class of applications (such as [2], [13], [5], [19], to give a small sample), not to mention the different ideas on more fundamental congestion control protocols or architectures, such as XCP and RCP [12], [6]. Despite various research projects, the standard mechanisms – as specified by the IETF and implemented by various systems – have remained fairly stable, and many systems still implement the classic NewReno-based algorithms.

After a decades-long stable state of affairs with TCP's congestion control, the IETF is currently working on modifications to some of the basic constants used by TCP. Among other changes, there is ongoing work to increase TCP's initial congestion control to 10 segments, commonly equivalent to some 15 KB worth of data. This is a fairly substantial change compared to the earlier situation where connections commonly start with initial window of 3 segments[1]. The initial congestion window of 10 packets is considered to be a significant performance improvement, because according to recent measurement data [7], such window would allow transfer of most of the current web objects within a single round-trip time. However, concerns have been raised about how such mechanism would affect users behind very poor and highly shared connections that are common in, for example, developing countries. Therefore, there has been active discussion on finding a more dynamic approach to adapt the initial congestion window, without considerable success so far.

The initial congestion window is not the only parameter hard-wired in the current standards track specifications that is under discussion. Also the retransmission timeout estimate is bounded by a minimal value of 1000 milliseconds[2], which is considered very long for many current network environments, even though it might still be appropriate, e.g., for high-delay wireless links. Also, initially setting the slow-start threshold (ssthresh) to infinity sometimes results in overshoot and several packet losses that could be avoided with a more appropriate setting of ssthresh.

The problem in a world with an increasing variety of network characteristics is that a single set of magic numbers, as used in TCP specifications and implementations, does not fit all possible environments. There will be increasing pressure on tuning these parameters for particular environments, which will likely be difficult even for advanced system maintainers, and a good common set of well-behaving parameters may sometimes be nearly impossible to find. A better approach would, therefore, be to develop the transport architecture towards avoiding any such hard-wired parameters, and instead applying dynamic methods for finding good initial settings.

In this paper, we propose a framework for automatically setting appropriate transport protocol and congestion control parameters already from the beginning of a connection, without relying on magic constants. We focus on the initial congestion window, slow-start threshold, and minimum RTO estimate, which were perhaps most often subject to past debate. By doing this, we can avoid the normal slow-start probing of the appropriate sending rate, and better utilize the available capacity from the outset. As noted above, this is also quite timely a topic for TCP standardization in the IETF.

We leverage the earlier idea of sharing state between connections, simultaneously and over time, as proposed by the Endpoint Congestion Manager [4], and in an IETF docu-

[1]The actual window size depends on the MTU used for communication.

[2]Some implementations use smaller constants in practice.

ment [17]. The idea of these proposals is, that a sender can initialize its congestion control state based on the recent history collected from other connections to the same destination.

The problem with the past proposals of connection state sharing is that it is difficult to pick the correct groups of hosts that have common path characteristics. Storing separate state for each host is not feasible even in modern hosts, given the large numbers of Internet hosts today. Often, several hosts share the same bottleneck, meaning that it would be useful and more effective to share a common state between all of them. A rough approach could be to share state between hosts that share a common IP address prefix. However, it is difficult to choose an ideal prefix length, because different networks come with different sizes, and it is hard to know where the communication bottlenecks are. Moreover, IP layer mobility may entirely invalidate such assumptions. And in case of a wireless cellular terminal the network bottleneck is likely on the first-hop wireless link, in which case it would be sensible to share a common congestion control state between all connections on the wireless terminal. The problem is that in a generic OS instance (such as in a laptop connected to a mobile terminal) the operating environment is not known, and the OS instance cannot determine in advance where the likely bottleneck is and what the correct granularity of state sharing would be. Neither can the user be expected to configure the congestion control behavior on the host every time the operation environment changes.

Instead of performing per-destination or per-prefix congestion control, we introduce the concept of *congestion control pathlets*, and maintain some congestion control state for different portions of a network path. The state management still happens in the sending host, and we keep the end-to-end loop for congestion avoidance and related algorithms. But along with the normal protocol operation we gather more specific information about the congestion level at different segments of the communication path, and store this information for use by future connections. During the initial connection establishment, specific *pathlet routers* record pathlet information about the connection path together with some hints about the congestion state at each pathlet. This information is encoded in normal TCP/IP packets, and a host uses the collected information to determine an appropriate congestion window. Instead of maintaining congestion state on per destination, the hosts keep congestion state per pathlet.

We have borrowed the concept of pathlet from a recent paper describing a new routing protocol [9]. Although we use the common concepts with the routing paper, we don't strictly follow the terminology and techniques of the earlier paper, but instead focus on transport and congestion control with rather different mechanisms. Reader should therefore consider these two solutions independently from one another.

The pathlet approach to congestion control is particularly effective in settings where a host needs to simultaneously communicate with a number of destinations behind path with heterogeneous characteristics. In such setup any traditional method of picking the congestion window is not effective, as we will show later, while the pathlet approach quickly converges to the correct sending rate. We start by discussing

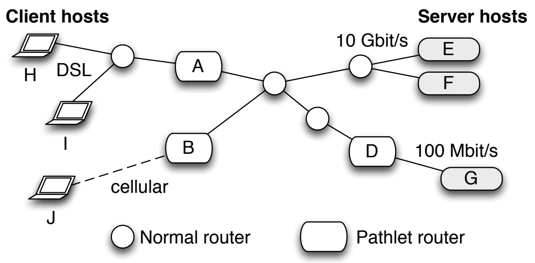

Fig. 1. An example instance of the network model.

our pathlet-based congestion control model in more detail, describe the protocol mechanism and its functionality as part of the normal TCP/IP operation, and finally report on our simulation-based evaluation.

II. Pathlet Transport Architecture

We will explain the Pathlet Transport Architecture (PTA) using an example illustrated in Figure 1. We do not deviate from the conventional end-to-end principles, and mostly rely on normal network elements: the control remains at the sending host based on end-to-end feedback from the receiver and the network path. As a new component, the architecture contains *Pathlet Routers* that can be used alongside the normal standard IP routers to record more specific information about the connection path, that can be used for more informed decisions at the sender.

A. Network operation

In the Pathlet Transport Architecture, the network path between sender and receiver is split into pathlets, as separated by *Pathlet Routers*. Pathlet routers are identified by (statistically) unique IDs. The identifier format is free: it could be based on IP addresses or random numbers. A pathlet is uniquely identified by the ID pair of the pathlet routers at its edges.

The Pathlet Transport Architecture can be incrementally deployed. In an initial deployment, there may be no pathlet routers on the connection path, in which case the path consists of just one pathlet. This is a valid setup, although the performance can be expected to be similar to a standard control block sharing mechanism that maintains a separate state to each host, as described by the Congestion Manager [4] or TCP Control Block Interdependence RFC [17]. In some cases even a single pathlet router aside conventional IP routers can significantly improve the performance, if it is located near bottleneck, for example near wireless link. Ultimately, one envisioned deployment could be to have the network edge routers operate as pathlet routers, while internal routers could be standard IP routers.

When a new flow is established (e.g., using a TCP SYN handshake), each pathlet router adds its identifier to the initial packet that passes them. When the packet reaches the receiver, it contains the sequence of pathlet routers that describes the network path taken to the receiver, which the receiver then echoes back to the sender. If the receiver does not support the Pathlet Transport Architecture, the sender falls back to normal end-host based connection state sharing. In the case of TCP/IP protocols the pathlet information can be recorded, for example,

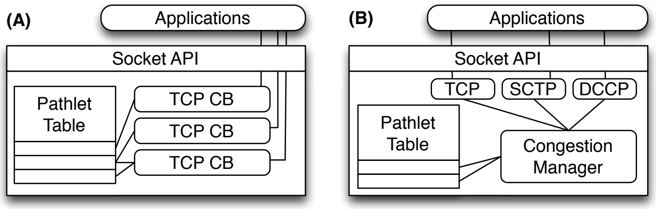

Fig. 2. Host architecture. (A): Incremental modification of current TCP implementation; (B): Revised congestion control architecture with common congestion manager module.

as IPv6 hop-by-hop extension header, or inside a TCP option that is processed by the pathlet routers (the latter obviously limits the number of pathlet routers).

The rest of the transport connection operation happens normally, except when there is congestion. Our model works best if routers support Explicit Congestion Notification (ECN) [15], although a variant of pathlet-based congestion control can also be implemented without ECN, if pathlet router is capable of tracking packet losses on a flow. When a pathlet router encounters a “Congestion Experienced” bit in an incoming IP header, it adds its identification to the packet, if no earlier pathlet router had done so. This information is echoed back to the sender, that can now detect not only that there has been congestion along the path, but also the pathlet on which congestion was first detected. Thereby, the sender can identify the bottleneck more accurately and use this information for the benefit of upcoming network flows that share the same pathlet.

B. End-host architecture

Figure 2 illustrates how a typical end host architecture is affected by the Pathlet Transport Architecture. The organization of different components is similar to an implementation with a host-based destination cache, although the details are different in the pathlet model. The diagram shows two alternative designs: (A) presents an incremental approach, where a pathlet table is added to a normal TCP/IP stack; (B) presents a more forward looking architecture similar to congestion manager, where congestion control operations are done by a common module that the transport protocol specific modules use in choosing the appropriate transport rate. A transport implementation continues to have per-socket control block instances for each active connection, that are used for active protocol operation and congestion management. The lifespan of the control block is usually equal to the lifespan of the application socket, and in many cases these communication sessions are so short that the normal congestion control operation does not have time to properly probe the network capacity.

In contrast to often short lived sockets, the pathlet table collects information of the different pathlets for longer term purposes. During connection establishment, a new socket first resolves the pathlet(s) used, and then uses the corresponding information from the pathlet table to initialize its protocol parameters appropriately. In this way, no magic numbers for parameters such as initial congestion window size or retransmission timeout are needed. The diagram also shows that input from multiple pathlets may be used when determining the socket state, because typically an end-to-end path used for connection consists of multiple pathlets.

After setting the initial parameters, the socket moves to normal operation. During the normal operation, however, the transport implementation collects data from the pathlets, and updates the appropriate pathlet entries in the pathlet table, to be used by future connections that share the pathlets.

The pathlet table contains at least the following parameters per pathlet:

- **Safe congestion window (SAFE_CWND)**: Congestion window that is determined to be safe to be used as an initial congestion window for the current pathlet. This can be based on recent observations of congestion signals from the pathlet, and we describe one algorithm for doing this shortly. This parameter should be picked conservatively to avoid excess traffic bursts to the network.
- **Safe slow-start threshold (SAFE_SSTHRESH)**: Slow-start threshold (ssthresh) that is appropriate for pathlet. Choosing an appropriate ssthresh helps avoiding slow-start overshoot that may result in several lost packets at the end of the initial slow start phase. This is particularly important with bigger initial congestion windows. This parameter is also based on the observations on congestion signals, but rather than being conservative like *SAFE_CWND*, SAFE_SSTHRESH should be set to be large enough, because too small ssthresh limits the connection startup performance.
- **Shortest and average round-trip time (MIN_RTT, AVG_RTT)**: The shortest round-trip time measured for connections with the particular pathlet involved. The initial and minimal RTO can be set based on these values, to improve the efficiency of timer-based recoveries.
- **Maximum transmit unit (MAX_MTU)**: The largest MTU that has been seen on the pathlet. Remembering this information can speed up the MTU discovery for new connections.

In the remainder of the paper we mostly focus on *SAFE_CWND*, because it has been a timely discussion topic in the research and standardization community, and appropriate choice of initial congestion window can be expected to result in most significant performance gains.

C. Some examples

We now briefly walk through the basic operation of the pathlet-based congestion control model, based on the setup illustrated in Figure 1. We pick a couple of communication scenarios from this setup under closer investigation, as illustrated in Figures 3 and 4. For the sake of the example, we’ll assume that the sending hosts have gathered data from earlier connections over the network.

With the first scenario, we point out that the pathlet routing architecture can be useful even in limited incremental deployments, with majority of the network equipment consisting of standard routers without PTA support. In Figure 3, a wireless provider has placed a pathlet router (B) in front of a slow wireless 2G link (as still common in many rural areas), but all other routers are standard network routers. The bandwidths

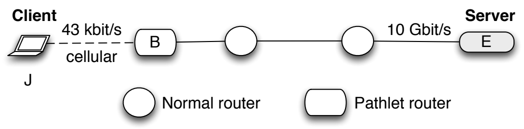

Fig. 3. An end-to-end path with a wireless link.

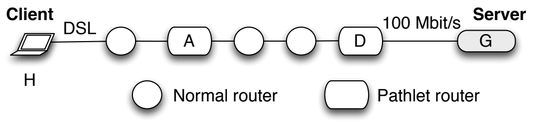

Fig. 4. An end-to-end path with a DSL link.

for such links are generally less than 100 kbps, and a small congestion window would be sufficient to fill up the link capacity. When the TCP SYN packet passes it, pathlet router B adds its identification to the pathlet header before passing it to the receiver. The receiver then echoes this information back to the sender. In practice the pathlet router labels are at least 32-bit statistically unique numbers.

Upon receiving the SYN-ACK segment the TCP sender reviews the state it has about pathlets EB and BJ. The state contains "safe congestion window" (SAFE_CWND), determined statistically from past connections over these pathlets and "minimum round-trip time" (MIN_RTT), among other possible parameters. The pathlet table reveals that pathlet BJ has the smallest value for SAFE_CWND and the largest value for MIN_RTT. Therefore, it uses the information from pathlet BJ to set the initial congestion window and retransmission timeout estimate. For a 2G link the congestion window is likely small, while the RTO estimate is long enough to avoid spurious timeouts. The congestion window estimate for pathlet BJ remains small, because pathlet router limits the transmission rate by marking its identifier to the Congested Pathlet field whenever the rate of incoming packets exceeds its capacity to process then. Note that an initial window of 10 segments would overshoot the cellular link capacity quite severely. If server E, on the other hand, would send data to receiver H behind a faster DSL link (see Fig. 1), it would notice a different pathlet router, and apply different congestion control state for the TCP connection.

Sometimes congestion occurs in the middle of network. Such situation can be detected when multiple pathlet routers are on the path. In the scenario shown in Figure 4 the sender first learns that the connection consists of pathlets GD, DA, and AH. If the network is under congestion on pathlet DA, the routers will add congestion notifications on IP headers, and the next pathlet router will mark the pathlet as congested pathlet. In this situation, the sender will apply the reduced congestion window to all subsequent connections traversing pathlet DA, unless the connection traverses a pathlet with even smaller capacity (e.g., the 2G wireless link in the previous example).

III. Protocol Operation

The Pathlet Transport Architecture operation consists of two distinct phases: the pathlet discovery to identify which pathlets are traversed by a transport connection being opened; and data collection, which happens during the normal communication and is used to adjust the pathlet-specific data. For the data collection phase, we focus on the congestion control parameters (congestion window and slow-start threshold), even though the Pathlet Transport Architecture could be used similarly to adjust different RTT-based timers such as retransmission timeout, or remembering the maximum transmission unit on the path.

A. Pathlet discovery

The PTA uses a new "Pathlet header" for collecting information about the network path during the initial discovery phase, and also later during the connection. For the rest of this paper, we assume that pathlet header is a new kind of IPv6 extension header[3]. In principle the same information could also be sent as a IPv4 header option, but those are known to involve serious deployment problems [14]. We believe that the new IPv6 equipment handles unknown extension headers properly, and will avoid the problems the legacy equipment has sometimes had with unknown IPv4 options. Another possible method could be to use TCP options, but that would only help TCP, and the 40-byte TCP option space is under contention by quite a few proposed TCP enhancements.

The Pathlet header is illustrated in Figure 5. The header can be divided into a common section, a pathlet discovery section, and a pathlet data section. The common section is a standard IPv6 extension header including the "Next Header" (NH) and Length fields. In addition, the header contains the number of slots reserved for pathlet identifiers during the pathlet discovery phase. If pathlet discovery is not ongoing, the value of "Slots" is zero, and the pathlet discovery section is omitted. "Rsvd" stands for 8-bits reserved for future extensions (and to align the pathlet identifiers on 32-bit boundaries). Each of the pathlet ID fields contains a 32-bit pathlet router identity, or zero, if the particular slot is empty. The sender is responsible for allocating the space for pathlet header[4]. It sets "Slots" to a large enough value such that majority of paths can be expected to be covered (see discussion of expected path lengths in Sec. V-B), and initializes the pathlet IDs with zeros. The "Congested Pathlet" field indicates the first pathlet router that detected congestion. The pathlet data section can be used for collecting also other pathlet data, and we leave the other potential uses for future work.

There are two versions of the Pathlet header that are identically formatted: the main Pathlet header is sent by the transport sender towards the receiver. When the receiver gets a Pathlet header, it copies the content of the Pathlet header into the first packet that goes back to the sender, as "Pathlet Echo" header with different header type. Intermediate pathlet routers do not process the "Pathlet Echo" header. In the case of bi-directional communication, a single packet can therefore have both the "Pathlet" header and the "Pathlet Echo" header.

As a packet with pathlet header passes a pathlet router, and has "Slots" value larger than zero, the pathlet router adds its identifier to the first free slot in the pathlet discovery section. If all slots are full before the packet reaches the receiver, the

[3]Yes, we believe that IPv6 *will* see eventual deployment, finally!

[4]Expanding headers in the middle of packet may be difficult in on-path processing.

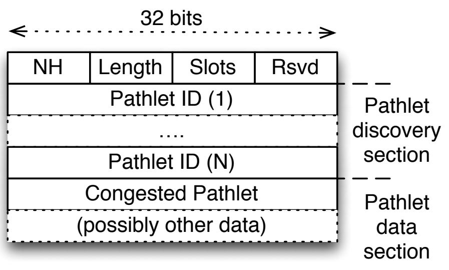

Fig. 5. Pathlet header

remaining pathlet routers simply pass the packet forward. In such case the remaining network path is treated as a black box, like in normal end-to-end transport session. While this may be a source of inaccuracy, it does not prevent the pathlet-based algorithms from working. Depending on the density of pathlet routers, the value of N might place a limit on the efficiency of this method. We envisage, in a future full deployment, to have pathlet routers at the edge routers between AS'es, and are not very concerned about this limit. The sender can also dynamically choose the number of slots based on the available space in packet, and based on past information about the expected maximum number of pathlet routers.

We also considered an alternative method for the pathlet discovery section, that would only contain one pathlet router identifier, and a hop count field, that is incremented by the sender for each subsequent packet until all pathlet routers have been discovered, similarly to the operation of the *traceroute* tool. This would save header space, and would not have upper limit to the number of pathlet routers on a path, but would require sending multiple packets for pathlet discovery (and a stable route during data collection). Given that one of our key motivations is to find appropriate initial values for congestion control parameters, this seems an unacceptable constraint.

Pathlet discovery is done in the beginning of a transport connection, but it can (and should) also be initiated during the connection either periodically, or based on some hint or trigger, to detect path changes. For example, if a mobile host knows that a hand-off has occurred, it must initiate pathlet discovery. When the other end notices from the Pathlet header that pathlet discovery has been initiated, it should take it as a strong hint to initiate the pathlet (re)discovery on its own for the opposite direction, because path changes usually affect both communication directions. Because the pathlet discovery header may take a significant amount of space, it can be done as a separate probe packet without transport payload, similar to zero-window probes in TCP.

B. Data collection

After it has been confirmed that the two ends of the connection support the Pathlet Transport Architecture, for example, through a successful pathlet discovery, the sender includes the pathlet header with pathlet data section in every outgoing packet. Because enhancing congestion control has been the main initial motivation in PTA, in the following we focus on collecting the congestion information along the path.

The intermediate routers between the pathlet routers can signal congestion by setting the ECN bits in the packet header. These routers can be normal IP routers without awareness of pathlets. When a pathlet router detects congestion on an incoming packet, it adds its identity to the Congested Pathlet field in the pathlet header, if this field has not been used by an earlier pathlet router. In other words, the Congested Pathlet field indicates the pathlet router at the end of the first pathlet where congestion was detected. If pathlet router itself is congested, i.e., it cannot push packets on the outgoing link at high enough rate, the congestion is considered to occur on the uplink pathlet, and the pathlet router can signal it by simply placing an ECN Congestion Established mark on the packet. There is also a special case when congestion occurs on the pathlet between the last pathlet router and the receiver. In this case the receiver marks a special *End-of-Path* code to the Congested Pathlet field, if it detects an ECN mark. As the pathlet header reaches the receiver, and subsequently is echoed back to the sender, the sender learns which pathlet was congested. Based on this information, the sender can do adjustments in its pathlet table, using the algorithms described below, in addition to following the normal rate reduction algorithms, such as reducing the TCP's congestion window.

The above-described protocol is capable of indicating a single congested pathlet per packet. This is sufficient, because in an end-to-end congestion control mechanism the transmission rate converges to the slowest bottleneck on the path. In other words, the sender gets most up-to-date information from the pathlet that is the bottleneck on its path. For pathlets that do not generate congestion signals the sender may not have recent capacity information. It is possible that bottleneck shifts over time to another pathlet because of changing traffic patterns, but the sender can notice this from changed Congested Pathlet field.

As stated in Section II, it is possible that there are no pathlet routers on a connection path. In this case the path is modeled as a single pathlet by the sender. The protocol mechanisms and related algorithms work correctly also in that case, although without performance advantages compared to traditional control block sharing methods.

C. The pathlet update algorithm

The Pathlet Transport Architecture decouples the protocol mechanism from the algorithms for using the pathlet information applied at the sender. We envisage that various advanced algorithms, or even aspects of machine learning, could be developed as later research based on the pathlet information. Below we outline a simple algorithm for setting the *SAFE_CWND* as an initial proposal. As will be shown in Section IV, even this simple algorithm can achieve noticeable performance improvements.

As illustrated in Figure 2, the sending operating system maintains a Pathlet Table, that contains some state for each pathlet known (recall that pathlets are identified by a pair of pathlet router identifiers), as described in Section II.

The basic algorithm follows TCP's congestion control principles and operates as follows.

1) For every pathlet P that is known to the host S, the host maintains a state for $\mathit{SAFE\_CWND}_P$ estimation. The

sender also maintains a count of congestion indications for each pathlet.

2) The source updates the *SAFE_CWND*$_P$ value once per time period *T*. Setting of *T* should be proportional to the typical RTT's over the particular pathlet. For our experimentations, we simply take the average RTT for the flows using pathlet *P*, and use that for time period *T*. The sender also remembers the congestion control values for the past *N* periods. The values for different periods are indicated by *SAFE_CWND*^i_P, for period *i*. The current period is period number 0. For each update the host uses its information of per flow flight size that is identified with *FlightSize*$_F$.
3) When a Pathlet Echo header contains a "Congested Pathlet" indication, the sender increments the congestion indication counter for the current period for that particular pathlet.
4) At the end of each period the related values are then updated as follows:
   a) If there were no congestion signs during the last *N* periods, the *SAFE_CWND*$_P$ is updated. The updated value for *SAFE_CWND*$_P$ is *MAX*$_{F,i}$*(FlightSize*$_F$*, SAFE_CWND*^i_P*)* for all $i \in [0..N]$ and for all $F \in [$ *flows using P in recent N periods*$]$.
   b) If a congestion sign is observed, *SAFE_CWND*$_P$ is set to *MIN(SAFE_CWND*$_P$*, MIN*$_F$*((FlightSize*$_F$*)/2))* for all $F \in [$*recent congested flows using P*$]$.

D. Selecting initial congestion window

Instead of using fixed setting for initial congestion window, a new TCP connection can use the information in Pathlet table for choosing an appropriate safe setting for the congestion window. In this case the initial congestion window is chosen according to the following steps.

1) Set ICW (initial congestion window) to MAX_INT (largest possible integer value).
2) For each pathlet *P* that this flow passes based on the pathlet discovery algorithm, check the $SAFE\_CWND_P$ state in the pathlet table.
   a) Set the ICW to $MIN(ICW, SAFE\_CWND_P)$.
3) If ICW is still equal to MAX_INT at the end of this process, it means that the pathlet table did not have data about the needed pathlets. In this case the congestion window could either be initialized to a fixed values in a traditional way, or some external information could be used to set it, as discussed in Section V.

The initial congestion window is shared between all flows started over the bottleneck pathlet within a round-trip time. The sender can distribute the initial window evenly, or prioritize between flows.

It may happen that a particular host does not send any data over an earlier used pathlet for a longer time period. The problem after such longer idle period is, whether the pathlet capacity estimation is valid anymore. A simple conservative approach would be to follow a method similar to TCP's congestion window validation [10], where the window estimate slowly decays over time. The downside is that doing so may weaken the ability to start at a (still) ideal rate from the beginning of the connection. Therefore, some advanced heuristics might be in place to predict the likely capacity on longer term intervals. We leave such heuristics for future work.

IV. Evaluation

We have evaluated the Pathlet Transport Architecture and a few different settings of fixed initial congestion window (ICW) with a set of ns-3 simulations. As we have discussed earlier, the protocol mechanism in PTA is independent from the ICW setting or any other algorithm used in the sender. However, for demonstrating the PTA performance and benefits of its extensive information sharing we focus on initial window. The other parameters benefit TCP in different ways; for example, the MAX_MTU parameter helps path MTU discovery, which is not easily demonstrated by simulations.

The simulations use the topology shown in Figure 6, in which the bottleneck links are the last links next to the router that connects to the receiver groups. Each groups of receiver are connected to the source through slightly different bottleneck links, with bandwidths of 100 Mbps, 4 Mbps, and 43 Kbps. Each of these bottleneck links are shared among the receivers from same group. The rest of the links have bandwidth of 1 Gbps. The propagation delays and queue sizes for these links are as shown in the diagram. The Maximum Transmit Unit for each of the links is 1500 bytes. In each simulation scenario, the flows are between a few sources and many receivers that share the same bottleneck links.

We mainly compare initial windows of 3 and 10 packets to the initial window determined by the PTA[5]. Initial window of 3 packets is the common size based on the IETF standard [3], assuming packet size of 1500 bytes. Initial window of 10 packets, on the other hand, is the value proposed recently to be used by TCP implementations [7] (and as already implemented in Linux).

The boxes denote pathlet routers, i.e., the connections between the senders and receivers consist of 4 pathlets. The routers support ECN, and we apply Random Early Detection mechanism (RED) [8] to place the ECN Congestion Established bits in packets. The minimum marking threshold in RED is 10 % of the outgoing link queue size, and the maximum threshold is 30 % of the queue size.

We use two different traffic profiles in our simulations, simply calling them traffic profile 1 and traffic profile 2. Traffic profile 1 follows a Pareto model with a long tail for short flows (Pareto mean 40 KB, shape 1.5), as motivated by other studies on increasing initial congestion window size [1], [7]. The size distribution is shown in Fig. 7. Traffic profile 2 has longer flows (Pareto mean 200 KB, shape 1.5), as shown in Fig. 8. The latter profile emphasizes the effect of larger initial congestion window, because for short flows the initial congestion window does not always have significant effect. Flows arrive in the network based on Poisson distribution. The

[5]The RFC specifies initial window in bytes, we speak of packets for convenience.

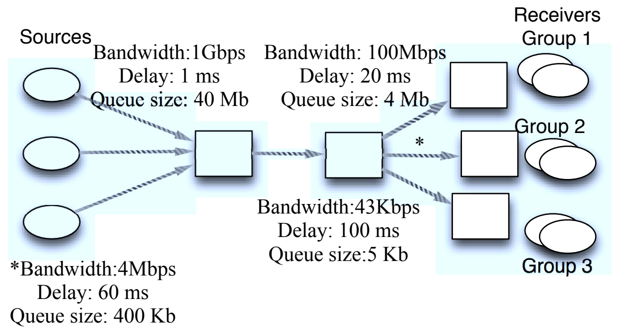

Fig. 6. The simulation topology

inter-arrival times between different flows are exponentially distributed. We vary the mean interarrival parameter to test different traffic intensities, and in most of the following graphs the mean interarrival time is shown on the x-axis of the graphs.

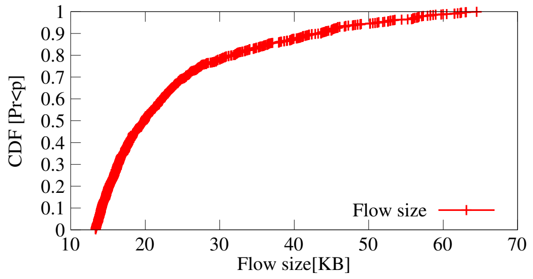

Fig. 7. Traffic profile 1 with mean flow size of 40KB

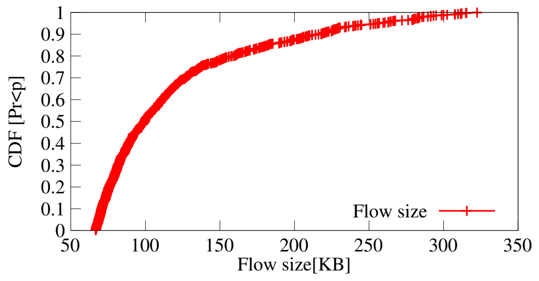

Fig. 8. Traffic profile 2 with mean flow size of 200 KB

A. *Flow completion time*

Since the main motivation for increasing the initial congestion window (ICW) has been to improve the flow completion times (FCT), we start our evaluations by investigating the FCT. For this scenario, we chose to test the case when the receivers of traffic profile 1 are behind the 4 Mbps link and they experience average RTT of 135 ms. In this case if the flow-interarrival time is bigger than 300 ms we consider the network to be not congested at all, while when the flow-interarrival time is reduced to 20 ms some signs of congestion are observed at the bottleneck link. As can be seen in Fig. 9, when there is not much congestion in the network, initial window of 3 packets does not utilize the network at full capacity and therefore uses more round-trip times to complete the transfer, and therefore in worse flow completion time performance. The benefit of larger initial congestion window can be seen from the graph.

When the congestion in the network increases, larger initial window is not necessarily better anymore. Figure 10 shows a

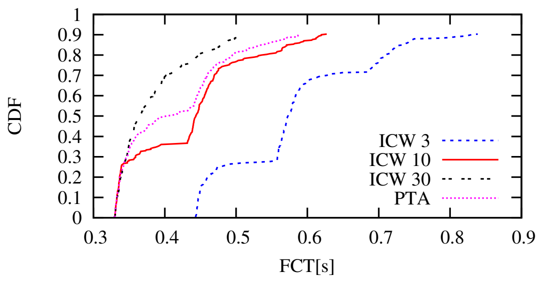

Fig. 9. CDF of flow completion times (FCT) without experiencing congestion

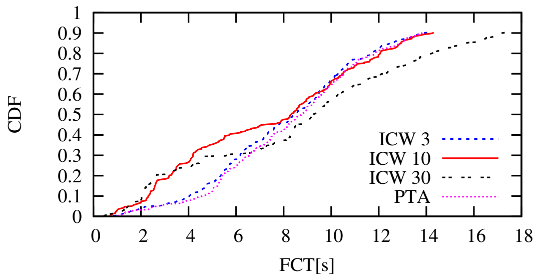

Fig. 10. CDF of flow completion times (FCT) with congested bottleneck

scenario with similar parameters as 9, but with increased load on the bottleneck link. As a result of increased congestion and retransmissions, there is no significant advantage in choosing a higher initial congestion window – in fact it can be harmful. However, because TCP can adapt to the correct rate with its own congestion control mechanisms, for long flows the relative difference is less noticeable, even if the higher initial window causes a sudden peak of congestion in the beginning of connection.

After showing the simple effect of increasing the initial window sizes we now focus on the effect of using the Pathlet Transport Architecture. One of the main motivations for the PTA-based method is its adaptability to different network conditions, because of its extensive information over different time periods and different flows. Figures 9 and 10 show that PTA-based method results in good flow completion times in both uncongested and congested cases. When there is congestion, PTA-based method behaves similarly to fixed initial congestion window setting of 3 packets, while in non-congested cases its performance is similar to the higher congestion window values. The two graphs show that PTA-based method is indeed able to adapt to different traffic loads dynamically, without a fixed initial congestion window setting.

We next take a look at a scenario where a server has clients from multiple different networks with wide range of different bottlenecks, as illustrated in Fig. 6. We believe this can be a realistic setting even today: while many customers are gaining bandwidth in their office and home network connections, there continues to be slow wireless cellular connections that are used to access the same data. The 43 Kbps bottleneck in our simulation setting is roughly similar to the 2G cellular networks that are still in use in developing countries, or in rural areas without 3G or 4G coverage. We execute the simulations with various flow interarrival times (as shown in x-axis) to investigate the effects of different levels of congestion. The shorter the flow-interarrival gets, the more traffic and

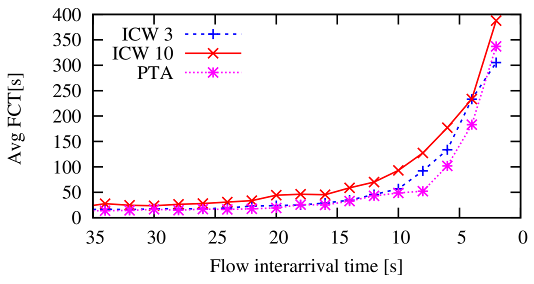

Fig. 11. Average FCT with 100 TCP flows between servers and clients with 43 Kbps bottleneck

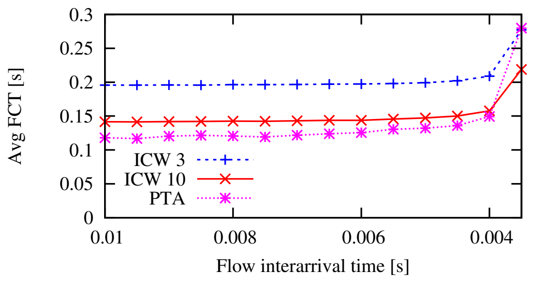

Fig. 12. Average FCT with 1000 TCP flows between servers and clients with 100 Mbps bottleneck with low level of congestion

congestion is created on each link.

Figure 11 shows the flow completion times over the very slow bottleneck, and Figure 12 shows the completion times over a high-speed bottleneck. The main observation is that PTA is able to find the right initial window size in both cases, while no fixed initial window value works perfectly for both cases. With a slow bottleneck like ours, even the standard initial window size of three packets may result in congestion. In these cases PTA-based method ended up starting from initial window of 1 or 2, based on the earlier pathlet feedback. For fast bottleneck initial window of 3 is clearly ineffective, and PTA-based method running on the same server can adapt to that adequately. Note that due to the high bandwidth in Figure 12, most of the graph covers a non-congested region and the effects of congestion start to be visible only in the right side of the graph.

Figure 13 shows the 100 Mbps bottleneck link under a more severe congestion. There is no benefit of larger initial window, and in fact, the large initial window can be harmful and increase the unpredictability, as can be seen on the right end of the graph with highest traffic intensity. PTA adapts well also in this scenario. The graph shows that ICW 10 performs irregularly: on light load it slightly outperforms PTA, but is harmful for performance when load increases. The slightly improved performance of ICW 10 in this case comes at the expense of heavy buffer load, as will be explained shortly in Section IV-C, and may therefore be harmful for other users of network.

As noted earlier, the fact that many of the flows are shorter than the initial window, especially in the case of initial window of 10 packets, mitigates the effects of the initial window selection, because many of the flows do not have that many packets to send. Figure 14 illustrates this effect on the 100 Mbps bottleneck case. As the flow size increases, the relative benefit becomes bigger.

B. Goodput

The flow completion time of a flow depends, of course, on the size distribution of flows. Therefore we next illustrate some results in terms of ICW size and goodput, i.e., file size divided by the flow completion time. This better shows the performance differences in bytes per second, which was not so easily perceived from the graphs above.

As a starting point we will show the average ICW size that PTA model achieves under various conditions. Figure 15 illustrates the average window size that our PTA-based method

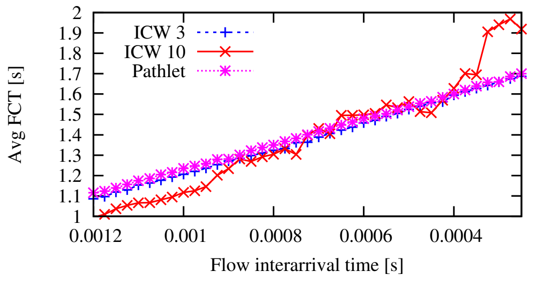

Fig. 13. Average FCT with 1000 TCP flows between servers and clients with 100 Mbps bottleneck with high level of congestion

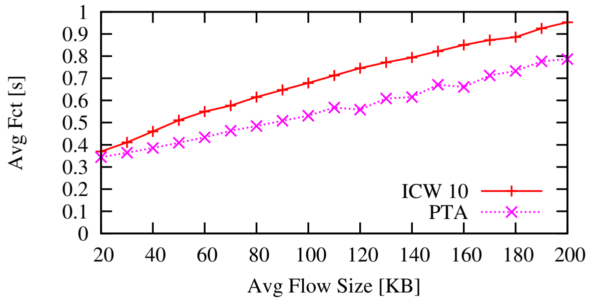

Fig. 14. The changes in FCT with increasing average flow size

achieves with different traffic profiles. As can be seen in the figure, the PTA-based method adjusts the ICW size based on the congestion on the link. Traffic profile 2 with longer flows naturally causes more load on the network than traffic profile 1 with shorter flows. The graph shows how this difference is reflected in the initial window selection made by the PTA.

Figures 16 and 17 show that with low levels of congestion, different initial window sizes results in different performance, but as the congestion increases the performance differences vanish. The graphs, again, show that initial window of 10 packets is better for high-speed links, while initial window

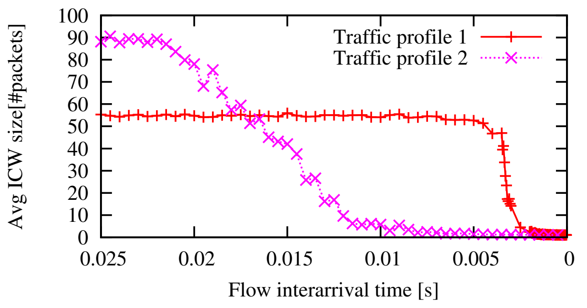

Fig. 15. Average ICW size with 1000 TCP flows between servers and clients with 100 Mbps bottleneck

of 3 packets works for low-speed links. PTA is better in both cases, because on high-speed links it can pick even higher values than 10, and in the low-speed case it stays at initial window of 1 or 2 packets.

If the reader wonders about the difference of goodput to the nominal link bandwidth, it is good to note that goodput covers the whole duration of TCP connection, including initial SYN handshake. Especially with short flows and longer relative round-trip time, this results in lower calculated goodput, because during initial handshake no data is transmitted.

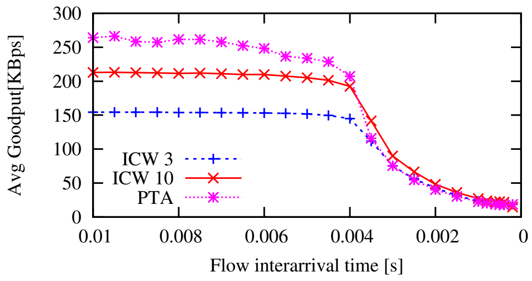

Fig. 16. Average goodput with 1000 TCP flows between servers and clients with 100 Mbps bottleneck

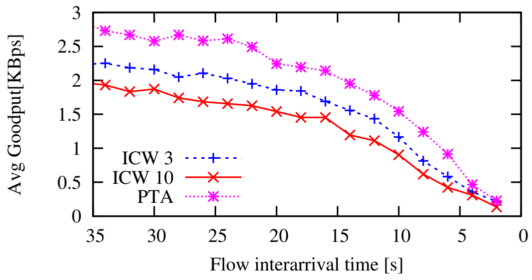

Fig. 17. Average goodput with 100 TCP flows between servers and clients with 43 Kbps bottleneck

The above results were based on traffic profile 1, that contains many short files. Figures 18 and 19 show a similar scenario, but with traffic profile 2 with higher mean file size. We can see the significant difference on the goodput for the 100 Mbps bottleneck, and how PTA is significantly better. For the 43 Kbps link the file size does not have noticeable effect.

C. Drop rate and queue length

In order to better understand the reasons in the above presented performance differences, we now investigate the packet loss rate (due to congestion), and the development of the queue length in the bottleneck routers.

Figure 20 shows the devastating effect of overly large initial congestion window in the slow wireless link, when it comes to number packet losses. These results are well in line with the earlier goodput graph, and show the correlation between the drop rate and goodput. PTA does not reduce the average queue length (Fig. 21), but fewer packet losses mean that the queue length is less variable and queuing delays more constant with PTA.

Figures 22 and 23 show the similar metrics for 100 Mbps bottleneck link. Congestion effects are shown only at the right side of the graphs. The initial window of 10 packets causes a significant number of packet losses and highly variable queue

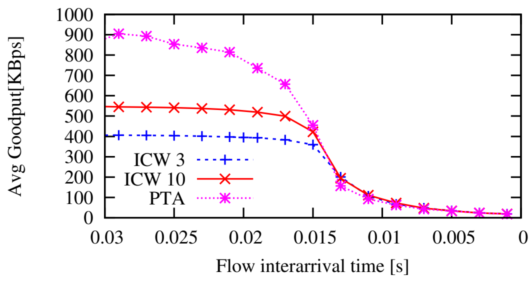

Fig. 18. Average goodput with Multiple TCP flows of traffic profile 2 between servers and clients with 100 Mbps bottleneck

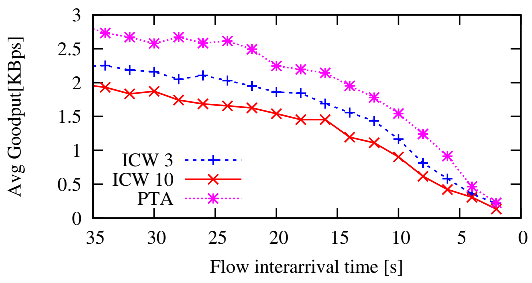

Fig. 19. Average goodput with Multiple TCP flows of traffic profile 2 between servers and clients with 43 Kbps bottleneck

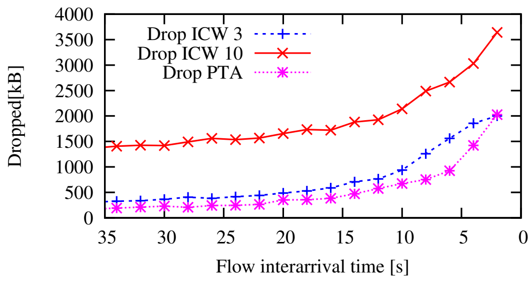

Fig. 20. Average drops with 100 TCP flows between servers and clients with 43 Kbps bottleneck

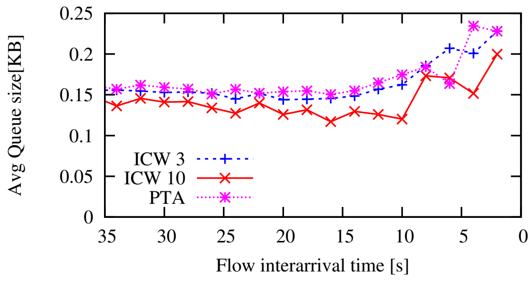

Fig. 21. Average bottleneck queue size with 100 TCP flows between servers and clients with 43 Kbps bottleneck

size when the bottleneck becomes congested. These results, too, are in clear correlation with the earlier goodput plots.

D. Fairness towards other flows

Finally, we investigate how increasing the congestion window affects on other flows that are using the standard initial congestion window of 3 packets. It can be expected that under congestion the more aggressive flows consume bandwidth from the conservative flows, and Figure 24 quantifies what this means on a bottleneck link of 4 Mbps (as shown in the topology in Fig. 6). The figure plots the average performance of 200 KB flows that apply initial congestion window of 3 packets, while there are a second set of traffic profile 1 flows,

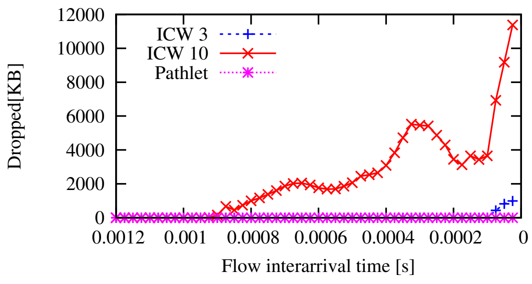

Fig. 22. Average drops with 1000 TCP flows between servers and clients with 100 Mbps bottleneck

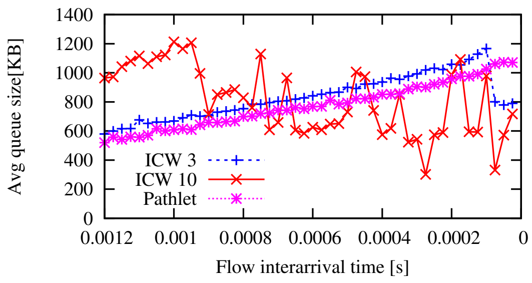

Fig. 23. Average bottleneck queue size with 1000 TCP flows between servers and clients with 100 Mbps bottleneck

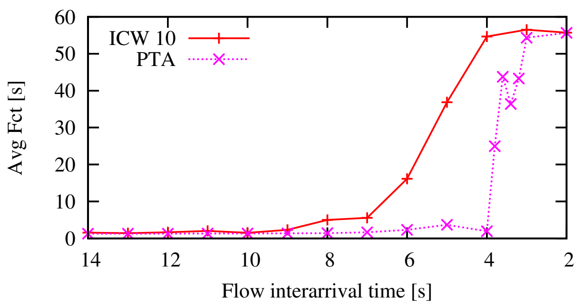

Fig. 24. The effect of improving ICW size for some flows, on the average FCT of existing long flows

with either initial window of 10 packets, or applying PTA. The flow interarrival time of the second set of flows varies as shown on x-axis.

Because of its adaptive nature, PTA is not as harmful to the long conservative flows as fixed initial window setting of 10 packets. At the right end of the draft the performance of the long flows unavoidably collapses because of the extreme congestion, but in that case there is little that any choice of initial congestion window can help.

V. Discussion

A number of aspects of our architecture proposal warrant further discussions.

A. Path changes and multipath operation

Internet paths may be asymmetric. PTA deals with this by performing independent pathlet discovery in each direction.

Moreover, network paths may change. In some cases, such as node mobility, one of the hosts may become aware of such a change by local means and initiate a pathlet rediscovery. As noted above, the receiver of a rediscovery should initiate one in the opposite direction as well. If a discovery yielded a full pathlet router section a host should consider initiating a rediscovery with a larger number of pathlet ID slots in the Pathlet header. Hosts should also watch out for other cues of path changes to initiate a rediscovery: They should store the complete pathlet router information from the last discovery and take congestion indications from unknown pathlet routers as an indication for a route change and they also should treat drastic changes in path characteristics as such a hint.

The PTA may be used to support *multipath TCP (MPTCP)* [18], which allows splitting a logical TCP connection across multiple sub-connections utilizing potentially disjoint paths. MPTCP uses a modified TCP SYN/ACK handshake to open each sub-connection, which in turn could benefit from pathlet (re)discovery to determine up front which paths share joint bottlenecks. This could serve as input to the path selection and packet scheduling in MPTCP.

Finally, in spite of all pathlet state sharing, the pathlet table of a host will be empty at boot time (and so *some* magic numbers would still be needed. However, pathlet data could be provisioned (e.g., from an DSL access pathlet router via DHCP) for local environments and might even be shared among users for the wire area: for example, search engines might gather pathlet data while crawling and supplement responses with pathlet data potentially relevant to the querier. Alternatively, mediated crowd-sourcing systems for sharing could be envisioned, but these are mainly optimizations for the first few connections.

B. Header overhead

Analysis over an AS-level topology shows that the mean AS-level path length is 3.77 [9]. If we assume one pathlet router per AS and maximum of 6 AS'es, this would result PTA causing an additional header overhead of 32 bytes by average for packets that include pathlet discovery section, that is usually present only in the beginning of a connection, and 8 bytes for packets that only contain the pathlet data section. Of course, it could be possible that some domains do not deploy pathlet routers at all, or that other domains have more than one pathlet router in its AS'es. We consider this header overhead reasonable, for example comparing to the various TCP extensions currently in use.

C. Interoperability with non-ECN routers

From the pathlet router's perspective, ECN support at intermediate routers allows an easy stateless operation. If ECN is not supported, i.e., packets are dropped to signal congestion, the pathlet router would need to track flow state in order to detect three consecutive duplicate TCP ACKs corresponding a data flow in order to declare congestion. While this is possible in principle, it will significantly add to the complexity of a pathlet router design. Therefore, in the case of a confirmed lost packet without the relevant pathlet information, a sender needs to assume that congestion may have occurred on any of the used pathlets, and do its adjustments accordingly.

D. Scalability

When ECN is supported on the path, pathlet routers do not need to store per-flow state. The main processing expense is to add router's own identification to the packets with pathlet discovery section, i.e., usually TCP SYNs. Therefore we do not believe that a DoS attempt injecting large amounts of pathlet discovery packets would be a significant concern for the system. A pathlet router could also skip processing of pathlet discovery packets and just forward the packet, if it becomes under extreme load.

VI. Related Work

There has been some previous work to dynamically select the initial congestion window, such as Quick-Start [16]. However, Quick-Start requires use of IP options, and participation by every router along the path to be effective. This combination makes it very hard to deploy. On the other hand, the Pathlet Transport Architecture works together with standard (preferably ECN-capable) routers, and can be incrementally deployed by eventually adding the pathlet routers on critical places and doing the necessary end host modifications.

There are also some more fundamental proposals for a new congestion control architecture, such as XCP [12] and RCP [6]. However, like Quick-Start, these require changes to protocol header structures, and consecutively, collaboration from all intermediate routers, therefore being difficult to deploy. The PTA-based congestion control mechanism keeps history over time and over different flows that share the same pathlet while each pathlet can contain many routers that do not know anything about the PTA model. Thus, in the PTA-based method the inital rate estimation can be done without direct inquiry from every single router on the path.

The earlier aggregated congestion control approaches that we know of view the network as a black box that is estimated by a single set of congestion control parameters. The novelty in PTA is that it explicitly aims to pinpoint the bottleneck portion of the path and use that for the benefit of the congestion control heuristics. We are not aware of congestion control schemes, that would at the same time identify these bottlenecks, but still adhere to the normal end-to-end principles. Although, with protocols such as good old TCP this information might not be that useful, but we believe that knowing the congestion location can be helpful in the design of better rate control mechanisms. Investing the design of new rate control mechanisms remains as part of our future work.

VII. Conclusions

We have introduced the Pathlet Transport Architecture that splits the connection path into segments, called pathlets, as identified by a new kind of pathlet routers. Pathlets allow more specific information about the connection path, for example to implement more accurate congestion control based on aggregated pathlet information. PTA still follows the standard congestion control principles, and the normal transport protocol practices. The PTA architecture can be incrementally deployed, and it works together with standard Internet routers.

Along with our PTA evaluation, we also showed that the choice of a correct initial congestion window is not an easy one. While large initial window improves performance in many cases, there are always regions where more conservative selections would have been more appropriate. PTA makes this choice on behalf of the system designer, by leveraging the collected network information.

The Pathlet Transport Architecture could be the first step towards a world of more adaptive protocol stacks, that use the information from previous interactions to make better decisions for current connections. The pathlet information could be exchanged among multiple hosts, or provisioned by network operators, to assists hosts in making better informed decisions about protocol parameters. We will investigate these ideas as future work.

Acknowledgements

This research was partly supported by the EU's PURSUIT project (FP7-INFSO-ICT-257217).

References

[1] Mohammad Al-Fares, Khaled Elmeleegy, Benjamin Reed, and Igor Gashinsky. Overclocking the Yahoo! CDN for Faster Web Page Loads. In *Proc. IMC '11*, 2011.

[2] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). In *Proceedings of ACM SIGCOMM '10*, pages 63–74, New York, NY, USA, August 2010. ACM.

[3] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. RFC 3390, October 2002.

[4] Hari Balakrishnan, Hariharan S. Rahul, and Srinivasan Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proc. of ACM SIGCOMM '99*, Cambridge, MA, USA, September 1999.

[5] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. Tcp vegas: new techniques for congestion detection and avoidance. In *Proc. of ACM SIGCOMM '94*, pages 24–35, London, UK, October 1994.

[6] Nandita Dukkipati. *Rate Control Protocol (RCP): Congestion Control to Make Flows Complete Quickly*. PhD thesis, Stanford, CA, USA, 2008.

[7] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. An argument for increasing TCP's initial congestion window. *SIGCOMM Comput. Commun. Rev.*, 40:26–33, June 2010.

[8] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, August 1993.

[9] P. Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet routing. In *Proc. of ACM SIGCOMM*, 2009.

[10] M. Handley, J. Padhye, and S. Floyd. TCP Congestion Window Validation. RFC 2861, June 2000. Experimental.

[11] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM '88*, pages 314–329, Stanford, CA, USA, August 1988.

[12] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM*, 2002.

[13] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. TCP Westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proc. of ACM MobiCom '01*, MobiCom '01, pages 287–297, Rome, Italy, July 2001. ACM.

[14] Alberto Medina, Mark Allman, and Sally Floyd. Measuring the Evolution of Transport Protocols in the Internet. *ACM SIGCOMM Computer Communication Review*, 35(2):37–51, April 2005.

[15] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, 2001.

[16] Pasi Sarolahti, Mark Allman, and Sally Floyd. Determining an appropriate sending rate over an underutilized network path. *Computer Networks*, 51(7):1815–1832, 2007.

[17] J. Touch. TCP Control Block Interdependence. RFC 2140, April 1997. Informational.

[18] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for Multipath TCP. In *Proc. of USENIX NSDI '11*, 2011.

[19] Lisong Xu, K. Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *Proc. of IEEE INFOCOM 2004*, volume 4, pages 2514 – 2524, Hong Kong, March 2004.