Transparent Flow Migration through Splicing for Multi-homed Vehicular Internet Gateways

Joshua Hare, Lance Hartung, and Suman Banerjee
Dept. of Computer Sciences, University of Wisconsin-Madison
Madison, WI, USA
{hare, hartung, suman}@cs.wisc.edu

***Abstract*—We propose a technique, called FloMiS, by which a flow can be migrated from one network to another without requiring any changes to the endpoints (Internet-based servers or mobile clients). Our work is motivated by the needs of our vehicular Internet service, called WiRover, through which we provide WiFi connectivity in public transit vehicles — city buses as well as long distance buses operating in and around Madison, WI — using multi-network gateways. In our vehicular environment, we have observed that each cellular path experiences stalls and failures resulting in outages to network traffic. To provide robust and uninterrupted experience for these flows, we implement the FloMiS mechanism in the gateways alone, through which the gateways re-initiate a request for the contents of the interrupted flow and splice them back to the original flow in a manner that is transparent to the mobile clients.**

FloMiS is designed to optimize the majority of network traffic, which our analysis (of about 6 million real user flows) indicates that FloMiS is capable migrating a flow in as little as two round trip times for more than 93% of the traffic volume. Clients left to their own mechanisms would typically recover after several minutes of disconnectivity.

I. Introduction

The vehicular environment is not immune to the demands of our mobile-centric Internet. In fact the vehicular environment is often challenged as wireless Internet connectivity typically suffers from either limited bandwidth or sporadic coverage. A number of prior and ongoing efforts have attempted to meet such demands by designing vehicular gateways to connect through multiple wireless (typically, cellular) networks *simultaneously.* Examples include MAR [16], Pluribus [13], and WiRover [12]. The multi-network structure provides vehicular gateways with high aggregate bandwidth and robustness that is not possible in a single network solution.

For more than 3 years we have operated our own deployment of multi-homed vehicular Internet gateways (also referred to as *gateways*). Our deployment encompasses a total of 5 passenger buses: 2 city transit buses that serve the city of Madison, WI, USA and 3 coach buses that serve the Midwestern portion of the USA. We have served tens of thousands of passengers over this period of time.

In our experience, we have found that aggregated bandwidth can be achieved for a client through simple per flow load balancing [12]. However, providing an Internet service that is robust to network failures is a more challenging problem. Prior designs [16], [13], [12] have all relied on a "multi-network tunnel to proxy" abstraction to provide a robust service. Network tunnels that are established between vehicular gateways and a specialized migration proxy allow a flow to be transparently migrated from a failed network link.

All authors are partially supported by the following grants from the US National Science Foundation: CNS-1040648, CNS-0916955, CNS-0855201, CNS-0747177, CNS-1064944, CNS-1059306, CNS-1343363, and CNS-1230751.

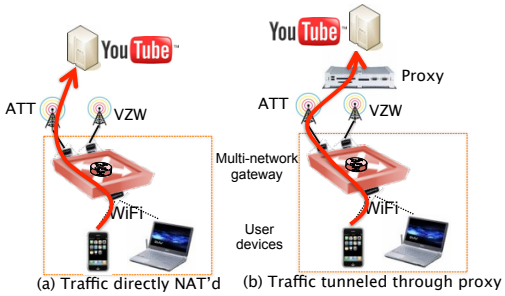

Fig. 1. **Two alternatives for flows in the vehicular gateway systems between Internet-based servers and user devices in the vehicles.**

When flow migration is coupled with link diversity significant improvements to robust connectivity can be realized [16]. In fact, our deployed Internet gateways are equipped with 2 diverse cellular interfaces. Previously, we had found that each individual network provided connectivity for 78% and 80% of the time the vehicles were operational. However, we found that at least one of the two cellular interfaces had connectivity for more than 94% of the time [12].

A. Multi-network Vehicular Gateways and Flow Migration

Typically, multi-network gateways provide passengers with a local WiFi hotspot to which they connect. Once connected the network traffic generated by passengers is forwarded over one of the available cellular networks.

Figure 1 depicts a multi-network gateway equipped with an AT&T interface as well as a Verizon interface. In such a scenario there are multiple ways to transport passenger's network flows between the passenger's device and Internet-based servers:
(a) *Directly NAT'd:* To route a flow the gateway must first select a cellular network and then apply NAT operations to each packet of the flow.
(b) *Tunneled through a specialized proxy:* Alternatively, the gateway can tunnel the flow to a proxy which would then NAT and forward the traffic to the YouTube server.

In both solutions a single user can benefit from the bandwidth available for both cellular networks. However, the

978-1-4799-2687-9/13/$31.00 ©2013 IEEE

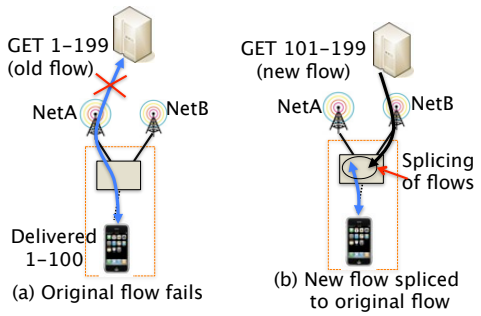

Fig. 2. **FloMiS works through splicing a new flow into an old flow requesting the same content.**

advantage of the *tunnel and proxy* solution is the additional robustness that is possible. The robustness is due to the fact that the YouTube server actually binds to the proxy's IP address. The proxy is thus providing a static IP address for Internet-based servers to bind to while the gateway's interfaces may be experiencing link failures and address changes due to network mobility. The *direct NAT* solution cannot provide a similar level of robustness since the Internet-based servers are actually binding to the cellular interface's IP address which may fail or change due to mobility. Such a failure would cause the flow to completely stall and require the user to intervene and re-start the flow.

When a link fails with the *tunnel and proxy* solution the gateway and proxy can transparently forward traffic over alternative network paths. For example, if the AT&T path were to experience a cellular outage (or even a period of poor network performance), the proxy and gateway can migrate the flow to the Verizon network without breaking the flow's end-to-end connection.

Despite all the advantages, the *tunnel and proxy* solution have a number of disadvantages. To tunnel packets from the gateway to the proxy each packet incurs an overhead of 28 bytes to accommodate the IP-in-UDP encapsulation. UDP encapsulation is used since many cellular service providers block IP-in-IP tunneling [13], [12]. Routing traffic through a proxy can further introduce routing inefficiencies (i.e., triangle routing and single point of failure).

In this work we take a new approach to the challenges of providing robust connectivity call FloMiS. First, we observe that robustness is primarily compromised when non-transient link failures occur and design a solution that mainly incurs overheads only when failures occur. We classify non-transient link failures as failures that: (a) are detectable by the gateway system and (b) would potentially impact a network flow and thus the user's experience. Secondly, since the vast majority of traffic generated by the users of our system is entertainment based (i.e., web surfing, audio/video streaming [12]), we have designed and built a solution that exploits specific characteristics of this type of traffic. Specifically, we exploit that fact that this traffic is typically: downlink, static, and unencrypted. In fact more than 93% of total traffic volume have these characteristics. We further exploit the fact that short interruptions to user's downloads may not directly impact the user's experience as we will explore further in Section IV.

Two key properties of FloMiS are that: (a) much of FloMiS's overheads are only required in the event of a link failure and (b) FloMiS is only applied in the event of non-transient link failures that could potentially have negative impacts on the user's experience. The second property is due to the fact that if the original link and thus flow recover before FloMiS could otherwise recover the download via splicing operations, FloMiS will abort the flow's migration. In all, FloMiS achieves a similar robust user experience as the *tunnel and proxy* solution for the majority of traffic volume without the overheads and inefficiencies of the *tunnel and proxy* solution.

Network	NetA	NetB
Total Failures	2480	773
Avg Failure Duration	57.2 sec	40.1 sec
Avg Time b/w Failures	118 min	158 min

TABLE I. **Non-transient link failure statistics as calculated from our deployment.**

B. Flow Migration with FloMiS

To explain how our FloMiS technique improves robustness we will use a simple example. Consider that a bus passenger (the client) is streaming a video flow (typically, over an HTTP connection). The gateway can monitor and record the client's initial download request for the video. Let us assume that the download originates through NetA and at sometime in the future, this download experiences an outage.

To migrate the download the gateway recalls the client's initial download request for the video. Coupled with knowledge of the last byte downloaded on the failed video stream the gateway pro-actively issues a *brand new* request to the same video server for the rest of the video. The gateway will then route this new download over NetB.

Once the download is re-established over NetB the gateway will *splice* the contents of the new download to the original download going to the client. Specifically, the gateway does a "packet re-writing" operation where the IP and TCP headers of the new download are mapped to mimic the correct headers of the original download. To the client it appears as if the original download resumed. This is shown in Figure 2. We refer to this technique as *Flow Migration through Splicing* or FloMiS.

Re-starting a download from a specific offset (utilizing a HTTP Range Request) is commonly used by some download managers as well as prior art [1], [7], [15]. HTTP Range Requests are typically supported for static HTTP content such as video and audio streams as well as image files. Further, to support this technique the gateway need only to hold state for a few additional packet header fields on a per flow basis, which we will provide more detail for in Section III.

With this optimization, the client will benefit from the fact that link and connection failures will be transparently handled by the gateway. The client further benefits since link failures will typically be detected at the gateway sooner than they could be detected at the client device. In Section IV, we provide additional support to show that client applications and devices must typically rely on TCP timeout mechanisms (on the order of minutes) to detect such failures where FloMiS can typically resume a download in about two round trip times.

How often do we need FloMiS?: FloMiS is invoked when there are failures in network paths, and so it is interesting to

also observe how often FloMiS is necessary. For our deployed gateway systems we have logged the events in which the operating system was able to detect the cellular modem connecting or disconnecting. We analyze this data which provides a conservative view of the impact that failures have on system robustness. We find that on average a non-transient link failure occurs every 118 to 158 minutes depending on the network provider, as seen in Table I. It is not unlikely for a user to be subject to such a failure given that in our prior work we have found that users on our coach buses were connected to our system on average for 53.23 minutes [12]. Further, these disconnections last on average for more than half a minute, which is likely due to the fact that the modem typically needs to reconnect to the cellular network.

C. Key Contributions

The primary contributions of this work are as follows:
1) We present the design and full implementation (as well as implementation challenges and experiences) for our FloMiS technique that provides flow migration support without imposing significant overheads that are require with prior multi-homed vehicular gateway architectures.
2) We detail the low-level TCP techniques used to transparently minimize the impact that non-transient link failures have on user experience.
3) We present a detailed evaluation with a focus on quantifying the impact that clients might experience in various usage scenarios.
4) We present a detailed analysis of how the network traffic is improved by FloMiS by analyzing our real vehicular usage traces (consisting of a snapshot of about 6 million flows) from our deployment that has been operational for more than 3 years.

II. Related Work

Due to the growth of mobile connectivity and multi-homed devices there is a large pool of prior art to consider. Mobile IP [3] and Mobile IPv6 [4] may be the most prevalent solutions that support client mobility. However, due to our requirements, multi-homing support is crucial for our system. At the transport layer a few notable solutions include: TCP-Migrate [20] and SCTP [5]. These solutions allows the transport protocol to migrate to alternative network path(s) when the primary path becomes unavailable. Transport layer solutions such as these require changes to the two end points of a connection. Implementing such end host changes are not feasible for our environment.

Another class of work has focused on multi-homed Internet devices where flow scheduling is often the primary challenge in static environments. For example, Thompson et. al. [21], schedule network flows in a multi-homed environment to the network that will provide the lowest RTT to a given destination.

Our work builds from prior multi-network vehicular gateway projects, namely MAR [16], PluriBus [13], and WiRover [12]. All three provide robust connectivity through a proxy and tunneling mechanism, however, in this work our goal is to provide robustness through more efficient mechanisms. Providing robustness to vehicular networks such as these can further benefit from knowledge of the historical network performance such that some link failures might be predicted [14], [18]. However, our FloMiS technique is still beneficial when unexpected link failures occur.

One component we leverage in this work is HTTP Range Request. Range Request have been suggested as a mechanism to offload cellular traffic to WiFi networks on mobile phones [7], [15]. Rahmati, et. al. [15], specifically use Range Requests to proactively migrate flows from the cellular interface to the WiFi interface of a mobile device. Other prior works [11], [7], primarily focus on selecting the best performing network through various mechanisms, however, such a problem can complement the problem of failure recover in multi-homed environments. Since these projects run on end hosts they can ignore many of the low level details that the Internet gateway is required to address for a successful implementation.

Snoeren, et. al. [19], present an interesting solution for end-host detection and recovery from server failures. This problem draws some parallels to the failure recovery problem FloMiS faces. However, as we will soon show, the key difference to take note of is that our FloMiS technique runs on the Internet gateway which introduces challenges that end-host solutions ignore such as TCP timestamps, selective acknowledgments, and receive windows as we will detail in Section III.

Finally, a vast pool of work focused on Delay Tolerant Networking (DTN) is complementary to our work. For example, the DieselNet project [22] provides Internet service in the face of frequent and lengthy connectivity outages. The techniques used to provide this connectivity would complement FloMiS's efforts to mask the impact of link failures. Due to the nature of such networks it is difficult to support high bandwidth streaming video applications or real-time VoIP applications. In the future, we might be able to incorporate some of the mechanisms used to cope with transient link failures [8], [7]. Similarly, delay tolerance is also important with opportunistic WiFi connectivity [9], [10] and thus also complements our system.

III. FloMiS Design

In designing FloMiS we posited that the most desirable property of flow migration is that it is completely transparent to the client device, client application, and end user. This enables an implementation without requiring any modifications to the devices real world clients use to connect to our deployments. To accomplish this transparency we believe that FloMiS must achieve three goals:
(a) To the client, FloMiS must appear to be utilizing the client's original network path.
(b) FloMiS must ensure the client receives the same content as they would have received before flow migration.
(c) FloMiS must complete a migration quickly as to not negatively impact the user's experience.
This section will focus on the first two stated goals and Section IV will focus on the third.

A. Flow Splicing

An outage on a path to a cellular endpoint is can be related to congestion, overloads, or failures on the cellular network

backhaul itself. Hence, in FloMiS, the gateway chooses to abandon an existing flow that is passing through one network path and initiates a request for the same contents through an alternate network path that is known to be functional. Once the contents begin to arrive on the new path, the gateway has to ensure that the client receives them as if they were part of the original flow. In particular, this assumes that the gateway maintains state of TCP level sequence numbers and acknowledgments of flows passing through it. Once new content is identified, it is merged into the original flow through packet re-writing operations — which we refer to as *splicing* of the flow. Splicing activities need only be invoked when an existing cellular network path experiences an outage and a migration to a new path is advantageous.

Unlike "split connection" designs where acknowledgments would be generated by the gateway, we rely on the client's original connection to maintain connection state. By forwarding the client's data acknowledgments we can ensure that duplicate acknowledgments and retransmissions are handled correctly.

When a failure is detected, FloMiS will create a new TCP connection with the server. Once the connection is created FloMiS takes a snapshot or checkpoint of the two flows for comparison. Using this checkpoint (CKP), FloMiS calculates a variety of offset values that are required to splice the two flows together. These offset values represent the relative difference between the two flows at the checkpoint. These offset values can then be applied to their respective header field values to "translate" the value from one flow to another. To quickly demonstrate this fact we will take a look at the offset value which is applied to the acknowledgment field of egress packets:

$$\begin{array}{lcl} new\_ack\_num & = & orig\_ack\_num + offset; \\ offset & = & (new\_ack\_ckp - orig\_ack\_ckp); \end{array} \quad (1)$$

Ideally we would like to generate the new acknowledgment number by simply adding an offset value to the client's original flow's acknowledgment number, as shown in Equation 1. This offset value should then be the difference of the new flow's checkpointed acknowledgment number less the client's original flow's checkpointed acknowledgment number. Using this formula we calculate a set of offset values, one applied to egress packets and the other applied to ingress packets, for the sequence, acknowledgment, and TCP timestamp header fields.

Occasionally, the value produced after applying the offset value may be incorrect. Such is the case when the original flow's checkpointed acknowledgment number is a duplicate acknowledgment, indicating packet loss. In this case FloMiS will issue the new download request based on this *dup ack* however, once the client receives retransmitted packet it may generate an acknowledgment for bytes that have not yet been sent via the new connection. During our development of FloMiS we found that forwarding such an acknowledgment resulted in a frozen TCP connection. Thus, the new acknowledgment value is set to the minimum of either the translated value or the maximum expected acknowledgment value, which is based on the bytes received over the new flow.

TCP Selective Acknowledgments: TCP SACK values are handled similarly to the TCP acknowledgment values since frozen TCP connections can also result from inconsistencies in the selective acknowledgments ranges. Further, the translation of both acknowledgment and sequence numbers are subject to caveats related to higher layer protocols and will be detailed shortly.

TCP Timestamps: The use of TCP timestamps are defined in RFC 1323 [6] (TCP Extensions for High Performance) which details that embedding a timestamp value into a TCP options field can improve TCP's ability to estimate a connections RTT. The specification does not require these timestamp values to be based on the system's clock.

During our implementation testing we observed that occasionally the content server would retransmit packets that the client had already acknowledged. What we found was that the client was generating TCP timestamp values differently such that in relation to our gateway system the values appeared to be significantly in the past. The content server would then silently discard these acknowledgments and eventually retransmit the data packets. Simple translations of this field similar to the other translated TCP fields prevents this issue, however, producing more accurate timestamp estimates is an area for future improvement.

Receive window size: The TCP receive window field is an aid to prevent the sender from overwhelming the receiver. However, the meaning of the value in this field may change between the two flows. This is the result of a TCP extension that is defined in RFC 1323 [6], which defines a window scaling factor that is exchanged in TCP SYN packets. For each flow, FloMiS records this scaling factor and after flows are spliced together the value in the TCP window field is adjusted accordingly. This translation (which is only applied if the scaling factors are not equal) requires only a few bit shift operations applied in the following way:

$$\begin{array}{lcl} new\_window & = & (orig\_window << orig\_scale) \\ & & >> new\_scale; \end{array} \quad (2)$$

B. Handling HTTP

To support HTTP content our FloMiS design benefits if a particular piece of content supports Range Requests. HTTP Range Requests are defined in RFC 2616 [2] (HTTP/1.1) and they provide a mechanism through which a client can efficiently restart a paused download from a specific offset. In the event a client's download is impacted by a link failure the client's application, device, or the client themselves may re-issue the download request. The client's re-issued request is likely to contain a Range Request for the unreceived portions of the desired content. Our FloMiS design aims to mimic the client's behavior by pro-actively sending a Range Request on behalf of the client. The advantage of this approach is that the gateway can detect and react to link failure much sooner that the client itself could have, as we will discuss further in Section IV.

When FloMiS is applied to a download flow it will establish a new connection and issue a HTTP Range Request based on the last acknowledged byte sent by the client, as seen in Figure 3. This TCP acknowledgment value can be used to determine a corresponding offset for the HTTP content. Typically, since TCP acknowledgments account for all TCP payload bytes, any prior content downloaded on this particular flow as well as any HTTP header bytes should be factored into this calculation. The calculation will then yield the next byte

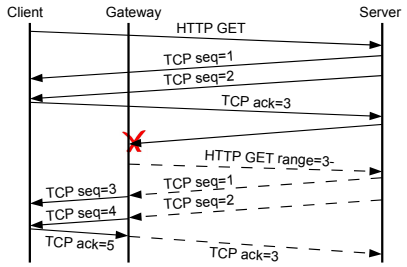

Fig. 3. **The flow of messages when FloMiS is applied to a HTTP download. FloMiS restarts the download based on the last TCP ACK sent by the client. FloMiS then translates each packet's TCP sequence numbers to appear to be part of the original flow.**

of content that the client expects to receive and thus a Range Request is issued starting at this byte.

FloMiS will then forward the content bytes onto the client by splicing the new flow to the original flow. Eventually, either one of the end points will close the TCP connection at which point FloMiS can translate these packets to ensure that both the new and original TCP connections are closed properly. To facilitate connection tear down immediately after the download finishes, FloMiS can modify the *Connection* field in the HTTP header. Often this field specifies that a connection should remain open, using the *keep-alive* field value, but if FloMiS were to modify this field's value to *close* the server will transmit a TCP FIN/ACK with the last byte of content data. This practice quickly closes connections so that any FloMiS flow state may be released.

HTTP header removal: When forwarding packets to the client, FloMiS must first ensure that any additional HTTP headers are removed and only the expected content bytes are forwarded to the client. However, the act of removing HTTP header bytes will skew the previously calculated sequence and acknowledgment offset values. Thus the offset values should be updated to reflect the discarded bytes. Offset values are then stored in a list with the byte ranges to which they are applicable. This is an important detail since packet retransmissions and duplicate acknowledgments may require an older version of the offset value to produce the correct translated values.

This issue of removing unexpected HTTP header bytes is compounded when the HTTP server does not transfer content in one contiguous range of bytes. In this case the HTTP server splits the content file into multiple smaller segments called chunks. Prepending each chunk is a smaller header which provides the size of the chunk that follows.

In such a case FloMiS would have to modify the chunk headers to be consistent with those of the original flow. If the original flow was not chunk encoded then FloMiS would simply remove all chunk headers. However, chunk encoding is more typical of dynamically generated content and is thus rare for static, Range Request supported content.

Replay HTTP GET Request: To restart a HTTP download the gateway stores the client's original GET request. This HTTP GET header is then modified by either changing an already existent requested range field or by inserting a range field if one does not already exist.

Cache control: Beyond modification to the GET request's range field, FloMiS must ensure that the new download will access the same content as the client's original download. This is achieved using HTTP cache control mechanisms. HTTP caches often utilize either the "Last-Modified" date or the "ETag" HTTP header fields to check to see if a particular piece of content stored locally differs from what is actually stored on the server. Despite the fact that the gateway never caches any HTTP content these mechanisms can enable FloMiS to ensure that a particular piece of content has not changed since the client sent the original request.

In the case that the content has changed from that of the original request FloMiS will cancel the flow's pending migration.

Non-Range Requests: Up to this point we have focused on content supported by Range Request. However, for static content that is not supported by Range Requests the gateway might be able to restart a download from the beginning of a request and discard downloaded bytes until the new download reaches bytes that the client has not yet received. Once the new download has caught up to the point where the original download experienced a failure FloMiS can complete the migration and begin splicing the flows together.

Since, this method may consume a significant amount of waste it may not be very practical. However, in Section IV-D, we find that on average non-Range Request flows are much shorter than flows supported by Range Request. In such a scenario it is still possible for a large download to be unsupported by range request in which case FloMiS risks the generation of a significant amount of waste if the download was well established at the point of failure. Further, if the download was well established at the point of failure it may take a significant amount of time for the new download to catch up during which the client's original TCP connection may timeout. We profile this connection timeout delay for a variety of devices in Section IV.

Connection Resumption: In either the case of Range Request and Non-Range Request flows there is a gap between a link failure and FloMiS splicing the original flow with the new flow. With Range Request flows this gap of time is based on the time to complete the TCP handshake for the new flow. During this period of time it is possible that the original link and thus the original flow recover. Therefore, FloMiS first checks to determine that the original flow has not resumed prior to finalizing the splicing of the flows. If the original flow has resumed FloMiS cancels the pending migration and splicing and closes the new connection to the server. If the original flow resumed after which FloMiS has already migrated and spliced the flows together we currently choose to continue the download using the migrated flow. A TCP reset is then sent to the content server over the original connection terminating that flow from the content server's perspective. By allowing the resumption of the original connection, our FloMiS technique is typically only utilized when non-transient link failures occur.

End-to-Endness: FloMiS breaks the end-to-end principle [17]. However, we believe this is a penalty worth paying for the seamless experience that users have across network outages. Significant efforts are made to ensure that FloMiS only acknowledges and forwards content bytes that the client would have received on the original flow. For our current

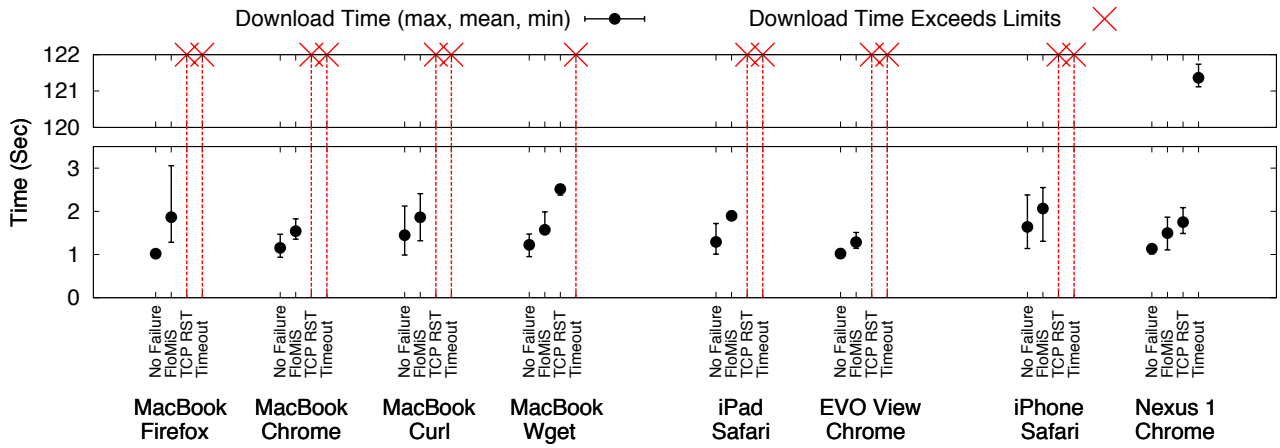

Fig. 4. **The amount of time to download a 96 KB image with various failure recovery mechanisms compared to the amount of time to download the same file without the occurrence of a failure event.**

prototype we focus on HTTP content since TCP port 80 traffic constitutes 97% of total traffic volume observed on our vehicular deployments.

C. Implementation

The FloMiS code is split into two main parts: code that manages the network data path and code that manages the control path. The control path handles the task of re-establishing a network connection on the client's behalf. When the software is started, the control path creates a pool of worker threads that will handle migration requests on behalf of the client device.

In the case of a link failure, the gateway will issue a request to migrate an impacted flow and one of these threads will be assigned the task of re-establishing a network connection and restarting the download from the point of failure. Before the data path is allowed to complete the migration and splice the flows together the worker thread verifies the server's response and if no errors are detected the worker thread notifies the data path with the new flow's tuple information. The data path can then splice the two flows together. This procedure can be repeated multiple times for a single flow that is impacted by subsequent failures.

IV. Evaluation

To evaluate the performance of our FloMiS system and to support our motivating claims we have installed our implementation on an Intel Atom-based platform which serves as our gateway. This hardware platform has a 1.6 GHz CPU with 2 GB of RAM and a 40 GB solid state hard drive. Our software is then run atop Ubuntu Linux 12.04 and the 3.2.0 Linux kernel. To connect various client devices the gateway exposes a local WiFi network. For this work we assume that the maximum bandwidth available to the gateway is constrained by the cellular backhaul networks currently connected to the system. To provide reproducible measurements we use a wired backhaul that is constrained to 2 Mbps with 100 milliseconds of delay.

A. Device Behaviors

In our vehicular environment, link failures are bound to occur (Table I). Such failures can produce a very frustrating experience for the end user which require users to reload web pages and endure audio/video buffering delays. To better understand how various devices and applications respond to link failures we studied a variety of test cases:
(1) *No Failure:* In this case a download is started and the download finishes without a failure. This test provides a base against which we can compare failure recovery techniques.
(2) *FloMiS:* In this approach, once the gateway detects a link failure FloMiS is applied until the completion of the download.
(3) *TCP RST:* In this approach, once the gateway detects a link failure a TCP RST is sent to the client.
(4) *Timeout:* In this case, the gateway makes no special effort to signal client of failures or transparently recover from failures. This is the typical situation clients would find themselves in if the gateway did not explicitly implement any link failure recovery mechanisms. In all failure cases, the gateway upon detecting a link failure will route new flows over alternative paths.

Download Test Results: To test the impact of cases(2)-(4) on user experience in a basic web download scenario we download a 96 KB JPEG image that is embedded into an HTML page. When the client device loads that web page, and thus the image, the gateway induces a failure event after 60 KB of the image file has been acknowledged by the client.

In Figure 4, we display the maximum, mean, and minimum download times from running 5 trial sets for each test case on each device and application. We found that the image took anywhere from 1.02 seconds to 1.64 seconds on average to download in the absence of a failure (the *No Failure* test case). The client device's typical failure detection mechanism (i.e., *Timeout* due to the fact that it is unaware of failures on the backhaul networks) resulted in the download stalling on most devices. For this test case we waited a maximum of 5 minutes before terminating the test, since we felt this would provide a real user ample time to either ignore the failed download or try to restart the download manually. However, the Nexus 1 device seemed to timeout after a period of only 2 minutes after which the browser issued a new request for the remaining portion of the image.

Contrasting this single device's ability to detect and recover from the failure event is our *FloMiS* technique's results. For

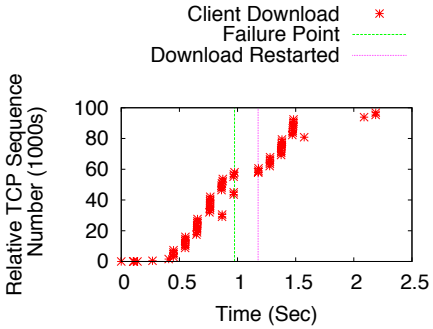

Fig. 5. **Example image download using an iPad when our FloMiS technique is applied. TCP sequence number of the download flow and the failure event that occurs after 60 KB of the image has been downloaded.**

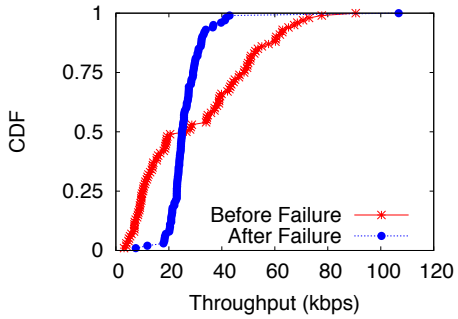

Fig. 6. **The CDF of the throughput achieved by 100 concurrent audio streams before and after a link failure.**

every device tested we found that downloads only took a few hundreds of milliseconds longer than the base case where downloads were failure free. The two alternatives for flow recovery (namely *Timeout* and *TCP RST*) did not really work for most platforms and browsers. When it worked, it was usually a slower recovery mechanism than what FloMiS was able to achieve.

It was interesting to observe that most devices and browsers did not respond in the *TCP RST* case with new connections and requests. Typically, sending a TCP reset will cause the client's socket connection to close with the error value that translates to: "Connection reset by peer." It is then up to the client application to interpret and handle this socket error. In most browsers we observed that the download was simply terminated and abandoned. We did however find two exceptions: the command line tool Wget and again the Nexus 1. In both cases after the reset was received a new connection was formed and a Range Request was issued by the client for the remain portion of the image file.

B. Failure Recovery Delay

To better understand the image download results we selected a particular trial from the iPad data set where our FloMiS technique was applied. For this trial we observed the greatest disparity between the total amount of time to download the image (1.9 seconds for this trial) to the base case where no failure occurred (1.0 second). When we graphed out the download's progress (Figure 5) we observed a gap in the middle of the download between the point in time when the failure occurs and the time in which FloMiS is able to resume the download on behalf of the client. The packet trace indicates the failure event occurred at 0.97 seconds and the download is restarted at 1.2 seconds (these points in time are indicated by the vertical lines in the plot). We refer to this period of time between the failure event and when the download is resumed as the *failure recovery delay*. In this particular instance the *failure recovery delay* of our FloMiS technique was only 0.23 seconds. Since the network delay of our testing setup was 100 milliseconds much of the failure recovery delay can be attributed to the overhead of setting up a new TCP connection with the content server. For this trial we observe that the *failure recovery delay* provides a better metric to judge the various technique than overall download time since in this case the overall download time can be skewed by packet retransmissions that occurred near the end of the download.

In further evaluating our exception cases, we find that Wget's average *failure recovery delay* for the *TCP RST* case was 1233 msec. The Nexus 1's average *failure recovery delay* for the *TCP RST* case was 275 msec and for the *Timeout* case it was 120405 msec (or roughly 120.4 sec). When compared to the Nexus 1's *FloMiS* case where the *failure recovery delay* was 223 msec. Signaling the client with a TCP RST may seem to be an attractive solution, however, it is important to note that in our environment it is not possible to reconfigure client devices and we thus must face the challenges that all legacy devices present.

Streaming Video Results: To further emphasize FloMiS's ability to mask failures from end users we conducted a set of streaming video experiments. Using both an Android tablet and an iPad we streamed an HTML5 video to both devices. Using Javascript APIs we are able to monitor the video's playout performance as well as the progress of video buffering. Similar to our image download results we found that both *Timeout* and *TCP RST* tests resulted in terminated downloads and eventually the video playout would stall after all buffered content was exhausted. In contrast the video playout was uninterrupted for all trials when our *FloMiS* technique was applied.

The uninterrupted playout can be attributed to the video player's behaviors. We observed that the video player typically buffered roughly 489.4 msec of content before video playout began. Our *FloMiS* technique had an average *failure recovery delay* of 218 msec which was easily masked by the video player's buffered content. For failures induced before video playout began we observed that video playout was typically delay 200 msec when compared to failures that were induced after video playout had already begun.

C. Load Test

To evaluate how our FloMiS technique scales we imagined a situation where 100 passengers are all streaming audio at the same time. In this scenario the gateway has two available interfaces: a NetA with bandwidth of roughly 1.5 Mbps and a NetB with bandwidth roughly 0.24 Mbps. Imagine that the gateway mapped connections onto these available network interfaces such that 50 flows were on each network. For this scenario the low bandwidth on the NetB network might be an indication that the vehicle is currently in an area where that network has poor signal strength. If NetB experiences a link failure then all 50 flows will require migration to NetA.

In Figure 6, we plot the CDF of flow throughput for all 100 flows before and after the link failure. Since the 100 flows are

Data set size	6,003,372 URLs
URLs with Range Request Support	2,172,215 URLs
AVG Content Size for URL w/ Range Request Support	1,073,301 Bytes
AVG Content Size for URL w/o Range Request Support	31,607 Bytes

TABLE II. **Based on a snapshot of more than 6 million user generated URLs we find that over 1/3 of all requests support HTTP Range Requests.**

split across the two networks before the link failure we see a bi-modal behavior in per flow throughput. However, after the link failure all NetB flows are migrated to the NetA link.

When we monitored the CPU usage of the system during the duration of this scenario we found that FloMiS on average only increased CPU usage by 1.325%. The average system memory usage during the duration of the scenario was increased by 5.52%, which translates to roughly 110 MB. The per packet processing delay (the time from which our code receives a packet to the time it wrote the packet out) on average for all processed packets was 78 μsec. The average *failure recovery delay* for all migrated flows was 218 msec.

Overall, we feel further optimizations can be applied to our implementation, however, given that the load used in this experiment exceeds the typical load we observed in our deployed systems we feel confident that our FloMiS technique will perform well.

D. FloMiS' Impact

The FloMiS design presented is focused to benefit HTTP content that is supported by Range Request. To better understand the impact FloMiS has in our particular environment we studied a collection of HTTP GET requests generated by real users of our deployed systems. This data set consists of a subset of logged requests sampled from user activity over a two year period of operation and consists of more than 6 million requests. Table II, highlights some of the characteristics of this data set.

We find that over 1/3 of all requests support Range Requests. We also find that typically larger content is supported by Range Requests, where the average size of supported content is more than 1 MB compared to 31.6 KB for unsupported content, as seen in Table II. In fact, the 1/3 of requests with Range Request support account for a total of 2.29 Terabytes of data while the other 2/3 of requests account for only 89.3 Gigabytes of data (which is less than 4% of the total amount of data represented by this data set). In our previous study [12], we observed that TCP port 80 traffic accounted for nearly 97% of traffic volume suggesting that FloMiS would benefit nearly 93% of all traffic volume generated.

V. Conclusion

We have introduced FloMiS, a technique necessitated by challenges we have faced over the past few years of running a vehicular Internet connectivity service. Providing clients with a robust, uninterrupted Internet connection on their legacy devices is mutually beneficial as client traffic provides a wealth of insight into the vehicular connectivity environment.

FloMiS exploits the static characteristics for the majority of content observed in our deployments. Further, the overheads of our FloMiS technique are only incurred by network traffic impacted by non-transient link failures. In contrast prior solutions rely heavily on tunnel and proxy architectures that imposes significant overheads to all traffic, even when no failures occur.

We believe that our FloMiS technique enables an efficient solution to improving user experience for all legacy client devices in multi-homed vehicular gateway systems. Further, as the popularity of streaming video and audio grow we believe the advantages of FloMiS will only become more impactful.

References

[1] Gnu wget. http://www.gnu.org/software/wget/.

[2] Hypertext transfer protocol – http/1.1. http://tools.ietf.org/html/rfc2616.

[3] Ip mobility support for ipv4, revised. http://tools.ietf.org/html/rfc5944.

[4] Mobility support in ipv6. http://tools.ietf.org/html/rfc6275.

[5] Stream control transmission protocol specification. http://www.ietf.org/rfc/rfc2960.txt.

[6] Tcp extensions for high performance. http://tools.ietf.org/html/rfc1323.

[7] Aruna Balasubramanian, Ratul Mahajan, and Arun Venkataramani. Augmenting mobile 3G using WiFi. In *MobiSys*, 2010.

[8] Aruna Balasubramanian, Yun Zhou, W. Bruce Croft, Brian Neil Levine, and Aruna Venkataramani. Web search from a bus. In *CHANTS*, 2007.

[9] Vladimir Bychkovsky, Bret Hull, Allen Miu, Hari Balakrishnan, and Samuel Madden. A measurement study of vehicular Internet access using in situ Wi-Fi networks. In *Mobicom*, 2006.

[10] Jakob Eriksson, Hari Balakrishnan, and Samuel Madden. Cabernet: vehicular content delivery using wifi. In *MobiCom*, 2008.

[11] Vittorio Ghini, Stefano Cacciaguerra, Fabia Panzieri, and Paola Salomoni. A hidden proxy for seamless & abc multimedia mobile blogging. In *NIME*, 2006.

[12] Joshua Hare, Lance Hartung, and Suman Banerjee. Beyond deployments and testbeds: experiences with public usage on vehicular wifi hotspots. In *MobiSys '12*.

[13] Ratul Mahajan, Jitendra Padhye, Sharad Agarwal, and Brian Zill. High performance vehicular connectivity with opportunistic erasure coding. In *USENIX ATC'12*.

[14] Anthony J. Nicholson and Brian D. Noble. Breadcrumbs: forecasting mobile connectivity. In *MobiCom*, 2008.

[15] Ahmad Rahmati, Clayton Shepard, Chad Tossell, Lin Zhong, Philip Kortum, Angela Nicoara, and Jatinder Singh. Seamless tcp migration on smartphones without network support. *IEEE Transactions on Mobile Computing*, 2013.

[16] Pablo Rodriguez, Rajiv Chakravorty, Julian Chesterfield, Ian Pratt, and Suman Banerjee. Mar: A commuter router infrastructure for the mobile Internet. In *MobiSys*, 2004.

[17] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 1984.

[18] Sayandeep Sen, Jongwon Yoon, Joshua Hare, Justin Ormont, and Suman Banerjee. Can they hear me now?: A case for a client-assisted approach to monitoring wide-area wireless networks. In *IMC*, 2011.

[19] Alex C. Snoeren, David G. Andersen, and Hari Balakrishnan. Fine-grained failover using connection migration. In *USITS'01*.

[20] Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *MobiCom '00*.

[21] N. Thompson, G. He, and H. Luo. Flow scheduling for end-host multihoming. In *INFOCOM '06*.

[22] Xiaolan Zhang, Jim Kurose, Brian Neil Levine, Don Towsley, and Honggang Zhang. Study of a bus-based disruption-tolerant network: mobility modeling and impact on routing. In *MobiCom*, 2007.